FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Modernizace webové aplikace pro online aukce
Student:	Jan Charamza
Vedoucí:	Ing. Jiří Mlejnek
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Analyzujte požadavky na systém pro online aukce. Proveďte rešerši existujících obdobných systémů a proveďte jejich srovnání ve vztahu k analyzovaným požadavkům. Navrhněte a implementujte webovou aplikaci pro procházení aukcí a jejich položek, přidávání limitních příhozů (příhozy před živou aukcí), účastnění se a dražení v živé aukci a správu uživatelů (uživatelův profil, přihlašování a registraci).
Systém důkladně otestujete, zdokumentujte a připravte pro produkční nasazení.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 4. února 2022 v Praze.

Bakalářská práce

MODERNIZACE WEBOVÉ APLIKACE PRO ONLINE AUKCE

Jan Charamza

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jiří Mlejnek
11. května 2022

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především panu učiteli ing. Jiřímu Mlejnkovi, že na všechny mé dotazy ohledně bakalářské práce odpovídal pohotově a že mi dal druhou šanci, i když nemusel. Také bych chtěl poděkovat mým blízkým, že se mnou měli v tomto stresujícím období trpělivost.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na jejímž základě se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 11. května 2022 .....

Abstrakt

Tato bakalářská práce řeší postup tvorby webové aplikace podle správných zásad softwarového inženýrství. Cílem práce bylo vytvořit plnohodnotnou webovou aplikaci pro provoz aukčního systému online umožňující propojení živé sálové aukce s příhozy přes online prostředí. Webová aplikace byla implementována jako oddělený frontend a backend, disponující testy a dokumentací, a následně nasazena do cloudového prostředí. V závěru jsou uvedené kroky, jak by mohla být výsledná implementace stabilnější s budoucími aktualizacemi.

Klíčová slova Livebid.cz, Aukční systém, Webová aplikace, Přihazování, Umění, Numismatika

Abstract

This bachelor thesis deals with the process of creating a web application according to the correct principles of software engineering. The aim of the thesis was to create a full-fledged web application for running an online auction system, allowing the connection of a live hall auction with bids via an online environment. The web application was implemented as a separate frontend and backend, having tests and documentation, and then deployed to a cloud environment. Finally, steps are given on how the final implementation could be more stable with future updates.

Keywords Livebid.cz, Auction system, Web application, Bidding, Arts, Numismatics

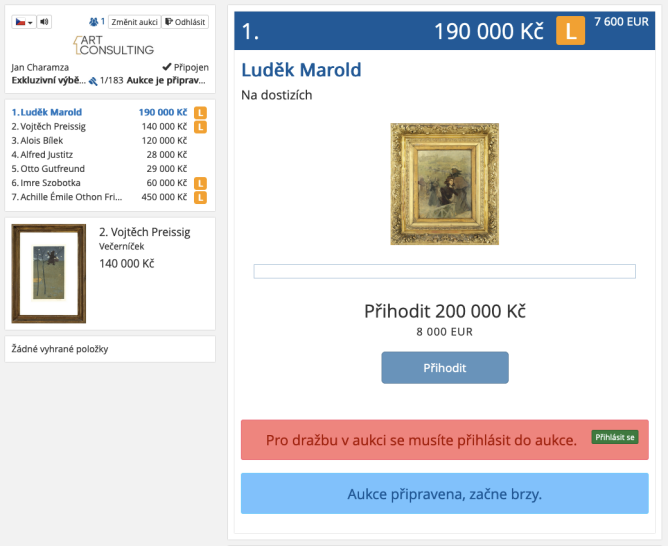
Seznam zkratek

HTML	Hyper Text Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
TS	TypeScript
JSON	JavaScript Object Notation
XML	Extensible Markup Language
DOM	Document Object Model
K8s	Kubernetes
CI	Continuous Integration
API	Application Programming Interface
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
SPA	Single Page Application
SSR	Server Side Rendered
CSR	Client Side Rendered
SSG	Static Site Generation
UC	Use Case

Kapitola 1
Úvod

Umění je jedno z nejstarších odvětví na zemi. Nalezené jeskynní malby se datují zpátky až 40 tisíc let před naším letopočtem.[2] Od té doby se postupně rozvíjely novější techniky maleb, sochařství a celkově umělecké tvorby. S tím se již v dobách starověkého Říma začali objevovat i sběratelé těchto uměleckých kousků hnaní touhou vlastnit pěkné věci. V dnešní době existuje spousta druhů umění a sběratelských předmětů, od fyzického hmatatelného, až po digitální, čistě virtuální. Běžný způsob pro získání uměleckých předmětů a obecně věcí, které jsou jedinečné, je licitovaná aukce (dražba), při které dopředu není známa výsledná cena a předmět vyhraje osoba s nejvyšším příhozem, čímž se maximalizují zisky pro prodávajícího.

Livebid.cz je online aukční platforma pro dražení v sálových aukcích (nebo i jen čistě online aukcích) z pohodlí domova. Jsou zde vystavené nadcházející aukce, kde si dražitel může v klidu procházet vystavené aukční položky online a označit si ty, které ho zajímají, či na ně rovnou vložit tzv. limit, který za ně bude během aukce přihazovat až do nastaveného limitu. Případně mohou přihazovat až přímo přes webové rozhraní během konání aukce, kde se při sálové aukci projevují i příhozy proti dražitelům fyzicky na sále. Při výhře dražené položky se další kroky komunikují s aukčním domem, platba a předání položky se provádí mimo systém.



Obrázek 1.1 Ukázka webového rozhraní pro živou aukci

Aukce jsou obsluhovány operátory. U online aukce stačí, když operátor aukci spustí a kontroluje, že je průběh aukce v pořádku. V případě sálových aukcí musí ideálně 2 operátoři přijet na aukci, kde se starají o online livestream licitátora a komunikaci s licitátorem. Hlásí licitátorovi změnu nejvyšší ceny na internetu a naopak zaznamenávají nejvyšší nabídku na sále. Zákazníky jsou tedy aukční domy, platící za vystavení svých aukcí. Pro uživatele je web zcela a zdarma bez reklam.

Platforma byla vyvíjena od roku 2013, kdy na ní pracovali 2 studenti střední školy. Na kódu pracovali až do poloviny roku 2021, přičemž většina práce po roce 2017 představovala spíše menší změny. Systém je rozdělen na 3 části: veřejná část webu, živá aukce a administrace.

Platforma funguje do dnešní doby stabilně, nicméně s připravovanou budoucí expanzí celého produktu je potřeba kód refaktorovat a připravit pro rychlé iterace nových funkcí. Majitel webu se tedy rozhodl přepsat celý web do moderních technologií.

1.1 Cíle práce

Hlavním cílem práce je vytvořit systém pro provoz aukcí propojující skutečné sálové aukce a online prostředí. Uživatelé budou moci procházet aukční položky, přidávat limity na položky a účastnit se živé aukce s přihazováním na položky, procházet limitované, vyhrané a oblíbené položky, či přidávat výrazy do hlídacího psa. Na živou aukci bude dohlížet aukční operátor, který může do systému přidávat příhozy ze sálu, upřednostnit příhozy a plně ovládat směr průběhu aukce.

Kromě implementační části bude cílem provést detailní analýzu požadavků systému, otestovat nové verze softwaru a zdokumentovat navrženou architekturu pro vývojáře a instalační příručku pro systémové administrátory.

Kapitola 2
Analýza

Před samotnou implementací a výběrem použitých technologií je za potřebí vytvořit analýzu aktuálního stavu a představ zákazníka ohledně nové implementace.

Při práci na této kapitole bylo často komunikováno s vlastníkem projektu a aukčními operátory ohledně pojmů, používání aktuálního systému a návrhů nadcházejícího systému. Komunikace k nim směřovala při jakýchkoliv nejasnostech.

2.1 Použitá terminologie

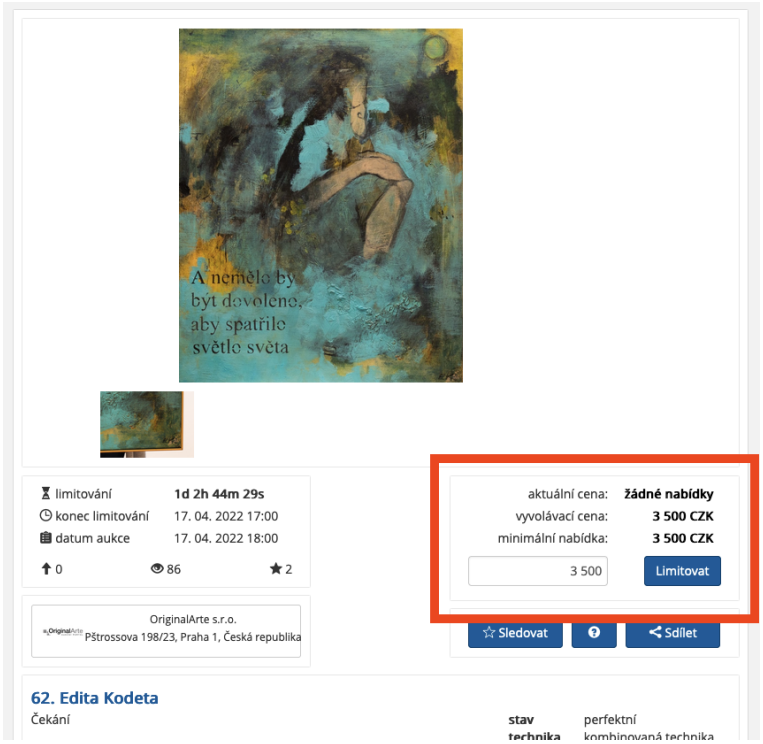
Pro konzistentní komunikaci a porozumění zákazníkovi je vhodné si nejprve nastudovat a dohodnout používanou terminologii v rámci komunikace požadavků na nadcházející systém. Následující seznam sestává ze všeobecně i pouze interně používaných pojmů.

2.1.1 Aukční pojmy

Dražba - Dražba je způsob prodeje, kde cena začíná na vyvolávací částce a zájemce s nejvyšší nabídkou vyhrává draženou položku. Průběh aukce je vedený licitátorem.

Příhoz - Příhoz je zvýšení nabídky dražitelem na další stupeň podle tabulky příhozů, nebo navýšení o konkrétní částku.

Limitování - Nastavení limitu je způsob nastavení finančního ”stropu”na zvolenou položku. Uživatel určí, jak vysokou částku je ochoten maximálně zaplatit a systém pak za něj bude během živé aukce do této částky přihazovat proti jiným dražitelům. Nastavená částka dražitelova limitu není pro ostatní dražitele viditelná. Viditelná je až při dosažení / překonání této částky. Výška limitu není známa ani aukčnímu domu, aby nemohl této informace využít pro maximalizaci ceny položky, kdy při vysokém limitu by přihazoval proti němu pouze jeden krok od maxima.



Obrázek 2.1 Okno pro přidání limitů před zahájením dražby

Tabulka příhozů - Tabulka příhozů udává kroky zvyšování ceny při dražbě. Aukční tabulku si nastavuje každý aukční dům sám podle vlastních preferencí. Limity i příhozy respektují utváření cen podle ceny kroku v určeném rozmezí cen.

Tabulka příhozů Více

Od	Do	Krok
0	5 000	100
5 000	10 000	500
10 000	50 000	1 000
50 000	100 000	2 000
100 000	10 000 000	5 000
10 000 000	-	10 000

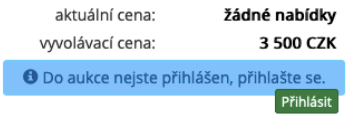
Obrázek 2.2 Příklad tabulky příhozů s šesti cenovými rozmezími

Aukční přirážka - Ceny během dražby jsou udávány bez aukční přirážky, kterou musí uhradit dražitel na konci dražby při úhradě nejvyšší nabídnuté částky. Často se jedná o procentuální přirážku (obvykle zhruba okolo 20 %) definovanou aukčním domem.

Odklepnutí aukčního kladívka - Odklepnutím aukčního kladívka licitátorem aukce končí přijímání dalších nabídek na položku a vlastnictví předmětu aukce přechází po zaplacení do vlastnictví nového majitele.[3]

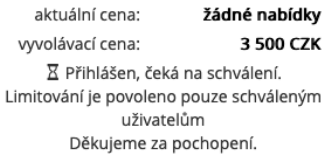
Maření dražby - V případě, že dražitel přihlášený do aukce neuhradí položky, na které vložil nejvyšší příhoz, dopouští se maření aukce a v takovémto případě nepřechází vlastnictví draženého předmětu do rukou dražitele.[3]

Přihlášení do aukce - Uživatelé Livebidu se musí do aukce přihlásit, aby se jí mohli účastnit. K tomu musí být dostatečně ověřeni (včetně dokladu totožnosti) pro vkládání limitů a příhozů.



Obrázek 2.3 Přihlášení do aukce

Schválení do aukce - Jestli příhlášení do aukce proběhne úspěšně určují pořádající aukční domy, které mohou uživatele v případě nejasností zamítnout.



Obrázek 2.4 Uživatel čeká na schválení do aukce

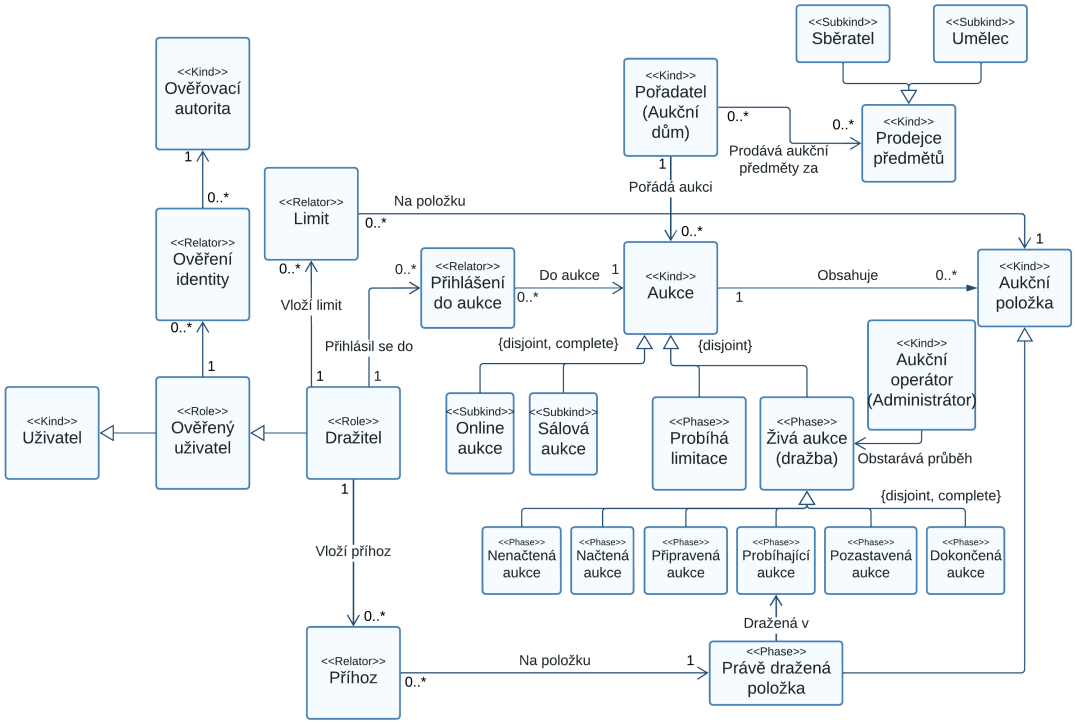
Sálová aukce - Aukce probíhá fyzicky ve vyhrazených prostorách s přítomností licitátora a dražitelů na sále. Nachází se zde i aukční operátor ze strany Livebidu, který zaznamenává příhozy udělané na sále a opačně na sále přihazuje za lidi dražící online. Fyzicky se mohou na sále i vyskytovat dražené položky, které si mohou dražitelé před začátkem aukce procházet (podobně jako na Livebidu ve virtuálním prostředí).

Obrázek 2.5 Ukázka sálové aukce. Uprostřed stojí licitátor přiřazující příhoz dražiteli. Nad licitátorem se nachází projektor zobrazující aktuální položku a její cenu.

Online aukce - Online aukce se nachází čistě v online prostředí ve webové aplikaci Livebid.cz. Průběh takovéto aukce je oproti klasické sálové variantě mnohem levnější, protože se nemusí

pronajímat prostory pro konání aukce, nemusí se platit licitátor a pro aukční operátory není potřeba hlídat příhozy na sále. Aukční operátor zde hlídá pouze dodržování hladkého průběhu aukce.

Živá aukce - V kontextu Livebidu znamená živá aukce právě probíhající dražbu. Rozlišuje se tak od samotného slova *aukce*, které znamená celou událost aukce od zveřejnění data aukce, zveřejnění katalogu s položkami, začátku limitování položek, konce limitování položek (většinou zhruba hodina před začátkem dražby), živé aukce (v rámci které probíhá dražba položek), až po placení závazků dražiteli a odesílání / předávání položek.



Obrázek 2.6 Doménový model vytvořený podle specifikace OntoUML

2.1.2 Role

Dražitelé - Uživatelé, kteří se mohou volně registrovat na webu Livebidu. Pokud provedou všechna potřebná ověření (ověření emailu a dokladu totožnosti), mohou se účastnit dražení v aukcích online. Stále jim však může být zamítnuta účast na konkrétní živé aukci přímo aukčním domem, pokud mají s uživatelem nějaký problém.

Důležité je mít na paměti, že v případě sálové aukce mohou být dražitelé i přímo fyzicky na sále bez nutnosti registrace na Livebidu (ne pouze online přes Livebid).

Aukční dům (dražebník) - Aukční dům se stará o příjem položek do aukce (kurátor) včetně jejich uskladnění do konce aukce, schvalování dražitelů podle jejich dokladů do aukce, o celkový průběh sálových aukcí, vystavování faktur a odesílání prodaných aukčních položek výherním dražitelům.

Z pohledu Livebidu jako služby je zákazník právě aukční dům. Aukční domy jsou schopny nahrát nové aukce bez nutnosti jediné návštěvy administrace na webových stránkách Livebidu.

Důležité informace po předchozí domluvě zašlou na helpdesk emailem, který zpracuje administrátor Livebidu podle formátu zaslaných dat. Každopádně je pro aukční domy přístup do administrace vhodný pro kontrolu stavu jejich aukcí (včetně různých metrik) a schvalování dražitelů do nadcházející aukce.

Licitátor - Osoba licitující aukci. Zahajuje dražbu, kontroluje běh dražby, vyzívá účastníky k dražbě a ukončuje dražbu. Při odklepnutí dražebního kladívka licitátorem se ukončí příjem nabídek na aktuálně draženou položku a zároveň dojde k převodu dražené položky na dražitele s nejvyšší nabídkou, který je povinnen uhradit nabídnutou částku včetně aukční přirážky aukčního domu.[3]

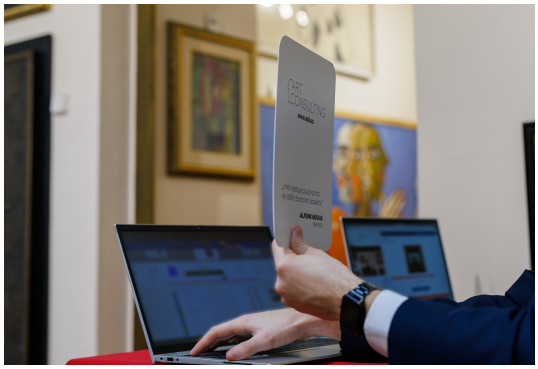
Během dražby určuje, komu položka připadne v případě, že se přihlásí pro stejnou cenu naráz (např. dražitel na sále a dražitel na Livebidu, případně více dražitelů na sále).

Umělec - Umělecké položky mohou do aukce poskytnout buď přímo umělci (v případě zesnulých jejich příbuzní), nebo sběratelé.

Sběratel - Poskytuje aukčnímu domu položky do aukce ze své osobní sbírky. Komunikaci s nimi řeší přímo aukční domy.

Aukční operátor - Aukční operátor se stará o běh aukce ze strany Livebidu. V případě čistě online aukce se stará o spuštění aukce a její kontrolu, že vše probíhá v pořádku, bez jakýchkoliv technických problémů.

V případě sálové aukce musí aukční operátoři přijet na místo konání aukce a spouštět aukci tam (s sebou mají navíc pro stabilitu aukce zabalené power banky a mobilní internetový hotspot). Navíc také musí připravit kameru pro livestream licitátora do okna pro dražitele na Livebidu. Dále při průběhu aukce musí do systému zanášet příhozy ze sálu a obráceně musí na sále přihazovat za uživatele, kteří draží online.



Obrázek 2.7 Aukční operátor přihazuje na sále pro online dražitele

Sálové aukce přinášejí prostor pro spoustu lidských chyb jak z pozice aukčního operátora, tak jiných zúčastněných stran. Proto je také potřeba mít připravené systémové nástroje pro opravy situací zaviněných lidským faktorem.

Administrátor - Má pod kontrolou celý systém Livebidu. Nejdůležitější vlastností je možnost vystavovat nové aukce a přidávat do katalogu položky, včetně obrázků, podle specifikace aukčního domu.

Administrátor může v systému vytvářet účty pro aukční domy a provádět všechny operace, co oni, ale navíc může například tzv. ”banovat”uživatele, což jim zamezí v dalším používání systému, dokud nevyřeší důvod, proč toto zablokování mají.

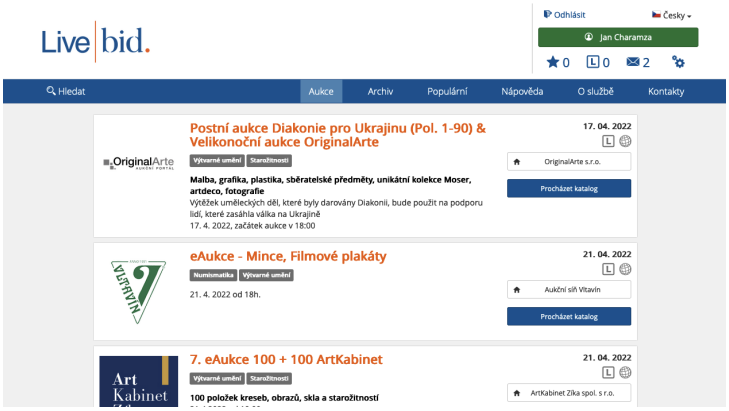
2.2 Aktuální systém

Sekce aktuální systém popisuje systém takj, jak již nyní funguje. Projekt Livebid funguje již od roku 2013 a od té doby se už core business logika pro provozování online aukcí vcelku ustálila. Nové požadavky jsou spíše pro pohodlnější používání systému.

2.2.1 Grafická podoba

Grafická podoba Livebidu vychází z Bootstrap knihovny, která má základní prvky, jako například tlačítka, navigační bar, layout, atd. již implementované a programátor už pouze řeší, jak to bude poskládané z hlediska struktury HTML kódu. Tato grafická podoba je podle klienta již považována za zastaralou, ale bude stále součástí přepisu, aby si uživatelé nemuseli zvykat na příliš nových věcí najednou.

Nový design se na web zanese až v budoucnu s příchodem nové verze celého produktu. Příprava frontendu by tedy měla počítat s touto budoucí změnou a podle toho přizpůsobit kód.



Obrázek 2.8 Bootstrap podoba úvodní stránky Livebidu

2.2.2 Produkční prostředí

Produkční prostředí stávající verze je hostované na Forpsi VPS hostingu. Je rozděleno na 3 prostředí: produkce (livebid.cz), předprodukce (prepro.livebid.cz) a vývojové prostředí (dev.livebid.cz). Přístup k předprodukčnímu a vývojovému prostředí je navíc chráněn jednoduchým vyskakovacím přihlašovacím oknem. Pro každé prostředí je spuštěna webová aplikace (backend a frontend jsou svázány spolu), backend pro živou aukci, MySQL a Redis databáze (databáze pouze pro prod a dev, prepro využívá dev).

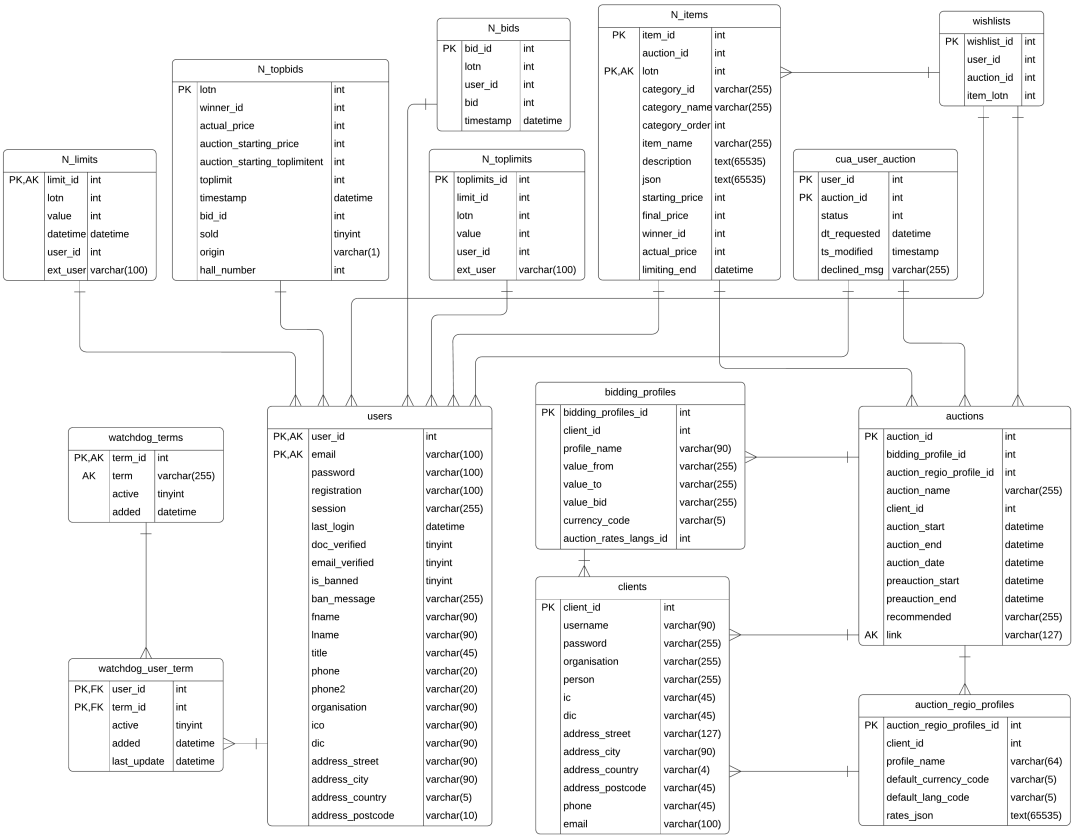
Celý backend funguje v rámci PM2, správci procesů NodeJS aplikací pro produkční prostředí. Webová aplikace je bezestavová, takže ji je možné horizontálně škálovat. V rámci PM2 produkčního prostředí je spuštěna 3 krát, takže i při pádu jedné z instancí web stále funguje v pořádku. Nahrávání nové verze do produkce probíhá díky postupnému nasazování v PM2 bez výpadků služby. Obrázky jsou ukládány v rámci úložiště VPS instance.

2.2.3 Databáze

Aktuální systém využívá relační databázi MySQL. Aukce jsou všechny v jedné tabulce, ale položky, limity a příhozy mají každá vlastní tabulku pro každou aukci, proto jsou v diagramu označeny s předponou $N\_$. Jedním z požadavků pro přepis aplikace je i úprava a vyčistění aktuální

databázové struktury, počínaje například sjednocením zmíněných tabulek do jedné tabulky pro všechny aukce, aby bylo možné mezi tabulkami efektivně vyhledávat v rámci SQL dotazů.

Migrace databáze probíhají manuální úpravou v rámci každého prostředí, nejsou dostupná ani žádná seedovací data pro okamžité spuštění vývojového prostředí. V administraci webu je možné přehrát aktuální produkční data do vývojového / předprodukčního prostředí.



Obrázek 2.9 Databázový ER model aktuální databáze

2.3 Modernizované řešení

Sekce modernizované řešení popisuje, co musí aplikace splňovat po doručení nové verze softwaru. Většina funkcionalit bude zachována stejná (včetně Bootstrapového vzhledu webu) a další velké změny se plánují až po dokončení tohoto přepsání aplikace, každopádně některé věci si přeje zákazník upravit rovnou.

Zákazník si přeje upravit proces ověřování uživatele, který si nyní pro každou aukci řeší sám aukční dům podle uživatelem vyplněných údajů. Schvalování nyní bude moci stále probíhat touto cestou, ale preferovanější cesta bude přes úplné ověření uživatele pomocí služby GetID[4], která provede kontrolu dokladu, fotografie uživatele kontrolující podobnost s fotografií na dokladu a ověření živosti pomocí pohybu obličeje. Aukční dům si poté nastaví politiku přijímání uživatelů do aukce podle jejich ověření.

Navíc si zákazník přeje upravit průběh registrace, kdy místo jedné dlouhé stránky bude registrace rozdělena na více stránek, které bude uživatel muset všechny projít pro úspěšné dokončení registrace. Registrace má navíc obsahovat kontrolu síly hesla a jestli není podle služby

HaveIBeenPwned hlášeno jako kompromitované. Vyplňování ulice má být doplněno o našeptávač pro vyplnění zbylých údajů o adrese.

2.3.1 Funkční požadavky

Funkční požadavky definují funkce, které musí produkt při předávání splňovat.[5] Popisují, co si zákazník představuje od výsledného produktu. Požadavky obecně je vhodné mít označeny krátkým identifikátorem, stručným názvem a popisem. Vhodné vlastnosti pro funkční požadavky jsou:

- Jednoznačný - Není možné si pod tímto požadavkem představit více možných věcí
- Nekolizní - Požadavky se navzájem logicky nevylučují
- Testovatelný - Je možné tento požadavek otestovat a říct ano, nebo ne, jestli je dodržený

2.3.1.1 Funkční požadavky pro novou aplikaci

FR1 *Přihlášení a registrace*
Návštěvník webu se může registrovat. Registrace bude probíhat přes více krokovou stránku, která bude požadovat email, heslo, jméno, telefonní číslo a adresu uživatele, včetně jeho zájmů, případně i IČO, DIČ a název firmy. Adresa je našeptávána podle vyplněné ulice a čísla popisného. Vyplněné IČO doplní DIČ a název firmy. Registrovaný uživatel se může přihlásit pomocí emailu a hesla. Při prvním přihlášení si musí nejprve ověřit svůj email. Email ověří kliknutím na ověřovací odkaz v příchozím emailu, který obdrží po registraci.

FR2 *Přihazování a limitování*
Uživatel schválený do aukce může v živé aukci vytvářet příhozy na aktuálně draženou položku. Uživatel nemůže navýšit svůj vlastní příhoz. Pokud uživatel přidal nejvyšší příhoz a skončila dražba položky, uživatel položku vyhrál. Může také v probíhající limitaci aukce přidat limit na jakoukoliv položku v aukci. Při dražbě položky s limitem v živé aukci systém přihazuje automaticky za uživatele do maximální výše limitu. Tento limit uživatel nemůže snížit ani smazat, pouze zvýšit. Maximální částka nejvyššího limitu je pro ostatní uživatele neznámá. V případě sálové aukce doplňují aukční operátoři do systému příhozy ze sálu a mají možnost přidat prioritu příhozu ze sálu vůči tomu z internetu. Sálové příhozy mohou odebrat, či mohou přímo všechny příhozy na aukční položku resetovat a začít aukci této položky znovu.

FR3 *Ověření uživatele*
Pro zažádání účasti v aukci si musí uživatel ověřit emailovou adresu a svou identitu. Identitu si musí ověřit pomocí fotografie dokladu, fotografie obličeje a pohybu obličeje v rámci externí služby GetID. Ověřený uživatel se může přihlásit do aukce (proběhne okamžitě), což je předpoklad pro vkládání limitů a přihazování během dražby.

FR4 *Prohlížet aukce*
Prohlížet výpis aukcí, detail aukce, aukční řád a informace může kdokoliv. Výpis aukcí je možné filtrovat a je rozdělený na nadcházející aukce a archivní aukce. Detail aukce zobrazuje všechny potřebné informace související s aukcí, jako název, popis, čas konce limitování a začátku aukce, podmínky účasti, pořadatele, cenové kroky a doporučené položky. Je možné zobrazit informace o aukčním domu s mapou a seznamem jeho jiných aukcí.

FR5 *Prohlížet aukční položky*
Prohlížet detail aukční položky a jejich výpis v detailu aukce, ať už všech, nebo rozdělené podle kategorie, může kdokoliv. Položky je možné procházet i mimo aukce seřazené podle parametru oblíbenosti s filtrováním, nebo jako výsledky hlídacího psa, které jsou vyhledané dle klíčových slov vybraných uživatelem. Ve výpisu položky je možné přidat položku do oblíbených a vidět, jestli jí uživatel vyhrává. V detailu aukční položky se nachází název, číslo položky, popis, obrázky, aktuální cena, počáteční cena, kolik má limitů a kolik uživatelů ji má oblíbenou, cenovou tabulku a v jaké aukci se nachází. Přihlášený uživatel se může zúčastnit aukce a vidět ostatní limity. Uživatel ověřený do aukce může na položku vkládat limity.

FR6 *Informace o uživateli*
Uživatel si bude moci zobrazit svoje osobní a kontaktní údaje. Kromě jména a příjmení si je bude moci změnit. Uživatel uvidí přehled všech jeho základních informací včetně stavu ověření, statistik, nadcházejících aukcí a předmětů na jednom místě. Uživatel také uvidí jeho statistiky ze zúčastněných aukcí a vyhraných položek. Ty budou rozdělené na textovou část, grafické znázornění podle měsíců a rozdělení podle aukčního domu.

FR7 *Vyhledávání*
Kdokoliv může vyhledávat mezi aukčními položkami, aukcemi a aukčními domy. Vyhledávání nejprve nalezne výsledky pro všechny zmíněné kategorie najednou a uživatel může přejít do konkrétního vyhledávání podle kategorie, které mají navíc možnost filtrovat mezi nalezenými položkami. Vyhledávání obsahuje našeptávač populárních klíčových slov.

FR8 *Položky relevantní pro uživatele*
Uživatel si bude moci zobrazit položky, které vyhrál, vložil na ně limit, nebo si je uložil do oblíbených. Pro každou tuto kategorii bude samostatná stránka. Položky budou zobrazené v kontextu aukcí. Bude se zobrazovat jejich aktuální cena a součet pro celou aukci. U limitů se bude navíc zobrazovat uživatelův nejvyšší limit, u oblíbených počáteční cena. Bude si je moci filtrovat.

FR9 *Živá aukce*
Živá aukce nabývá jedné z fází: nenačtená aukce, načtená aukce, připravená aukce, probíhající dražba, pozastavená aukce a hotová aukce. Během živé aukce je aktivní jedna položka, na kterou lze během probíhající dražby vkládat příhozy. Při probíhající dražbě je zde časovač odpočítávající konec do odklepnutí aukční položky. Po uplynutí odpočtu se pak přechází na další položku. S každým příhozem se časovač resetuje a začíná od znovu. Aktuální fázi živé aukce a interval časovače příhozů je nastavitelný aukčním operátorem. Pokud je živá aukce v jiném stavu než nenačtená aukce, aukce je viditelná pro kohokoliv na veřejné části Livebidu.

2.3.2 Nefunkční požadavky

Nefunkční požadavky popisují požadované kvality systému.[6] Nefunkční požadavky mají splňovat stejné vlastnosti pro jednoznačnost jako funkční požadavky. Nefunkční požadavky mohou popisovat škálovatelnost, dostupnost, zabezpečenost, použitelnost, rychlost služby a mnoho dalších hodnot. Systém může bez nefunkčních požadavků fungovat, každopádně nemusí splňovat očekávané kvality od uživatele a jeho správců.

2.3.2.1 Nefunkční požadavky pro novou aplikaci

NR1 *Responzivita*
Webová aplikace musí být dostupná a použitelná i pro mobilní uživatele se zařízením až po rozlišení 320x480, není však vyžadována samostatná mobilní aplikace.

NR2 *Podpora posledních verzí prohlížečů*
Webová aplikace bude podporována pro 6 posledních verzí prohlížečů Firefoxu a Chrome.

NR3 *Horizontální škálovatelnost systému*
Systém bude možné škálovat přidáním dalších běžících instancí za load balancer.

NR4 *Plně přeložený web*
Celý web musí být plně přeložený, aby šel procházet jak v češtině, tak v angličtině.

NR5 *Svižnost webu*
Aplikace se při prvním načtení / obnovení vykresluje jako SSR (HTML a CSS konkrétní stránky je předpřipraven na serveru, ale při procházení stránek využívá CSR (stránku sestavila čistě strana prohlížeče, dotazy na server probíhají až v rámci této stránky s indikátory načítání), což dodá pocit svižnější aplikace.

2.3.3 Případy užití

Případy užití (často také anglicky use case) jsou detailnější způsob zápisu (nebo rozpadu) funkčních požadavků podle jejich aktérů, požadavků a scénářů. Scénáře představují chronologickou sekvenci interakcí mezi uživateli a systémem.

2.3.3.1 Seznam případů užití

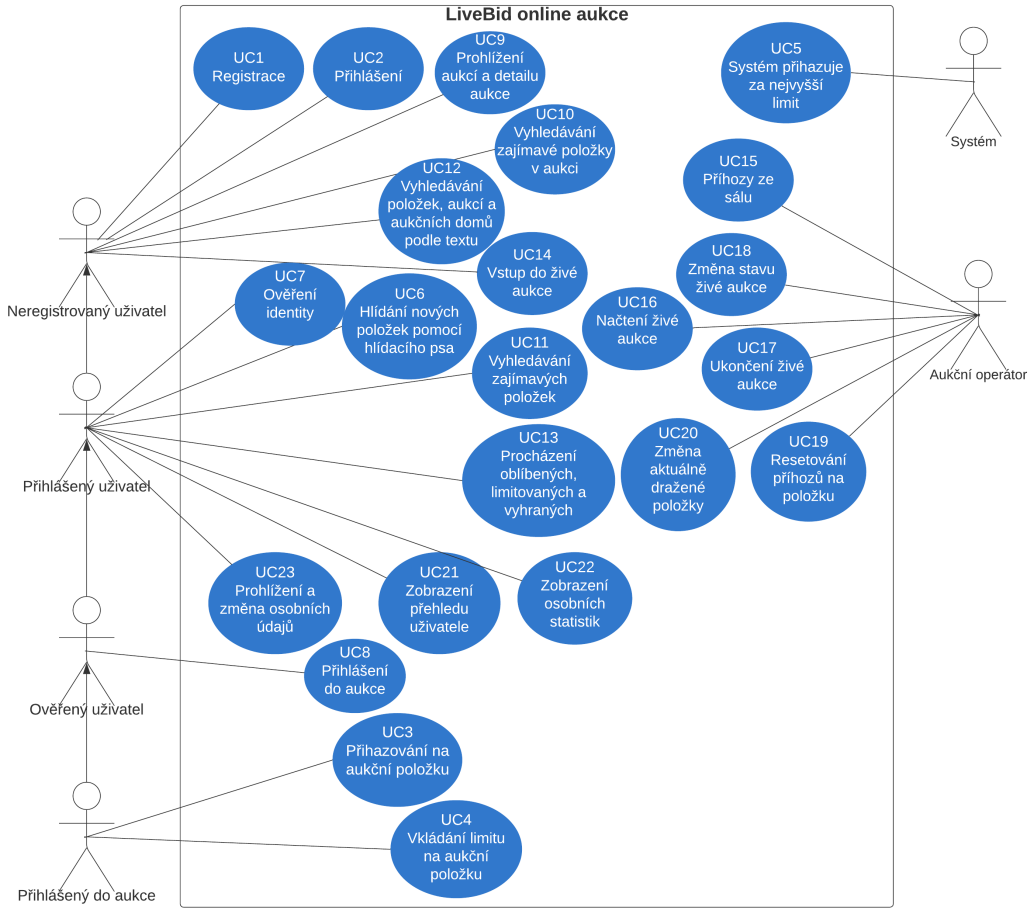
Vzhledem k velkému počtu případů užití zde bude detailně rozepsáno jen pár případů, ale bude zde uveden seznam všech případů užití včetně diagramu případů užití.

Veřejný web
UC1 *Registrace*
UC2 *Přihlašování*
UC9 *Prohlížení aukcí a detailu aukce*
UC10 *Vyhledávání zajímavé položky v aukci*
UC12 *Vyhledávání položek, aukcí a aukčních domů podle textu*
UC14 *Vstup do živé aukce*

Přihlášený uživatel
UC4 *Vkládání limitu na aukční položku*
UC6 *Hlídání nových položek pomocí hlídacího psa*
UC7 *Ověření identity*
UC8 *Přihlášení do aukce*
UC11 *Vyhledávání zajímavých položek podle jejich popularity*
UC13 *Procházení oblíbených, limitovaných a vyhraných položek*
UC21 *Zobrazení přehledu uživatele*
UC22 *Zobrazení osobních statistik*
UC23 *Prohlížení a změna osobních údajů*

Živá aukce
UC3 *Přihazování na aukční položku*
UC5 *Systém přihazuje za nejvyšší limit*
UC15 *Příhozy ze sálu*
UC16 *Načtení živé aukce*
UC17 *Ukončení živé aukce*
UC18 *Změna stavu živé aukce*

UC19 *Resetování příhozů na položku*
UC20 *Změna aktuálně dražené položky*



Obrázek 2.10 Model případů užití

2.3.3.2 Detaily případů užití

UC1 *Registrace*
Umožňuje registrovat se novým uživatelům. Registrace bude probíhat přes více krokovou stránku, kde bude k vyplnění: 1. email a heslo, 2. jméno, telefonní číslo, doručovací adresa a volitelně fakturační adresa, IČO, DIČ a název firmy, 3. oblasti zájmů (alespoň jeden musí uživatel vybrat) a poslední 4. potvrzení smluvních podmínek. Adresy jsou našeptávány podle vyplněné ulice a čísla popisného. Vyplněné IČO doplní DIČ a název firmy. Vyplňování hesla zobrazí jeho sílu na 3 úrovně (slabé, střední, silné). Uživatel v každém kroku vidí, ve kterém se zrovna nachází a jaké mu ještě zbývají do konce. Může se libovolně vrátit zpátky na předešlé kroky. Registrace využívá Google Recaptcha v3 řešení proti spamu.

Předpoklady:

- Není přihlášen
- Není již registrován se svou emailovou adresou

Scénář:

1. Uživatel si chce vytvořit nový účet.
2. Systém zobrazí formulář s povinnými poli pro emailovou adresu, heslo a heslo znovu.
3. Uživatel zadá neregistrovaný email a vyplní stejné heslo do obou polí pro hesla.
4. Systém zobrazí uživateli, jaké má jeho heslo úroveň síly (slabé, střední, silné) a zkontroluje, zda se shodují.
5. Uživatel přejde na další stránku
6. Systém zobrazí formulář s povinnými poli pro jméno, příjmení, telefonní číslo a doručovací adresu obsahující stát, ulici, číslo popisné, město a PSČ, včetně volitelné fakturační adresy (která navíc obsahuje i další název příjemce), IČO, DIČ a názvu firmy.
7. Uživatel z České republiky vyplní jméno, příjmení, telefonní číslo, ulici a číslo popisné.
8. Při vyplňování ulice a čísla popisného se objeví našeptávač s celou adresou.
9. Uživatel potvrdí výběr našeptávače.
10. Našeptávač doplní celou doručovací adresu.
11. Uživatel pokračuje na další stránku.
12. Systém zobrazí výběr ze zájmů, kde alespoň jeden zájem je povinný.
13. Uživatel vybere zájmy, o které má zájem a jde na další stránku.
14. Systém zobrazí všeobecné smluvní podmínky, které si musí uživatel celé pročíst až do konce.
15. Uživatel si je přečte do konce.
16. Systém zobrazí povinné odsouhlasení všeobecných obchodních podmínek a také možnost vytvořit si účet.
17. Uživatel odsouhlasí obchodní podmínky a pokračuje dále.
18. Systém přesměroval uživatele na poslední stránku registrace, kde mu gratuluje.

UC4 *Vkládání limitu na aukční položku*
Uživatel přihlášený do aukce může vkládat limity na jakékoliv položky v rámci aukce. Již vložený limit nemůže snižovat ani mazat, pouze navyšovat. Uživatel může vkládat libovolně vysoký limit, jeho hodnota musí pouze respektovat tabulku příhozů.

Předpoklady:

- Uživatel je přihlášený do aukce
- Začalo limitování položek v aukci
- Neskončilo limitování položek v aukci
- Aukce je zveřejněná

Scénář:

1. Uživatel si vybere aukční položku, o kterou má zájem.
2. Na stránce se objeví okno s detailem položky, kde se nachází část pro vložení nového limitu obsahující aktuální cenu, začáteční cenu, cenu s dalším krokem podle tabulky příhozů a input pro vložení ceny limitu.
3. Uživatel klikne na input pro vložení ceny.
4. Pod inputem pro vložení ceny se objeví našeptávač s dalšími kroky ceny podle tabulky příhozů, v kterém může uživatel procházet pomocí šipek nahoru, dolu a klávesou enter odsouhlasit výběr.
5. Uživatel vybere cenu pomocí našeptávače cen, nebo zadá vlastní výši limitu a klikne vložit limit (nebo klávesu enter).
6. Objeví se okno s odsouhlasením vložení limitu.
7. Uživatel v tomto okně klikne na tlačítko "Vložit limit".
8. Objeví se potvrzení o vložení limitu.
9. Uživatel vidí svůj limit v seznamu limitů označený jako "můj limit"a narozdíl od ostatních uživatelů zde vidí skutečnou výši svého limitu.

UC10 *Vyhledávání zajímavé položky v aukci*
Návštěvník webu může procházet všechny (nebo podle kategorie) položky v aukci. Pokud položka uživatele zaujme, může si ji jako přihlášený uživatel přidat do oblíbených položek, nebo si ji otevřít a prohlédnout její detail v popup okně. V detailu aukční položky se nachází název, číslo položky, popis, obrázky, aktuální cena, počáteční cena, kolik má limitů a kolik uživatelů ji má oblíbenou, cenovou tabulku a v jaké aukci se nachází. Přihlášený uživatel uvidí historii limitů položky. Pro přechod na detail další položky uživatel nemusí vypínat popup okno, ale může přejít rovnou na detail další / předchozí položky.

Předpoklady:

- Aukce je zveřejněná

Scénář:

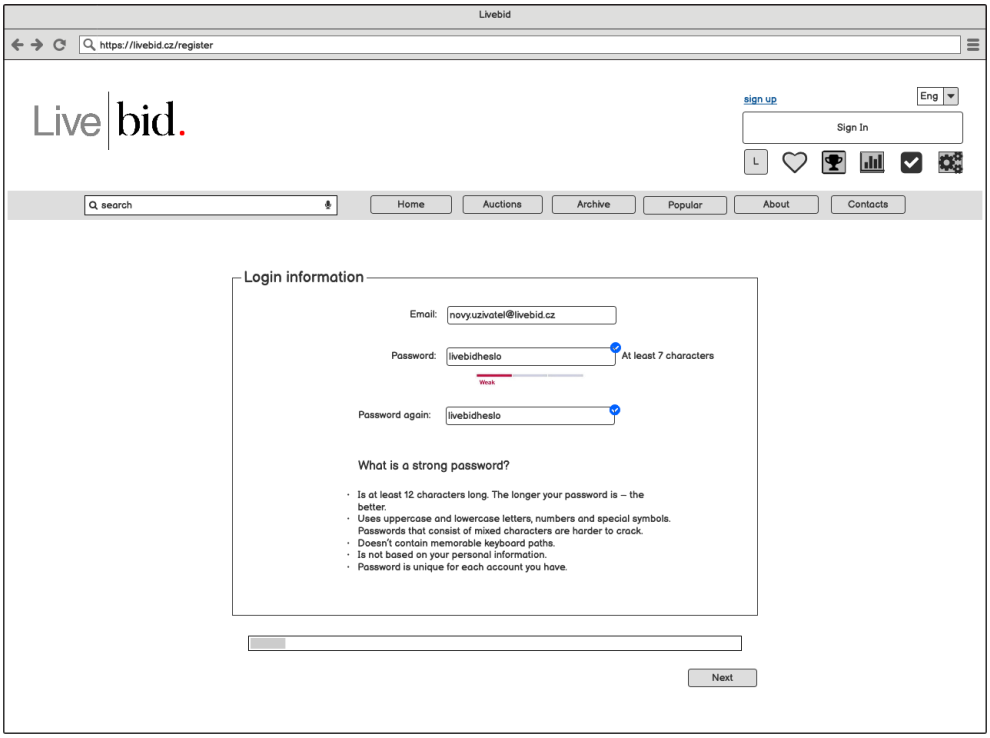
1. Uživatel je v detailu konkrétní aukce.
2. Systém zobrazí menu s možností zobrazit všechny položky, nebo jen pro konkrétní kategorii.
3. Uživatel si vybere zobrazení aukčních položek.
4. Systém zobrazuje výpis aukčních položek. Pro přihlášené uživatele zobrazuje možnost přidat si položky do oblíbených. Uživatelovi oblíbené položky mají navíc modrý rámeček. Položky, co vyhrávají, mají zelený rámeček a prohrávající červený.
5. Uživatel si prochází aukční položky. Otevře si položku, která ho zaujme.
6. Systém zobrazí detail aukční položky v modal okně s možností přejít na další, nebo předchozí aukční položku v aukci. Zde si může přihlášený uživatel znovu přidat položku do oblíbených a vidí zde stejné barevné orámování jako u výpisu položky.
7. Pokud položka uživatele zaujme, přidá si ji do oblíbených, nebo na ni může vložit limit, jinak přeskočí na krok 9.

8. Systém upraví barevné ohraničení položky.
9. Uživatel přejde na další / předchozí položku.
10. Systém přejde na krok 6 s další / předchozí položkou v aukci.
11. Pokud si už uživatel nechce prohlížet detaily položek, zavře modální okno.

2.4 Wireframy

Pro konkrétnější představu používání určitých funkcí byly provedeny návrhy wireframů. Wireframy jsou prototypové návrhy stránek znázorňující prvky, jejich přibližné rozložení na stránce a procházení mezi nimi. U wireframů je zásadní nesoustředit se na design uživatelského rozhraní, protože pak návrh trvá déle a pro navrhujícího je takový návrh těžší zahodit ve prospěch lépe rozmístěných návrhů.

Wireframy byly vytvořeny v rámci aplikace Balsamiq, která se přesně zaměřuje pouze na funkci komponent a nikoliv jejich vzhled. Wireframy korespondují s případy užití a jsou vývojáři předávány přímo u nich.



Obrázek 2.11 Ukázka wireframu registrace pro první krok

Kapitola 3
Použité technologie

Kapitola použité technologie se zabývá konkrétním výběrem technologií a jejich stručným popisem.

3.1 Webové rozhraní

Webové rozhraní je pro uživatele hlavní způsob, jak interagovat se systémem. Hlavním účelem je nabídnout uživateli jednoduchou možnost, jak se dostat od registrace, přes přihlášení do aukce, až po přidávání limitů a nakonec účast v živé aukci a přidávání příhozů v reálném čase. Ideální je jednoduše pochopitelné rozhraní, ve kterém nebudou uživatelé ztraceni a dokážou se dostat ke koupi položek, které je zajímají.

Kromě toho by bylo vhodné, aby bylo webové rozhraní příjemně plynulé pro procházení. K tomu je vhodné volit přístup tzv. single page application. Stránka se v tomto režimu stáhne pouze jednou a průchod na další stránky už je pak bez stažení nové stránky. Při správně voleném přednačítání stránek, cachování backend požadavků a code splittingu javascriptového kódu stránek bude procházení webem rychlé a přirozené.

Cachování je proces, kdy se málo měnící se výsledky uloží do mezipaměti, z které se po nějakou dobu načítají bez skutečného počítání výsledků. Technika code splittingu rozdělí sestavený kód do několika balíčků. Prohlížeč si tak stáhne pouze ten kód, který je zrovna relevantní. Další části aplikace si automaticky načtou potřebný kód pro další fungování.

3.1.1 React

React je JavaScriptová knihovna původně vyvíjená společností Facebook pro tvorbu uživatelského rozhraní [7]. Hlavní výhodou Reactu je deklarativní tvorba uživatelského rozhraní, kde je podoba komponenty definována při určité konfiguraci vnitřních stavů. Při změně nějakého ze stavů je komponenta znovu překreslena a všechny komponenty pod ní.

Takovéto změny by samy o sobě s naivní implementací byly velmi náročné, protože smazání prvku z DOMu[1] a jeho opětovné poskládání znovu s každou změnou stavu by působilo pro uživatele "sekaně". Tvůrci Reactu pro tento případ vymysleli takzvaný Virtuální DOM, který si vytvoří svůj vlastní virtuální model DOMu a poté při změně stavu "skutečný" DOM přeorganizovává podle změn od poslední verze.

React je často označován jako framework, podobně jako Angular. Narozdíl od Angularu však neposkytuje žádné funkce navíc, než jen jaké jsou potřeba pro zobrazování uživatelského rozhraní.

[1] DOM (Document Object Model) je programátorské rozhraní pro manipulaci s obsahem prvků na stránce. `https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction`

React neobsahuje funkce pro routování napříč stránkami, neobsahuje nástroje pro vytvoření nasaditelného buildu, atd. To dává vývojářům na jednu stranu volnost, na druhou stranu si musí sami vybrat správnou knihovnu pro řešení těchto problémů.

Výpis kódu 3.1 Ukázka funkcionální komponenty s hooky v Reactu

```
const SomeComponent = () => {
        const [val, setVal] = useState(0)

        useEffect(() => {
                const interval = setInterval(() => {
                        setVal(val => val + 1)
                }, 1000)
                return () => clearInterval(interval)
        }, [])

        return (
                <div>Counter is: {val}</div>
        )
}
```

3.1.2 NextJS

NextJS je webový framework postavený na knihovně React.[8] Rozšiřuje React o další hodnotné funkce. Z nich, pro tento projekt, jsou zajímavé funkce pro správu routování mezi stránkami (včetně optimalizovaných načítání stránek), server side rendering, optimalizace obrázků.

NextJS dovoluje využít i kapacit backendu NodeJS prostředí. Mohou se v něm vytvořit API endpointy, psát funkce optimalizované pro edge servery, nebo již zmiňované server side rendering stránky (zkráceně SSR). SSR při prvním načtení stránku sestaví na straně serveru a uživateli ji pošle již sestavenou, což se v rámci běžných React projektů děje až v prohlížeči.

NextJS je vyvíjený společností Vercel, která poskytuje jednoduchý cloudový hosting, na který lze projekty vytvořené (nejenom) v NextJS nahrát pomocí jednoho příkazového řádku. Pro tento projekt byl Vercel hosting používaný pro nahrání prototypovacích verzí, kde byl backend nahrazen mockovaným prostředím. Další výhoda Vercelu je, že poskytují verzování nasazení, takže je možné se kdykoliv podívat na starší verze návrhu frontendu.

3.1.3 TypeScript

TypeScript je programovací jazyk vyvíjený společností Microsoft, postavený na JavaScriptu se silnou podporou datových typů [9]. Slouží jako vhodná náhrada JavaScriptu, doporučovaná alespoň pro střední a velké projekty.

Největší přidaná hodnota TypeScriptu je tedy zmíněná podpora staticky kontrolovaných datových typů. Proměnným a návratovým hodnotám funkcí jsou přiřazeny datové typy. TypeScript pak při jejich přiřazování kontroluje, jestli někde programátor neudělal chybu. Správnost těchto přiřazení se kontroluje pouze staticky při transpilaci, při běhu aplikace (runtime) mohou přiřazení špatných datových typů nastat. To se může stát například při získávání hodnot z API, kde pro zajištění správnosti datových typů musí být provedena kontrola za běhu aplikace, což musí provést programátor samostatně. Dále také TypeScript přidává podporu rozhraní (interfaces) a dekorátory.

Datové typy není potřeba přiřadit vždy. V některých případech je TypeScript schopný si je sám přiřadit, například při inicializaci hodnoty v momentu její deklarace. Tomuto způsobu přiřazení se říká “Type inference”. Pokud proměnné přiřadíme hodnotu ‘false‘, TypeScript počítá s tím, že tato proměnná má datový typ boolean. Kdybychom jí chtěli přiřadit číselnou hodnotu,

TypeScript vrátí chybovou hlášku. Pokud bychom skutečně chtěli, aby tato proměnná uměla datový typ boolean a číselnou hodnotu, musel by jí být explicitně přiřazen datový typ ‘boolean — number‘.

TypeScript samotný se na webu nespouští, kód se musí nejprve transpilovat pomocí příkazu ‘tsc‘. Naopak je TypeScript bez transpilace vítán v runtime prostředí Deno (nový projekt původního tvůrce NodeJS). [10]

3.1.4 State management

State management se zabývá správou a přehledným předáváním stavů napříč aplikací. Jako stav je zamýšleno například přihlášení uživatele, aktuální hodnoty filtrování, průběh živé aukce, atd. Správně řešená správa stavů je klíčová pro rozrůstající se aplikaci. Některé stavy stačí řešit lokálně v rámci komponenty, na což často vystačí základní React hooky pro lokální uložení stavů. Pro stavy využívané ve více komponentách již příliš vhodné nejsou, protože se musí předávat z komponenty na komponentu v rámci argumentů, a to se při větším počtu takovýchto stavů může stát velmi jednoduše nepřehledné.

Tento problém lze v Reactu vyřešit použitím kontextů. Obalením komponenty s kontextem k němu lze ve všech potomcích přistupovat. Ten je možné upravovat pomocí definovaných akcí a úpravu stavu tak víc ochránit před nechtěnými změnami.

3.1.4.1 MobX

MobX verze 6 je knihovna pro pohodnější správu stavů (nejen) v rámci Reactové aplikace. Pro vytvoření stavů v MobX stačí vytvořit třídu, které se v konstruktoru zavolá funkce makeAutoObservable(this) a z vlastností této třídy se stanou stavy, z setter metod se stanou akce, kde stavy se přenastavují jednoduchým přiřazením nové hodnoty do vlastností instance třídy a z getter metod derivace (vypočtené hodnoty) stavů. Z této třídy se poté vytvoří instance, která se buď předává jako globální proměnná, nebo se může předávat s využitím již zmíněných React kontextů. Komponenty využívající těchto stavů je potřeba zaobalit do funkce observer. Komponenty se poté po zavolání akce stavového úložiště sestaví znovu.

Výpis kódu 3.2 Příklad MobX store třídy

```
class SomeStore {
        private someValue: number = 0

        constructor() {
                makeAutoObservable(this)
        }

        getSomeValue(): number {
                return this.someValue * 2
        }

        setSomeValue(val: number): void {
                this.someValue = val
        }
}
```

3.1.5 Stylování komponent

Knihovna Stitches přidává podporu definice CSS stylů v React části kódu.[11]. Podporuje server side rendering pro styly a poskytuje typově bezpečné definice CSS pravidel. Další velkou výhodou

Stitches je integrovaná podpora pro varianty stylů, například *Button* může mít variantu rounded nebo variantu size s možnostmi small, medium a big.

3.1.6 Mockovaná data

Komunikace s backendem bude probíhat formou přístupu přes Rest API, které je zadefinované v API agreementu. API agreement, neboli dohoda o fungování rozhraní mezi frontendem a backendem, je soubor ve formátu OpenAPI v2 definující podobu komunikace pro každý endpoint na backendu. Kromě toho budou živé aukce probíhat přes obousměrné websocketové spojení. V první fázi vývoje frontendu ale bude stačit pouze přístup k backendu mockovat, čímž se nejprve definuje použité rozhraní bez řešení konkrétní implementace, která se začne řešit až při konečném tvaru rozhraní. Při práci jednoho člověka na frontendu a backendu tak není potřeba často měnit kontext programovacího prostředí, při rozdělené práci v týmu tak není potřeba při tvorbě frontendu čekat na dokončení implementace backendu.

Knihovna MSW (Mock Service Worker) poskytuje možnost jednoduše definovat mocky HTTP endpointů, které jsou spuštěné přímo v prohlížeči.[1] MSW zachycuje požadavky na síťové úrovni a nahrazuje je mockovanými zprávami. Zápis mockovaných handlerů je inspirovaný knihovnou Express. Mocky psané v knihovně MSW lze použít i pro testování správné funcionality kódu v unit testech.

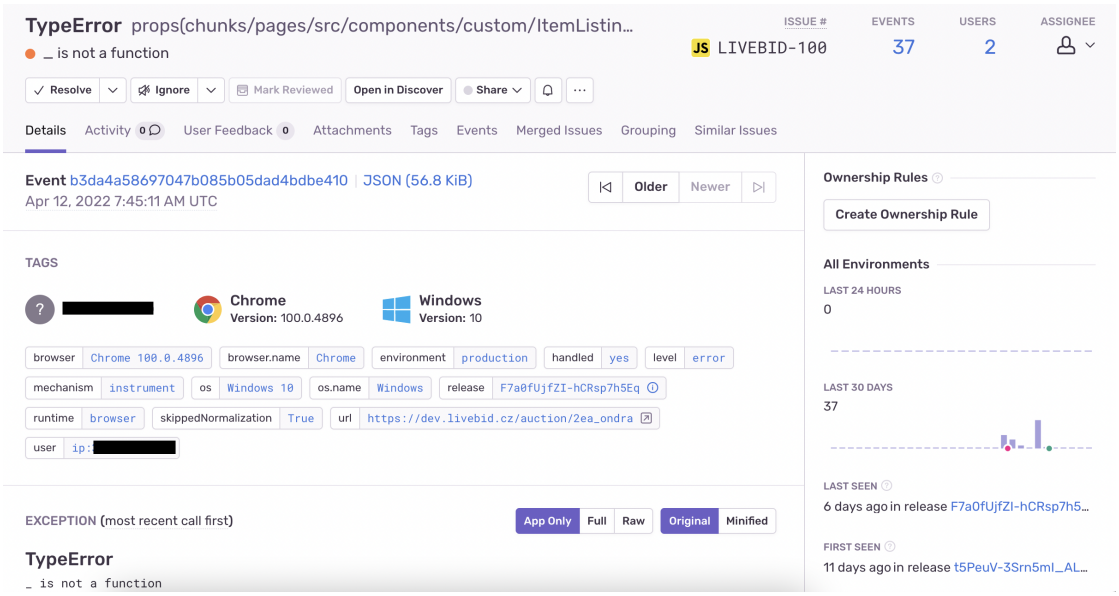
Výpis kódu 3.3 Příklad mocku pro endpoint books [1]

```
rest.get('/books', (req, res, ctx) => {
  // So the issue is when there's a single book!
  return res(
    ctx.json([
      {
        id: 'ea42ffcb-e729-4dd5-bfac-7a5b645cb1da',
        title: 'The Lord of the Rings',
        publishedAt: -486867600
      }
    ])
  )
})
```

3.1.7 Sentry.io

Ve vývojovém prostředí lze vcelku pohodlně dopátrat příčina chyby, protože jsou dostupné konzolové výstupy, aplikace poskytuje debug výpisy chyb a víme, co jsme před výskytem chyby provedli za operace. V případě nasazení frontendu na produkční prostředí tyto informace vůbec být dostupné nemusí, protože je přístup ke konzolovým výstupům uživatele bez jejich poskytnutí nemožný. Aplikace na produkci má speciální optimalizovaný build bez debugovacích nástrojů a není jisté, jakými kroky se uživatel dostal k chybě.

Pro sledování těchto problémů je nasazena služba Sentry.io, která sleduje výskyt chyb v prohlížeči na straně uživatele a pokud se nějaká chyba objeví, detailně ji reportuje na web služby sentry.io.[12] Zde lze vidět informace o nastalé chybě, na jaké stránce se objevila, z jakého prohlížeče uživatel přistupoval. Navíc chyby seskupuje dohromady a sleduje, jak často se objevují, v jakých verzích se objevují a na jakých zařízeních se procentuálně objevily.



Obrázek 3.1 Ukázka zaznamenané chyby na Sentry.io

3.1.8 Údržba kódu

Udržovaný kód je důležitý pro snažší budoucí rozšiřování kódu. Je vhodné mít veškerý kód v projektu konzistentní a podle předem určených pravidel, kterých se budou všichni vývojáři v týmu držet. Vhodné to je však i pro tým jednoho člověka. Kód by měl psát každý tak, aby mu rozuměl i po měsíci, co na projektu nebude pracovat.

Pro dodržování nastavených pravidel jsou vybrané nástroje, které je umí i částečně opravit. Ty jsou spuštěny na začátku procesu tvorby commitu. Pokud objeví chybu, tvorba commitu neprojde.

3.1.8.1 Prettier

Prettier je formátovací nástroj. V této sestavě neslouží ke kontrole kódu, ale při každém uložení celý soubor přeformátuje do jednotného formátu. Formátuje odsazení, odstraňuje nepotřebné středníky, příliš dlouhé řádky rozdělí na více řádků, atd. Lze použít jak na TypeScript (JavaScript), tak i na HTML, CSS a JSON soubory.

Nastavení není příliš komplexní, protože tvůrci Prettieru se rozhodli, že bude tzv. opinionated, tedy že některé věci si nejde úplně přizpůsobit podle vlastních představ, ale podle obecně přijímaných podmínek. Pro potřeby tohoto projektu to však nevadí a konfigurační soubor má pouhých 10 řádků.

3.1.8.2 ES Lint

ES lint je lintovací nástroj určený pro JavaScript / TypeScript. Lintovací nástroje mají za úkol hlídat jednotný styl nahraného kódu, v tomto případě vždy před nahráním kódu na git. Na rozdíl od Prettieru jeho konfigurace dovoluje nastavit každý detail. Jako základ konfigurace bylo využito obecné nastavení od firmy AirBNB, které se dále upravovalo podle konkrétních představ projektu. I přes to má konfigurační soubor 67 řádků. ES Lint je schopný některé chyby sám opravit pomocí příkazu ‘eslint —fix‘.

3.2 Backend

Backendový systém se stará o businessovou logiku celé aplikace. Obstarává pro frontend veškeré informace ohledně aktuálních aukcí a aukčních položek, informace konkrétní pro přihlášeného uživatele a v neposlední řadě funkce tvořící nejvíce přidanou hodnotu, limitování a příhozy během živé aukce. Backend počítá s neočekávanými a špatnými vstupy, které blokuje s vhodnou chybouvou hláškou. V případě živé aukce si dává pozor, aby byly požadavky vykonávány za sebou a nedocházelo ke ztrátě integrity dat.

Kromě toho bude úkolem backendového systému umět komunikovat přes RestAPI a websockety, všechny informace persistentně zaznamenat do databázového systému, logovat všechny aktivity komunikované s backendem do záznamového logu (včetně chybových hlášek na straně serveru) a být konfigurovatelný.

3.2.1 Komunikační rozhraní

Jelikož je aplikace rozdělena na frontend a backend, je potřeba mezi nimi definovat komunikační rozhraní, přes které si mohou mezi sebou vyměňovat informace.

3.2.1.1 Jednosměrná komunikace

HTTP je přirozená volba pro webové aplikace, protože to je základní protokol pro načítání stránek na internetu. HTTP komunikace je jednosměrná, formou požadavku a odpovědi. Dotazy se skládají z metody, cesty url adresy, verze HTTP protokolu, hlaviček a těla přenášejícího obsah požadavku / odpovědi. HTTP komunikace je bezestavová, stavovost lze přidat manuálně přenášením ID relace s každým požadavkem.

Výpis kódu 3.4 Příklad HTTP 1.1 GET požadavku v textové formě

```
GET /Protocols/rfc2616/rfc2616-sec5.html HTTP/1.1
Host: www.w3.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:100.0) ...
Accept: text/html,application/xhtml+xml,application/xml;q=0.9, ...
Accept-Language: cs,sk;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate, br
Referer: https://www.google.com/
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
If-Modified-Since: Wed, 01 Sep 2004 13:24:52 GMT
If-None-Match: "2cc8-3e3073913b100-gzip"
TE: trailers
```

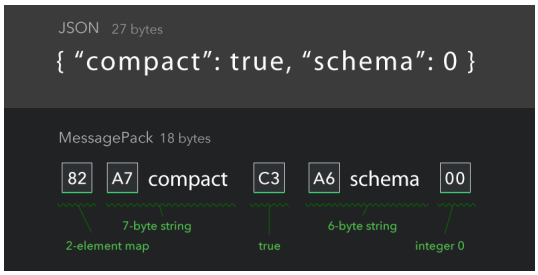
Rest API je aplikační rozhraní splňující REST architekturální vlastnosti.[13] REST nedefinuje technické detaily, ale pravidla pro architektonické rozvrhnutí endpointů. Návrh API podle REST principů musí dodržovat 6 omezení:

- Jednotné rozhraní - Každý zdroj (resource) má svůj jednoznačný endpoint.
- Oddělení klienta a serveru - Klient a server jsou odděleny rozhraním (v tomto případě síťovým HTTP rozhraním).
- Je bezestavové - Komunikace si nepamatuje stav z předchozích požadavků.
- Je cachovatelné - Data ke čtení je možné cachovat.

- Vrstvená systémová architektura - Komunikace s backendem může jít přes několik vrstev, které jsou pro klienta neviditelné. Jedna z takových vrstev může být load balancer.
- Volitelný "Code on demand"- Server může poslat klientovi kód, který může provést na své straně.

Jako formát přenášených dat byl zvolen JSON (JavaScript Object Notation), protože se s ním díky jeho rozšířenosti jednoduše pracuje v jakémkoliv jazyce (obvzláště v JavaScriptu), je lidsky dobře čitelný, ale přitom ne tolik datově náročný, jako například XML.

Existují úspornější formáty než JSON, například MessagePack, který jako vstup a výstup využívá JSON model, ale data jsou přenášena v binární podobě, čímž dokáže ušetřit okolo 25 % až 50 %, v závislosti na přenášených datech (čím méně dlouhých textů, tím lépe). Každopádně nástroje pro práci s tímto formátem nejsou tak rozšířené, jako ty pro JSON. Například v prohlížeči v konzoli se síťovou aktivitou není okamžitě vidět, jaká data jsou přenášena a kvůli kterým se objevila chyba.

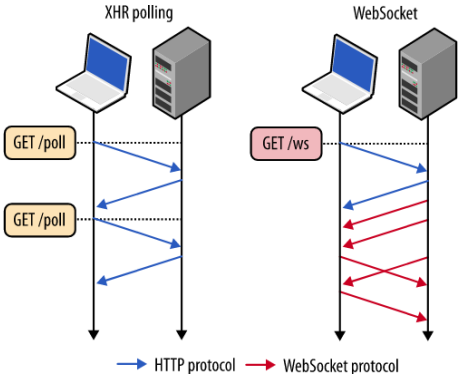


Obrázek 3.2 Úspornost formátů MessagePack vs JSON [14]

3.2.1.2 Obousměrná komunikace

Websockety představují protokol umožňující obosměrnou komunikaci v prohlížeči nad protokolem TCP.[15] Pro navázání websocketové komunikace se musí klient nejprve dotázat serveru HTTP handshake požadavkem. Pokud je požadavek v pořádku (jak protokolově, tak z hlediska ověření uživatele pomocí hlaviček), server naváže s klientem obousměrnou komunikaci.

Websockety umožňují realtime zážitek v internetovém prohlížeči, který se v minulosti musel nahrazovat buď častým dotazováním serveru, jestli se data změnila, nebo tzv. long pollingem, který navázal spojení se serverem a server odpověděl až při změně dat. Tato realtimová komunikace bude vhodná pro aukční modul, kdy se při každé změně ceny okamžitě pošle všem zúčastněným stranám aktualizace o výherci a aktuální ceně.



Obrázek 3.3 Jednosměrná HTTP komunikace vs obousměrná websocket komunikace

3.2.2 Go

Go je staticky typovaný jazyk se syntaxí jazyka C, vyznačující se bezpečnou prací s pamětí díky používání garbage collection a jednoduchou prací s vlákny s použitím jejich "goroutines". Go byl jako jazyk pro backend zvolen pro jeho předpovídatelnost (až monotónnost), se kterou by měl být kód do budoucna snáze čitelný.[16]

Jazyk Go nabízí ve standardní knihovně implementaci http komunikace, se kterou lze napsat vlastní plnohodnotný server pro komunikaci s backendem. Každopádně pro méně verbální kód a již připravená běžná řešení HTTP komunikace byl vybrán framework Fiber.[17] Framework Fiber je inspirovaný NodeJS frameworkem ExpressJS a nabízí jednoduché vytváření routování, vložení vlastního middleware kódu (či již předpřipravený middleware) a spoustu dalších funkcí pro obsloužení HTTP komunikace, jako například zpracování JSON formátu v těle požadavku a vrácení odpovědi zase ve formátu JSON.

3.2.3 MySQL

MySQL je open source relační SQL SŘBD (systém řízení báze dat) vlastněný společností Oracle.[18] Hlavní potřebou pro databázi je trvalé uložení dat ve strukturované formě. MySQL má data uchovávaná ve formě tabulek, které si na pozadí udržuje zachované v optimalizovaných strukturách, dovolující rychlé vyhledávání podle hledaných klíčů. V těchto datech je ale také vhodné vyhledávat spojitosti mezi různými entitami. MySQL dovoluje zanést relační vazby mezi jednotlivými entitami a efektivně mezi nimi vyhledávat pomocí cizích klíčů.

MySQL odpovídá na SQL dotazy. SQL dotazy jsou psané textovou formou. SQL dovoluje dotazy typu INSERT, SELECT, UPDATE, DELETE a mnoho dalších. Každý SQL dotaz musí dodržovat určitou posloupnost klíčových slov, které definují parametry požadavku. Například SELECT nesmí obsahovat ORDER před WHERE klíčovým slovem.

Výpis kódu 3.5 Příklad SQL dotazující se na konkrétní výpis aukcí

```
SELECT *
FROM auctions
LEFT JOIN clients
ON clients.id = auctions.client_id
WHERE client_id = '1'
ORDER BY created_at DESC
GROUP BY auction_id
LIMIT 0, 10;
```

Podle průzkumu vývojářů na StackOverflow za rok 2021 se jedná o nejpopulárnější databázi.[19]

3.2.4 ORM

ORM (Object Relational Mapping) je metoda převodu databázových dat na objektové struktury bez nutnosti psaní SQL kódu.[20] ORM standardizuje rozhraní pro komunikaci s databází, což také dává možnost jednoduše přesunout kód na jiný databázový systém, pokud ho zrovna zvolená ORM knihovna podporuje. Každopádně čistá ORM abstrakce pro složitější požadavky nestačí a občas je potřeba psát v rámci ORM také SQL dotazy.

3.2.4.1 GORM

Knihovna GORM dovoluje definovat databázové modely entit a vazby mezi nimi (1 - 1, 1 - N, N - M).[21] Se základní embdedded strukturou GORM modelu v sobě modely obsahují automaticky

inkrementované ID, automaticky nastavované created_at a modified_at sloupce, navíc i deleted_at sloupec pro soft delete pomocí příkazů delete (záznamy budou v databázi stále dostupné, i když budou "smazané"). Kromě MySQL podporuje také databáze PostgreSQL, SQLite, SQL Server a Clickhouse. GORM dovoluje při každém novém nasazení provést automatickou migraci databázového modelu podle modelů entit, každopádně chybí podpora pro verzované migrace s možností vrácení na starou strukturu databázového schématu.

3.2.4.2 Migrate

Pro samotnou migraci by bylo vhodnější použít velmi obecně pojmenovaný nástroj "migrate"(golang-migrate/migrate). Migrate je možné použít buď jako nástroj v příkazové řádce, nebo jako knihovnu v rámci projektu v jazyce go.[22] Migrace jsou psané jako SQL soubory identifikované pořadovým číslem ve variantě up (migrace nasazena) a down (rollback migrace). Na začátku spouštění projektu se spustí příkaz migration.Up(), ten vše nastaví a připraví pro použití knihovnou GORM.

3.2.4.3 Sqlc

Při používání GORM se objevila jedna velká nevýhoda. GORM dotazy probíhají v runtime prostředí, takže v čase kompilace nelze zjistit, jestli se v dotazu nenachází předvídatelná chyba. Tyto problémy řeší knihovna sqlc, kompilující typově bezpečný kód pro práci s databází z předdefinovaných SQL souborů.[**sqlc**] Vygenerovaný kód nemůže v SQL dotazu využívat neexistující sloupce, jinak by kompilátor skončil chybou. Také využívání vygenerovaného kódu má omezení, jaké typy lze použít v argumentech. V sqlc se databázové modely nedefinují jako objekt v go, ale jako CREATE TABLE v SQL.

3.3 Nasazení aplikace do produkčního prostředí

Nasazení softwaru, nebo-li anglicky deployment, je souhrn procesů a kroků pro zpřístupnění nové verze softwaru jeho uživatelům.[23] Před nasazením je, mimo jiné, vhodné provést unit testy, které by v případě nesrovnalosti zastavily proces nasazení. V případě tohoto projektu je potřeba nasadit systém na internet pro fungování ve webovém prostředí, ideálně bez výpadků při nahrávání nové verze.

3.3.1 Docker

Docker je technologie zajištuje konzistentní chování ve vývojovém a v produkčním prostředí. Docker zaobaluje běh aplikací do virtualizovaného prostředí, do takzvaných kontejnerů.[24] Kontejnerizace oddělí chování aplikace od systému, na kterém běží, co nejvíce, jak je to možné (definované v Dockerfile a docker-compose.yml) a běží na vlastním operačním systému, který je definovaný pro kontejner, takže při vývoji není nutné myslet na specifika produkčního serveru, protože jakýkoliv systém, který umí spustit prostředí dockeru, spustí aplikaci stejně, jako vývojový stroj.

Produkční docker engine je možné ovládat voláním docker příkazů na vzdáleném serveru, nebo pohodlněji přímo v příkazové řádce vývojového počítače, či CI pipeliny, nastavením proměnné prostředí *DOCKER_HOST* a volat příkazy, nebo nahrávat Docker obrazy pohodlně z vlastní příkazové řádky.

Výpis kódu 3.6 Připojení se ke vzdálenému docker enginu

```
export DOCKER_HOST=ssh://UZIVATEL:IP_ADRESA
```

Jediné, na co je potřeba dbát, je, jestli server operuje na procesoru architektury x86 (CISC), nebo ARM (RISC). V případě jiné architektury procesoru, než na vývojovém počítači, máme 2 možnosti. Buď musíme sehnat jiný stroj s patřičnou architekturou procesoru pro sestavení Docker obrazů, nebo Docker obrazy sestavovat se speciálním build příkazem *buildx* a počítat s faktem, že bude sestavování probíhat trochu déle.

Výpis kódu 3.7 Příkaz pro sestavení Docker obrazu Livebidu na jinou architekturu procesoru

```
DOCKER_BUILDKIT=1 docker buildx build \
  --build-arg ENVIRONMENT=development \
  -t "$REPO_URL:latest" -t "$REPO_URL:$TAG" \
  --platform=linux/amd64 . -f $DOCKER_FILE
```

3.3.2 Kubernetes

Vhodnější, než spoléhat při produkčním prostředí na nahrávání do docker enginu, je využít vhodný nástroj, který obstarává docker kontejnery za nás. Při nasazení aplikace by bylo vhodné, kdyby zapínání nové verze nezpůsobilo dočasný výpadek služby pro uživatele. To by šlo udělat tak, že by při zapínání nové verze aplikace stále běžela stará verze a provoz se přehodil ze staré na novou až při dokončeném zapínání, terpve po kterém by se stará verze vypnula.

Kubernetes (zkráceně K8s) je nástroj pro správu kontejnerizovaných služeb a plánovaných úloh, původně vyvíjený společností Google. Kubernetes je skvělý nástroj na spravování běžících služeb v produkčním prostředí. V našem případě pro nahrávání nových verzí aplikace bez výpadku. Kubernetes také zvládne spustit více instancí (v kontextu K8s nazývaných *pody*) aplikace pro horizontální škálování a dokonce je schopný automaticky škálovat počet běžících podů podle vytíženosti služby.

Nevýhodou Kubernetes je složité zprovoznění na serveru a příliš zbytečně rozsáhlá konfigurace nasazení a služeb. Z tohodo důvodu existují i služby zaměřené na správu používání Kubernetes, každopádně při úspěšném zprovoznění základního nastavení je poté používání vcelku jednoduché.

3.3.3 Poskytovatelé cloud služeb

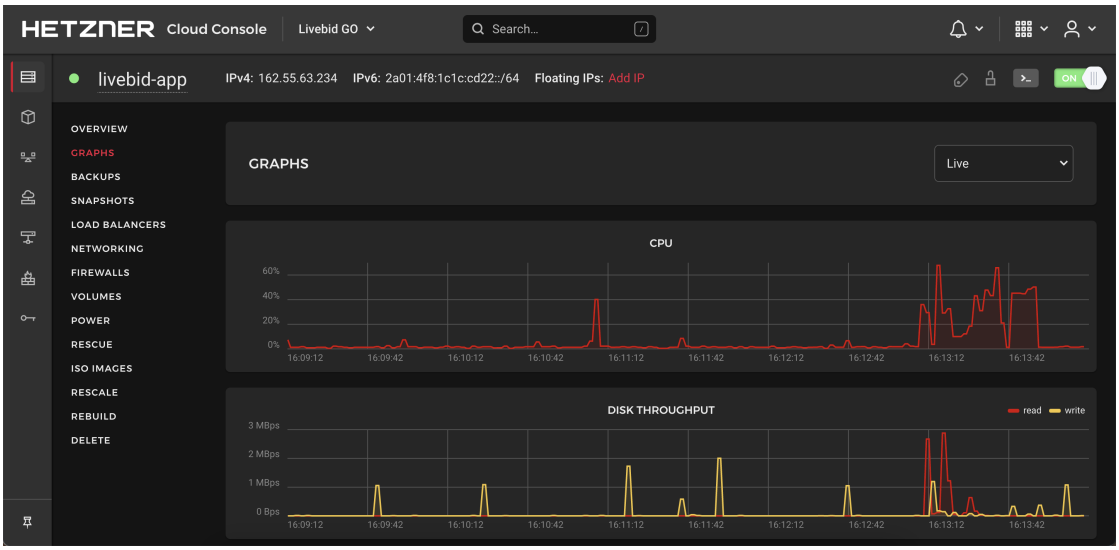
Pro hostování online aplikace konkrétního projektu je vhodnější využít nějakou službu poskytující hostování aplikace, než se snažit provozovat server na vlastním hardwaru, obzvlášť byl by umístěn v domácnosti. Kromě potřeby stabilní elektřiny a internetu, dostatečného chlazení počítače, které může být dost hlasité, až po hlídání, jestli je operační systém aktuální a nevyskytuje se na něm zastaralý podpůrný software, ve kterém se objevily nebezpečné zranitelnosti.

Navíc u poskytovatelů cloudových hostingů je často součástí služby alespoň základní monitoring vytíženosti serveru / aplikace. Škálování je mnohem jednodušší (obvzlášť u prostředků tzv. ”on demand”), protože například u následujících služeb se platí vždy pouze za skutečně spotřebované sekundy. Když se rozhodne, že má být server zrušen, nebo naopak rozšířen o další instance, cena je lineární.

3.3.3.1 Hetzner

Hetzner je jeden z klasických poskytovatelů VPS (Virtual Private Server) hostingu, který je obecně nejvíce náročný na prvotní nastavení a dev ops, každopádně je nejlevnější pro dlouhodobý běh aplikace.[25] Nabízí možnost vytvořit VPS s několika úrovněmi kombinací vCPU a RAM prostředků, počínaje na 1 vCPU a 2GB RAM za 4.22€ včetně DPH. Tvorba nového VPS nabízí možnost vybrat OS s předinstalovaným docker enginem, automatické přiřazení administrátorova SSH klíče, přidání do virtuální sítě VPS v rámci Hetzneru, či firewall nastavitelný v prohlížeči.

V rámci Hetzner hostingu je také možné si vytvořit spravovanou load balancer instanci. Ta se stará o přerozdělení požadavků z aplikace mezi několik VPS služeb, které se za ní ukrývají. V případě používání Kubernetes je tento loadbalancer zbytečný, protože si load balancování obstarává Kubernetes sám. Ideálnější by byla možnost, kdyby Hetzner nabízel možnost cloudové spravované Kubernetes služby, což nabízí ze stejné třídy VPS hostingů například DigitalOcean[26], nebo Linode[27]. Ti dále kromě Hetzneru nabízejí spoustu dalších hostovaných služeb, jako například spravovanou databázi, každopádně jsou zase o krok dražší než Hetzner. Obecně je však Hetzner dostačující pro levnější hostování serverových aplikací.



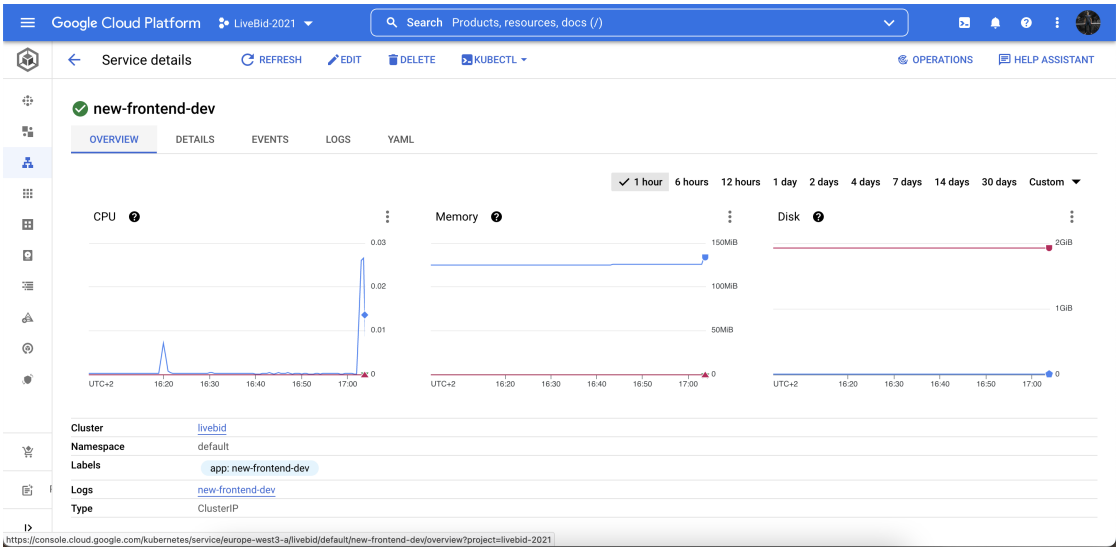
Obrázek 3.4 Přehled vytížení serveru na Hetzneru

3.3.3.2 Google Cloud Platform

Google Cloud Platform (zkráceně gcloud, nebo gcp) je jeden ze tří největších poskytovatelů cloudových služeb (další jsou AWS a Azure).[28] Nabízí spoustu různých cloudových služeb. Mimo klasického VPS hostingu nabízí například i cloudové funkce, spravovanou databázi a kubernetes, konkrétní nastavení monitoringu služeb a hlášení neobvyklých hodnot, cloudové úložiště, atd. Monitoring vytížení, včetně výpisu do konzole, je mnohem detailnější než u Hetzneru, jelikož lze vidět pro celý kubernetes cluster, pro jednotlivé služby, nebo dokonce jen pro jednotlivé pody.

Kubernetes a databázová služba je vytvořena na pár kliků, dlouhodobě bude stabilnější než vlastní hosting, jsou dostupné vizualizace jednotlivých služeb a poskytovatel se sám stará o aktualizace a hladký chod obou dvou služeb. Navíc v případě databáze poskytuje každodenní zálohy. Cloudové úložiště je další výhodou. Soubory jsou bezpečně uloženy, ale stále k nim máme přímý přístup z prohlížeče, včetně možnosti šifrování, historie verzí souborů, atd.

Google nabízí 300 dolarů na odzkoušení Google Cloudu pro první 3 měsíce používání, obecně je však Google Cloud mnohem dražší než klasické VPS hostingy, jako je například Hetzner. Google cloud dokáže být levnější v případě správného využití cloudových kapacit, jako jsou například cloudové funkce, které si účtují pouze skutečně spotřebované prostředky při zavolání HTTP dotazu. Naopak nevýhodou cloudových funkcí je vytvoření závislosti na konkrétním poskytovateli, ať už to je AWS, Azure, nebo Google.



Obrázek 3.5 Přehled vytížení jednotlivé Kubernetes služby na Google Cloud

3.3.3.3 Heroku

Heroku je cloudový hosting typu PaaS (Platform as a Service).[29] Podporuje nasazení aplikace s minimální konfigurací s každým novým commitem na master branchi v gitu. Například v případě go pro nasazení jednoduché webové aplikace stačí použít proměnnou prostředí pro získání portu, který má aplikaci použít a aplikace je připravena pro nasazení. Stejně jednoduché je vytvoření nové databáze a její propojení s aplikací.

Pro vývojáře tedy představuje skvělý nástroj, jak se zaměřit více na tvorbu a experimentování, než na řešení běhu aplikace na vlastním serveru. Nejjednodušší hosting s malou databází, který není spuštěn celý měsíc, je možné na Heroku získat zcela zdarma, každopádně škálování služby začíná být velmi drahé. Proto je Heroku vhodné pro prototypování, nebo naopak služby, které se dokáží zaplatit samy.

Kapitola 4
Implementace

Implementace celého systému je rozdělena na 3 části:

- Frontendová aplikace
- Backend API modul
- Backend Live modul

4.1 Frontend

Frontendová NextJS aplikace se nestará pouze o sestavení statického webu, o což by se snažil například klasický Reactový projekt. NextJS nabízí sestavení stránek na straně serveru za běhu pro stránky, které jsou potřeba být neustále aktivní, jako například stránky obsahující ceny. Dále framework nabízí možnost vytvořit API endpointy, stejně jako má backendový server, což je využito například pro kontrolu přihlášení na veřejných vývojových stránkách, nebo pro proxy Sentry.io, protože s jeho výchozí adresou je blokovaný moderními prohlížeči.

4.1.1 Struktura projektu

Frontendová aplikace využívá výchozí strukturu NextJS frameworku, rozšířenou o vlastní preference.

- `docker` — docker a k8s
- `locales` — překlady
- `public` — veřejné soubory
  - `icons` — vektorové obrázky ikonek
  - `imgs` — ostatní obrázky
  - `static` — překlady statických stránek
- `scripts` — podpůrné skripty
- `src` — veškeré zdrojové kódy
  - `api` — vlastní API endpointy definované na frontendu
  - `components` — všechny použité komponenty na jednom místě
  - `hooks` — React hooky
  - `mocks` — mockované backend endpointy
  - `pages` — routování podle struktury souborů stránek
  - `stores` — state management úložiště
  - `styles` — stitches a sass styly

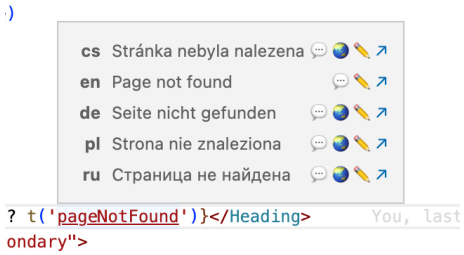
```
└─ types ..... Typescript typy chyb a dto
└─ utils ..... podpůrné kódy
```

4.1.2 Vícejazyčné překlady

Projekt je navržený pro mnohojazyčné používání, aktuálně jsou dostupné pouze čeština a angličtina. Překlady jsou obsaženy v adresáři ‘locales‘ a jsou rozděleny do několika jmenných prostorů, mezi nimiž jsou obecné, nastavení a statické stránky. Každý jmenný prostor má vlastní json soubor, ve kterém jsou překlady uloženy jako klíče objektu. Záznam s překladem může navíc obsahovat i proměnlivá data, doplněná podle správného názvu proměnné ve dvojitých složených závorkách.

NextJS samotný obsahuje podporu překladů na úrovni url adresy, kdy anglický překlad je pod stránkou ‘livebid.cz/en/auctions‘ a český bez podstránky ‘livebid.cz/auctions‘. Podporuje i možnost rozdělení na domény, kde livebid.com by obsahoval anglické texty a livebid.cz ty české. Pro práci s překlady je navíc využita knihovna next-translate. Ta implementuje i18n dosazení textů v NextJS, včetně okamžité změny textů při změně jazyku (bez přenačtení stránky).

Práci dosazování oboujazyčných textů do souborů s překlady mi ulehčovalo rozšíření I18n Ally. Umí v kódu rozpoznat překladové texty a zobrazit jejich překlad pro vybraný jazyk, zobrazit okno pro okamžité doplnění do souborů jazyků se správným klíčovým slovem překladu a při vyplnění anglické varianty textu dokáže přeložit texty do ostatních jazyků.



Obrázek 4.1 Překlady pro ostatní jazyky s I18n Ally včetně ikonek pro zahájení úpravy

4.1.3 Mockovaný backend

Mockovaný backend představuje falešnou náhradu backendu, který posílá data ve stejném formátu, ale dál s nimi pracuje pouze omezeně na straně uživatele. Mockování je vhodné pro rychlejší vývoj, protože v případě týmové práce není potřeba čekat na dokončení backendu, ale pouze se dopředu dohodnout na dodržovaném rozhraní. Rychleji se takto interují zapomenuté nedostatky aktuálního rozhraní mezi backendem a frontendem, protože jelikož není implementace skutečná, může se mnohem rychleji měnit a dostávat tak skutečným požadavkům frontendu, které pak může backend implementovat až s výslednou iterací rozhraní.

Mockovaný backend je využit kromě lokálního prostředí i na veřejně přístupném vývojovém prostředí hostovaném na vercel.com. Zdejší prostředí je vhodné pro rychlou týmovou iteraci designu aplikaci / funkcionality včetně nového rozhraní s backendem.

V neposlední řadě je mockovaný backend skvělý způsob, jak jednoduše otestovat funkcionalitu frontendu v unit testech. Některé mocky je možné využít tak, jak jsou použity viditelně na frontendu, což jsou obecně ty představující ideální průchod aplikací a pro chybové případy je často potřeba vytvořit konkrétní chybovou situaci (která by v takovém případě přišla z backendu) a hlídat, jak by se s ní frontendový kód vypořádal.

4.1.4 Testování

Testování frontendu aktuálně probíhá formou unit testů. Unit testy testují jednotlivé kusy kódu, zda pro požadovaný vstup vytvoří požadovaný výstup. Testují jak state management logiku, podpůrné funkce, tak i reactové komponenty, kde u těch komplexnějších a důležitějších zjišťuje, jestli podle aktuálního stavu zobrazují i nezobrazují správné věci. Například v případě Livebidu to je komponenta zobrazující u detailu aukce, jaké podmínky uživatel splňuje. Ta má spoustu různých kombinací, které je lepší pokrýt. Unit testy jsou spouštěné před každým vytvořením commitu společně s TypeScript statickou kontrolou kódu a vytváření commitu selže, pokud se objevila chyba. Tato kontrola je znovu spuštěna při pushnutí commitů v pipeline Gitlabu.

Každopádně unit testy samotné nedokáží zajistit, jestli systém skutečně funguje. Další fází jsou integrační testy. Představují testování se skutečnými závislostmi, například s databází, s jinou službou v rámci systému, atd. Integrační testy jsou však mnohem pomalejší než unit testy, proto se pouštějí méně často než unit testy.

Nejlepším způsobem, jak otestovat skutečnou funkčnost systému, jsou koncové (end to end) testy. V případě frontendu to může být falešně klikající skript s použitím knihovny Cypress, který prochází předem definované stránky a hledá, zda se správně zobrazují položky. Další variantou koncových testů jsou smoke testy. Smoke testy před nasazením zkoušejí, jestli systém na určitých místech reaguje správně. Pro koncové testy je potřeba mít spuštěný frontend, backend i databázi s namockovanými daty, v rámci kterých pak kontrola pracuje.

V rámci tohoto projektu žádné integrační ani koncové testy dostupné nejsou, ale v budoucnu by bylo vhodné pokrýt nejdůležitější případy aplikace s koncovými testy.

4.1.5 Vlastní UI komponenty

Ačkoliv se design inspiruje Bootstrapem, protože si vlastník Livebidu přeje zachovat stávající vizuál, UI komponenty jsou psané pomocí Stitches knihovny a složitější jsou kostrou podloženy knihovnou Radix UI (například výběr rozsahu na posuvníku). Komponenty jsou psané tak, aby je bylo možné kdykoliv nahradit jiným vzhledem / funkcionalitou / knihovnou, která ale bude splňovat stejné rozhraní pro použití komponenty v rámci systému. Celý projekt je tak připravený na budoucí přechod na nový, modernější design, který však budou doprovázet i změny v rozpoložení komponent na stránkách.

Komponenty jsou rozdělené na common, composed, custom, icons, layout a pages. ‘Common‘ obsahují základní prvky (jako například inputy, tlačítka, layouty, modal, ...), ze kterých lze vytvořit vlastní složené komponenty. V ‘composed‘ jsou také obsaženy základní komponenty, které jsou při použití složeny z více relevantních podkomponent. ‘Custom‘ jsou již složené komponenty, opakující se na více různých stránkách. V ‘pages‘ jsou složené komponenty relevantní pouze pro jednu stránku. Komponenty layoutů, použitých na každé stránce ve funkci getLayout, jsou obsaženy ve složce ‘layout‘. V poslední řadě je zde složka pro komponenty ikon, vygenerovaných z svg obrázků pomocí příkazu ‘yarn icons‘.

4.1.6 API dotazy

Každý dotaz na backend má kromě mockovaného endpointu na frontendu i vlastní typově zabezpečenou funkci, přijímající a vracející jasně definované hodnoty. Pro každý endpoint je možné nabídnout fetch, hook, prepare a prefetch funkci. Všechny tyto 4 varianty jsou definovány pouze jednou, v rámci definice dotazu a zpracování odpovědí ze serveru.

Fetch je asynchronní funkce čekající na výsledek ze serveru, vhodná pro jednorázový dotaz. Hook varianta poskytuje možnost, jak získávat neustále aktuální (nebo cachované) hodnoty. Podle konkrétní konfigurace pro aktuální endpoint je možné, aby hook znovu aktualizoval data při změně tabu, při obnovení internetu ze stavu offline, při aktivitě na stránce po několika sekundách, při vložení komponenty do DOMu nebo při změně jazyka stránky. Prepare připraví renderovaná

data pro SSR stránky. Prefetch je možné použít pro přednačtení endpointu na straně prohlížeče, například přednačte předchozí a další položku v rámci otevřeného detailu položky v modal okně.

4.1.6.1 Zpracování chyb

Jelikož TypeScript nenabízí možnost, jak přiřadit typy chyb vyhozených v rámci funkce (podobně jako to má Java), byla vytvořil vlastní strukturu pro předání výsledku, která jasně definuje, jaké chyby se mohou objevit při volání této funkce. Tato vlastní datová struktura je v rámci systému pojmenována Success. Obsahuje buď očekávaný výsledek ‘success.data‘, nebo typově ohraničenou chybu ‘success.error‘, což může být zjištěno kontrolou vlastnosti ‘success.ok‘ (pravda pokud success obsahuje očekávaný výsledek).

Vyhodnocení výsledku z endpointu je zabaleno do funkce Success resolve, která očekává jako návrat jeden ze stavů výsledné hodnoty: ok, error, fail. S ‘ok‘ je vše v pořádku, byla navrácena očekávaná hodnota. V případě ‘error‘ je navrácena jedna z definovaných uživatelských chyb (TypeScript zde dovolí vložit pouze očekávané chyby). ‘Fail‘ nastane v případě, že se objevila neošetřená chyba a záznam je nahlášen na Sentry.io pro prozkoumání s bližšími detaily.

Výpis kódu 4.1 Příklad funkce pro volání endpointu s typově ošetřenou návratovou hodnotu

```
export const apiChangePassword = async ({
  currentPassword,
  newPassword,
  recaptcha,
}: ChangePasswordProps): ResolvedSuccess<boolean, TooManyAttemptsError |
WrongLoginError> => {
  return Success.resolve(async ({ ok, error, fail }) => {
    const res = await fetch(`${API_URL}/user/settings/login/password`, {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json',
        Recaptcha: recaptcha,
      },
      body: JSON.stringify({
        oldPassword: currentPassword,
        newPassword,
      }),
    })

    if (res.status === 400) {
      return error(new WrongLoginError())
    }
    if (res.status === 429) {
      return error(new TooManyAttemptsError())
    }
    if (res.status >= 400) {
      return fail(generateApiError(res.status))
    }

    const json = await res.json()

    return ok(json)
  })
}
```

4.2 Backend API modul

Backendový API modul obstarává veškerou business logiku pro aplikaci, kromě živé aukce. Frontend přes HTTP dotazy komunikuje s backend serverem a získává tak jak data pro zobrazení na frontendu, tak způsob, jak provádět změny. Při provádění změn, ať už to je registrace, nebo vložení limitu, jsou vždy kontrolované vstupní parametry, stejně jako na frontendu. Na frontendu jsou vstupní požadavky kontrolovány pro uživatelskou přívětivost. Na backendu jsou kontrolovány za tím účelem, aby nebylo možné vkládat nepovolené příkazy HTTP požadavkem napřímo.

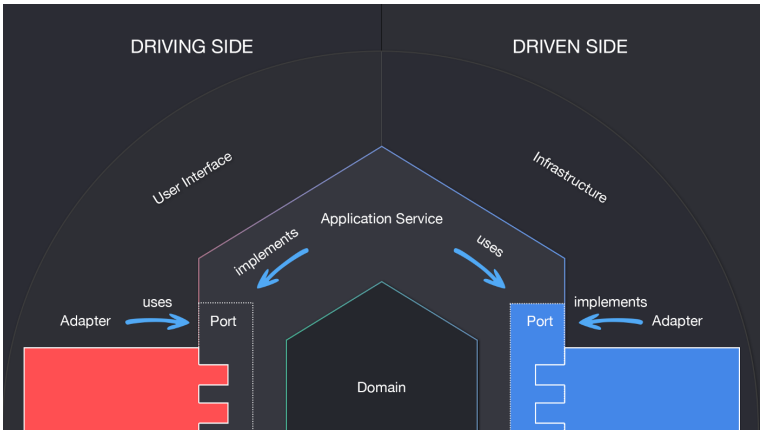
4.2.1 Struktura projektu

Adresářová struktura backendové aplikace využívá běžně používanou strukturu aplikací psaných v jazyce Go. V rámci adresáře 'internal' využívá vlastní adresářovou strukturu kompatibilní s hexagonální architekturou.

- `assets` — ostatní zdroje
- `cmd` — spouštěcí main kódy programů
  - `api` — spuštění modulu api backendu
  - `live` — spuštění modulu živé aukce
  - `seed` — spuštění seedování databáze
- `configs` — konfigurační soubory
- `deployments` — soubory související s nasazením
- `internal` — zdrojové kódy aplikace
  - `app` — obecné věci pro spuštění aplikací
  - `dto` — definice přenosového formátu objektů
  - `domain` — definice doménových / databázových modelů
  - `services` — služby rozdělené podle zodpovědností
  - `utils` — podpůrné zdrojové kódy
- `scripts` — podpůrné skripty

4.2.2 Hexagonální architektura

Backendová aplikace dodržuje volnější hexagonální architekturu. Hexagonální architekura je architektonický vzor inspirovaný vícevrstvou architekturou. Vícevrstvá architektura je rozdělena na vrstvy (například 3 vrstvá struktura: komunikační vrstva - logická vrstva - datová vrstva), kde každá vrstva komunikuje pouze s vrstvou pod ní, čímž si mezi sebou rozdělily své role.



Obrázek 4.2 Ilustrace toku v hexagonální architektuře[30]

Hexagonální architektura, také označována jako ports and adapters, se skládá z business (doménové) logiky představující jádro aplikace.[30] Jádro aplikace definuje rozhraní pro porty, ke kterým se lze připojit pomocí adaptérů. Jako adaptér je vnímaný například repozitář persistentně ukládající data, nebo controller pro API endpoint.
Příklad rozdělení adresáře v ‘services‘ pro *auth* službu:

```
auth.....................................................služba pro ověření a registraci uživatele
├── api.go ..............................................adaptér s api controllerem
├── config.go ...........................................konfigurační soubor pro tuto službu
├── constants.go ........................když počet konstant narůstá, mají vlastní soubor
├── mocks.go ..............................mockované varianty určené pro unit testování
├── repository.go ..................................repozitář pro práci s SQL přes GORM
├── service.go ..........................................................business logika
└── service_test.go ..................................... unit testy pro busines logiku
```

Práce s adresáři je v jazyce Go obecně jiná než například v Javě, nebo TypeScriptu. Adresář v Go představuje balíček, v rámci kterého všechny objekty vidí ostantí objekty (i private). V případě využití veřejných objektů z jiného balíčku se musí v rámci souboru tento balíček nejprve importovat, a poté název složky používat jako předpona s tečkou pro využití zdrojů z tohoto balíčku, například pro struct Kniha z balíčku models se bude volat ‘models.Kniha‘. Z toho důvodu se objekty pojmenovávají tak, aby nedocházelo k tzv. ”koktání”, například ‘models.ModelKniha‘.

4.2.2.1 Jádro

Soubor ‘service.go‘ obsahuje logiku a definuje rozhraní portů, využívajících principu dependency inversion. Za těmito rozhraními se schovávají skutečné závislosti, importované pomocí dependency injection. Kromě toho jsou zde také definované vlastní chyby, které v rámci této služby mohou nastat.

Doménové modely jsou všechny definované v rámci adresáře ‘entities‘, mimo adresář ‘services‘, aby mohly volně odkazovat na ostatní modely s cyklickou závislostí. Například aukce odkazuje na položky a položka zase zpátky na aukci.

4.2.2.2 Adaptéry

Adaptéry jsou definovány v ostatních souborech v rámci tohoto balíčku. MySQL repozitář je definován v rámci souboru ‘repository.go‘, který implementuje rozhraní definované v ‘service.go‘. API controller je definován v souboru ‘api.go‘, který odkazuje na rozhraní business logiky, de-

finované zase v souboru ‘service.go‘. Narozdíl od repositáře není controller definován pomocí rozhraní, protože je na něm závislý pouze router, který očekává konkrétní HTTP implementaci.

4.2.3 Dependency injection

Implementace backend kódu využívá techniky dependency injection. Ta rozšiřuje princip *inversion of control*. Objekt si nevytváří závislé objekty uvnitř sebe, ale jsou do něj vloženy například přes konstruktor. Díky tomu se nemusí kód objektu zaměřovat na životní cyklus závislostí a je možné testovat objekt s mockovanou závislostí.

Dependency injection je možné provádět manuálně, nebo je možné využít knihovnu, která tuto práci bude řešit automaticky. Proto je na backendu použita knihovna Wire pro automaticky řešené dependency injection prováděné před samotnou kompilací. Jednou ze zásad idiomatického go kódu je ”žádná překvapení”, kód by měl být nudný a předvídatelný.[31] Proto knihovna Wire doplňuje závislosti pomocí příkazu, který podle dostupných konstruktorů vygeneruje kód, vypadající jako manuální dependency injection. Vývojář přesně vidí, jak za sebou jde posloupnost volání konstruktorů a neprobíhá žádné dosazování na pozadí v runtime prostředí.

4.2.4 Migrace dat

U projektu modernizace webové aplikace Livebidu je potřeba počítat s tím, že již má stávající databázi. Ta každopádně není kompatibilní s nově navrženou databází, takže byl vytvořen skript, který data ze staré databáze převede na nová. Ten je napsaný v JavaScriptu a jeho výstupem je jeden velký SQL soubor, pomocí kterého lze importovat stávající data na novou verzi webu. Ty jsou pak seedována do databáze s potřebným databázovým schématem.

Knihovna GORM podporuje automatickou migraci databázového schématu. Ta bohužel nepodporuje verzování migrací, což znemožňuje například rollback schématu databáze na předchozí verzi a obecně větší kontrolu nad migrovaným schématem.

Verzované migrace je možné provádět pomocí knihovny *migrate*. První migraci je možné vytvořit pomocí GORM automigrate příkazu (SQL query), další migrační soubory už je potřeba spravovat manuálně. Obecně jsou migrace mnohem lépe řešeny například v ORM v jiných jazycích, například Prisma, TypeORM, Django, atd.

4.3 Backend live modul

Modul živé aukce se stará o obstarávání živých aukcí s komunikací přes websockety. Tento modul využívá sdílený kód s API backendem, jelikož jsou oba definované ve stejném kódu, pouze jsou jinak poskládané v rámci spouštěcího kódu v ‘cmd‘ adresáři.

4.3.1 Zahájení spojení

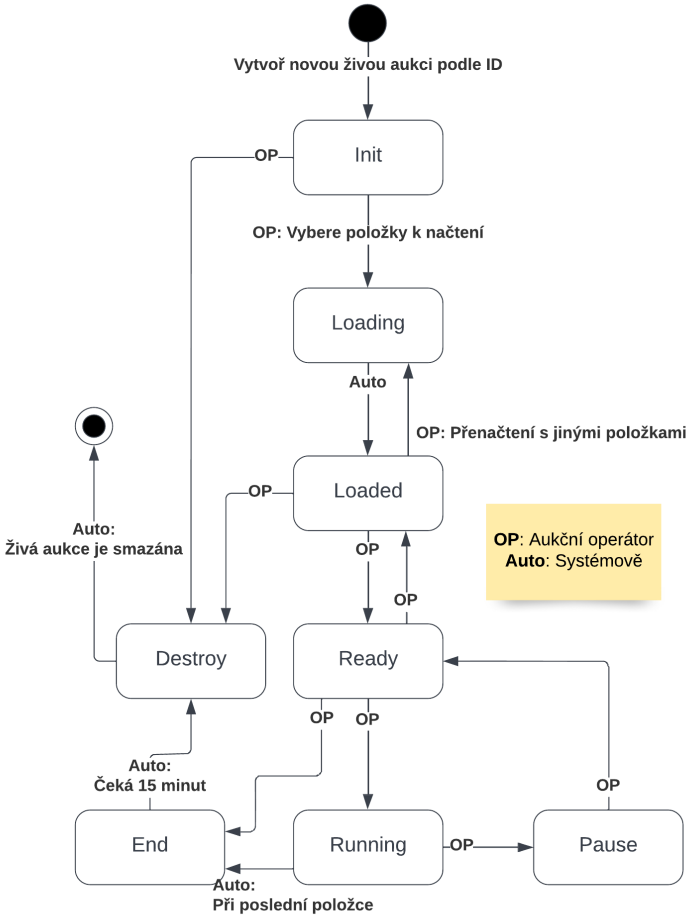
Live modul čeká na zahájení komunikace přes HTTP s upgrade požadavkem na websocket spojení. V hlavičce HTTP dotazu se v rámci cookies nachází autorizační token, podle kterého jsou vpuštěni do aukce pouze přihlášení uživatelé. Podle požadované url je uživatel přepojen do určité místnosti, na výběr má z možnosti overview a auction. U auction je ještě potřeba specifikovat ID aukce, celková požadovaná url vypadá následovně ‘/room/auction/700‘.

Výpis kódu 4.2 Ukázka HTTP dotazu o navázání websocket spojení

```
GET /room/overview HTTP/1.1
Host: livebid.cz
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: XJn26mONd8l9xHhbq8fNFA==
Sec-WebSocket-Version: 13
Cookie: nest.sid=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb9xlcyI6W14s...
```

4.3.2 Životní cyklus místnosti

Overview místnost je vytvořena automaticky systémem při spuštění a slouží pro zobrazování aktuálně běžících aukcí na veřejně dostupné části webu. Místnosti pro aukce jsou však vytvořeny, až když o ně nějaký uživatel požádá, včetně aukčního operátora. Bez spuštění aukce aukčním operátorem aukce nijak nereaguje a při odpojení posledního uživatele se zase smaže (nebo 15 minut po ukončení aukce). Živá aukce začíná, až když jí aukční operátor načte s požadovaným rozsahem aukčích položek.



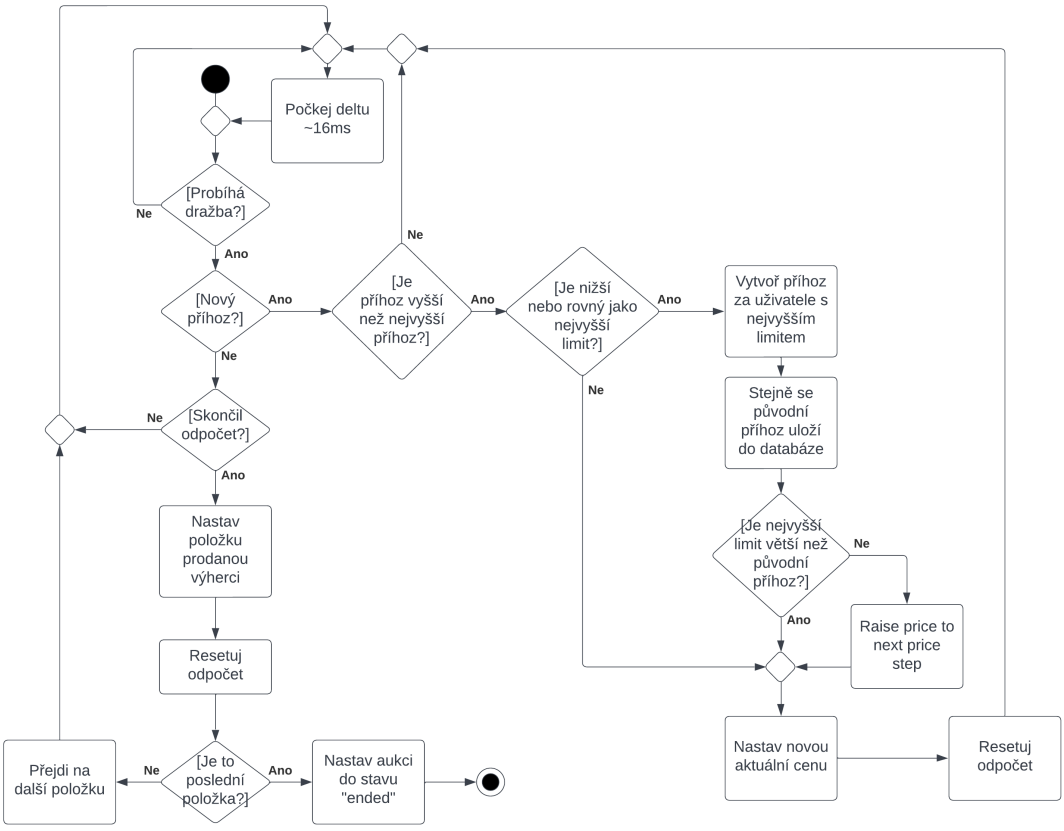
Obrázek 4.3 Průchod stavy živé aukce

4.3.3 Ověření rolí

Backend při živé aukci rozlišuje klasické uživatele a aukční operátory, kteří získávají navíc tajné informace o průběhu aukce, jako například jaké je jméno aktuálního výherce položky, nebo jaká je výše nejvyššího limitu. Server rozlišuje, zda je informace pro všechny uživatele, nebo individuálně pro každého zvlášť. Server přes websockety zasílá pouze informace, které se pro uživatele změnily, což si může dovolit díky vlastnosti websocketů, že jsou přenášené pomocí TCP paketů.

4.3.4 Synchronní fronta

Běh aukce je ve smyčce vyhodnocován každou dvacetinu sekundy. Příhozy na aktuální položku jsou vyhodnocovány v synchronní frontě, kdy si systém zkontroluje, jestli proti příhozu ještě nemá vznést protinávrh limitem. V takovém případě má výherní prioritu uživatel s limitem. Toto řešení je použito, aby během dražby nevznikaly nekonzistence způsobené fungováním ve více vláknech. Synchronní fronta je v rámci jazyka Go implementována pomocí kanálů.



Obrázek 4.4 Průběh vložení nového příhozu v živé aukci

4.3.5 Message bus

Každá změna v živé aukci je předávána do overview místnosti, ty však mezi sebou nemají žádný přímý kontakt. K přenosu informace mezi nimi je použit message bus, kde se jedna strana může přihlásit k odběru změn určité události a druhá strana vysílá zprávy o změně této události. Reálně se k odběru a vysílání těchto zpráv může přihlásit kdokoliv. V rámci této sběrnice mohou chodit

i jiné druhy událostí a v případě mikroslužeb může být implementace této sběrnice nahrazena službou, jako je RabbitMQ, nebo Kafka, které řeší message bus na síťové úrovni.

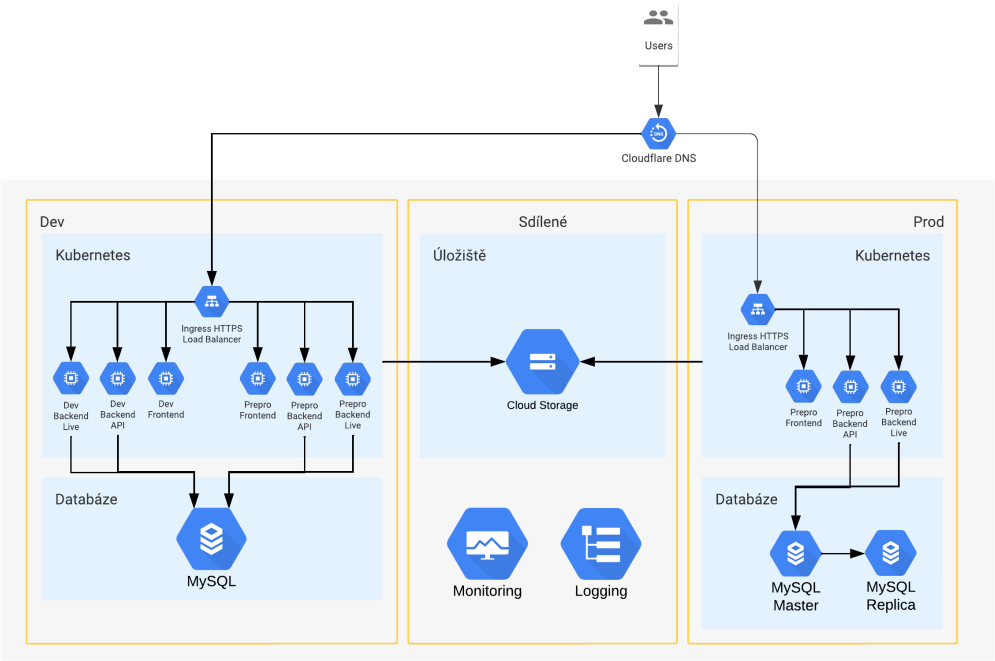
4.3.6 Vysoká dostupnost

Pro živou aukci je velice kritické, aby byla stabilní, protože její (opakovaný) pád v průběhu dražby může způsobit zničenou pověst značky LiveBid. Proto je modul živé aukce navržen tak, že mu stačí dostupná databáze jen pro prvotní načtení informací o načtených položkách. Pokud během aukce bude databáze zpomalena, nebo zcela vypadne, aukce bude pokračovat bez problémů dále.

Nové příhozy jsou do databáze zanášeny asynchronně. Průběh aukce se na jejich vykonávání nezastaví. V případě nedostupné databáze jsou příhozy dále zanášeny kromě do konzole i do tzv. černé skříňky, což jsou záznamy v textové podobě, získatelné na konci aukce. Uživatelé vstupující do aukce jsou ověřeni ”offline”, podle jejich JWT authentizace. Jediný problém během výpadku databáze budou mít uživatelé, kteří se budou chtít přihlásit na veřejné části webu, aby se mohli účastnit.

4.4 Cloud hosting

Projekt je hostován na Google Cloud Platformě, z důvodu využití stabilních spravovaných služeb, kvalitního monitoringu a logování zpráv v konzoli, včetně hlášení nastalých chyb na backendu. Frontend a backend jsou spuštěny v kubernetes clusterech, rozdělených na produkční prostředí a vývojové prostředí. Stejně tak je rozdělena spravovaná SQL databáze na 2 prostředí, kde produkční databáze má navíc 1 repliku pro čtení dat. Obrázky aukcí a položek jsou ukládány na sdílené úložiště Google Cloud Storage. Úložiště je rozděleno na 2 buckety, veřejně přístupné soubory a soukromé, šifrované soubory. Na provozované služby je nastaven monitoring, který v případě nabytí vysokých hodnot hlásí závadu do Slackového kanálu *#monitoring*.



Obrázek 4.5 Graf rozložení služeb v GCP

Kromě produkčního prostředí jsou na Google Cloud Platformě také hostovaná vývojová prostředí (*dev*) pro testování nových funkcionalit v cloud prostředí a staging prostředí (*prepro*) pro připravení na nasazení do produkčního prostředí. Kromě těchto je také dostupné zdarma prototypovací prostředí na Vercelu, které je díky rychlejšímu deploy procesu a verzování jednotlivých nahraných nasazení využíváno pro sdílení UI/UX návrhů, mezi kterými lze historicky zpětně procházet.

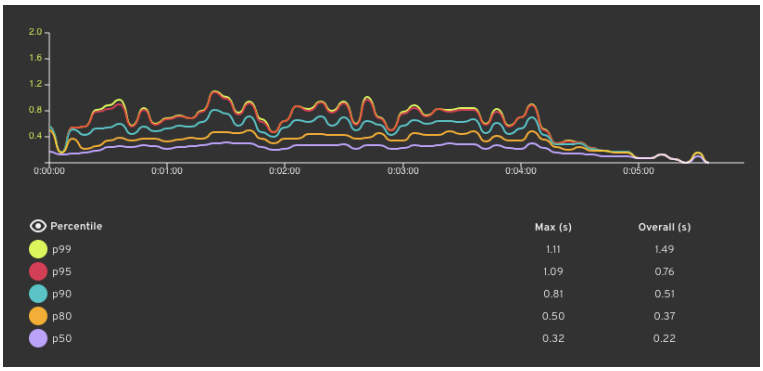
Nasazení nové verze na Google Cloud Platformu probíhá aktuálně pouze manuálně, vytvořením nového docker image frontendu / backendu a aplikováním na kubernetes cluster příkazem 'kubectl apply'. Toto workflow je zjednodušeno spuštěním skriptu './deploy.sh'. Do budoucna bude vhodné implementovat nasazení nových verzí přes Gitlab CI/CD pipeline, po provedení potřebných testů.

4.5 Zátěžové testy

Zátěžové testy webové aplikace poskytnou představu o potřebných prostředcích pro fungování aplikace s určitým počtem uživatelů online. Livebid má na stávajícím webu nasazenou analytiku uživatelů pomocí Google Analytics, podle které se průměrný počet aktivních uživatelů ve vytížených večerních hodinách pohybuje zhruba okolo 120.

Zátěžový test byl proveden na platformě Loadster.[32] Tato platforma poskytuje testování spuštěné ve skutečných prohlížečích, probíhající podle předpřipraveného scénáře. Test byl nasimulován pro nepřihlášené uživatele v rámci veřejné části webu při procházení dostupných stránek. Test byl spuštěn po dobu 5 minut v 50 paralelních prohlížečích. Nižší číslo spuštěných prohlížečů, než kolik je průměrně aktivních uživatelů, bylo zvoleno z důvodu, že aktivní uživatelé nejsou z hlediska vytvořeného počtu požadavků za vteřinu tolik agresivní, jako zvolené testy. Testy byly spuštěny pro konfiguraci serveru webové aplikace s 3vCPU + 4GB RAM a databáze s 2vCPU + 4GB RAM.

Výsledky tohoto pětiminutového testu naznačují, že při zátěži 50 uživatelů aktivně proklikávajících veřejnou část webové aplikace, 99 % načítání proběhlo do 1.5 sekundy, 80 % proběhlo v čase okolo půl sekundy. Zátěž serveru webové aplikace byla 200 % CPU a zátěž databáze byla 150 % CPU, vytížení RAM bylo v rámci testu ve stejných řádech jako při stavu s 0 aktivními uživateli. Výsledné hodnoty tohoto testu se pohybují v očekávaných hodnotách. S touto konfigurací je dostupný prostor pro připojení dalších aktivních uživatelů, případně s rozšířením prostředů serveru a databáze se kapacity aktivních uživatelů lineárně zvýší ještě dále. V rámci tohoto testu nebylo dostupné cachování výsledků, které by násobně usnadnilo práci oběma serverům.



Obrázek 4.6 Percentily doby odezvy

4.6 Dokumentace

Dokumentace je důležitá jak z hlediska předání hotového projektu zákazníkovi, tak při nabírání nových vývojářů. Dokumentace projektu je rozdělena do několika kategorií, podle role čtenáře příručky. Příručky pro vývojáře a administrátora jsou dostupné v příloze.

4.6.1 Příručka pro vývojáře

Příručka pro vývojáře je pro frontend i backend pojata v ‘readme.md‘ souboru v rámci repozitáře (psaná pomocí MarkDown syntaxe). Kromě doporučení dodržovaného stylu kódu a struktury, se zde vývojář dočte, jak nainstalovat potřebné předpoklady, spustit projekt ve vývojovém prostředí a jak nahrát nové změny na cloudový hosting.

Vývojář má navíc přístup k vygenerované dokumentaci API endpointů ve specifikaci OpenAPI, dostupné v backend repozitáři ‘docs/swagger.yaml‘, vygenerované pomocí nástroje swagger z backendového kódu. Zde si může projít všechny endpointy a jejich přijímané / návratové hodnoty. Pomocí nástroje Swagger Editor je možné spustit lokální server, s kterým lze specifikace prohlížet pohodlněji v prohlížeči, ze kterého jdou i spouštět volání jednotlivých endpointů.

4.6.2 Příručka pro administrátora

Příručka pro administrátora obsahuje informace o konfiguraci cloudového prostředí a jak na toto prostředí nahrávat nové aktualizace webové aplikace.

4.6.3 Příručka pro uživatele

Do budoucna by bylo vhodné z případů užití sestavit i příručku pro uživatele ”Jak se používá Livebid.cz”a distribuovat jí online v rámci tzv. knowledge base systému a propagovat přímo na webu Livebidu.

Kapitola 5
Závěr

Cílem práce bylo navrhnout a implementovat modernizovanou verzi webu LiveBid.cz, včetně všech jeho původních funkcionalit, rozšířenou o nový způsob ověřování identity uživatelů pro účast v aukci. Nová implementace je připravena pro plný provoz nového webu a zajistí vlastníkovi webu stabilitu pro implementace nových budoucích funkcionalit, včetně mobilní aplikace.

Nejprve byla provedena analýza stávajícího fungování webu. Byly vytvořeny aktualizované funkční a nefunkční požadavky včetně případů užití pro modernizovanou verzi webu. Podle požadavků byl navržen doménový model a byly vytvořeny wireframy pro případy užití.

Použité technologie stručně popisují vybrané technologie a za jakým účelem byly vybrány právě pro tento projekt. Implementace se pak zabývá jejich využitím a tím, jak byly zaváděny jednotlivé části projektu. Provedený zátěžový test aplikace představil možný počet aktivních uživatelů. Nakonec jsou k projektu sepsané dokumentační podklady jako příručky pro vývojáře a administrátory.

Budoucí rozšíření budou zahrnovat modernizaci administrace, která nebyla součástí této práce. Dále bude vhodné vytvořit k systému mobilní aplikaci pro pohodlnější používání systému uživateli na cestách. Přínosné by pro uživatele bylo sepsání uživatelské příručky, popisující fungování celého systému z pohledu uživatele.

Příloha A

Dokumentace

Zde je přiložena dokumentace dostupná pro vývojáře a administrátory projektu. Dokumentace je psána v angličtině ve formátu Markdown (koncovka .md), který se čtenáři zobrazuje jako formátovaný text. Uvedena je však čistě v podobě Markdown, aby text nesplýval s formátováním textu práce.

A.1 Frontend

Dokumentace pro vývojáře Frontendu umístěna v rámci frontendového repozitáře jako soubor ‘readme.md‘.

Výpis kódu A.1 Dokumentace pro vývojáře v rámci Frontendu

```
# Livebid Frontend
Next version of Livebid frontend made to work together
with the new backend via API calls.

## How to run this app?
### Prerequisities
- Git
- NodeJS v14 and higher
- Yarn (‘npm install -g yarn‘)
- Prettier and ESLint extension in your IDE
- Also i18n Ally for VSCode is a nice extension.

For deployment you will also need:
- Docker
- Google Cloud CLI

### Instalation
- Install deps with ‘yarn install‘

### Configuration
To run this app, you need to configure it.
Configuration is handled via ‘.env‘ files.
For production there is ‘.env.production‘ file and
for development there is ‘.env.development‘.

Empty configuration can be copied from ‘.env.example‘ file.
```

```
Configuration files are loaded from root folder.

### Running
- Run dev server with command ‘yarn dev‘
- Page will be available at [http://localhost:3000](http://localhost:3000)

### Deployment
- Deploy with command ‘./deploy.prod.sh‘

If project / details change, deployment destination can be configured in:
- deploy.prod.sh
- docker/prod/k8s.yaml

### Additional
#### Localization
- Install localazy ‘npm install -g @localazy/cli‘
  (https://localazy.com/docs/cli/installation)
- Upload ‘localazy upload‘
- Download ‘localazy download‘

#### Generating icons
- To add new icon, place svg image with appropriate name
  into ‘public/icons/‘ folder
- Regenerate icons with command ‘yarn icons‘

#### Generating minified static pages for translation
- Static pages are located in ‘public/static/‘
- Generate min files with command ‘yarn pages‘
- Then copy them to corresponding ‘locales/‘

## Decisions
### Folder structure
- docker - docker & k8s
- locales - translations
- public - public files
  - icons - icon svg files
  - imgs - other images
  - static - static pages translations
- scripts - support scripts
- src - source codes
  - api - custom API endpoints defined on frontend server
  - components - custom UI components
  - hooks - React hooks
  - mocks - mocked backend endpoints
  - pages - NextJS structure based routing
  - stores - state management stores
  - styles - stitches & sass styles
  - types - Typescript types for errors and DTOs
  - utils - support functions
```

A.2 Backend

Dokumentace pro vývojáře Backendu (API i live modul) umístěna v rámci backendového repozitáře jako soubor ‘readme.md‘.

Výpis kódu A.2 Dokumentace pro vývojáře v rámci Backendu

```
# Livebid Backend

Livebid backend is split into 2 parts:
- API module
- Live auctions module

## How to run this app?
### Download
Before you start with this project, make sure you have
these dependencies installed:

- make (gcc)
- golang 1.18
- docker

Then run `make install`.

### Config
To run this app, you need to configure it.
Configuration is handled via `.env` files.
For production there is `.env.production` file and
for development there is `.env.development`.

Empty configuration can be copied from `.env.example` file.
Configuration files are loaded from `\configs` folder.

### Run Dev Server
`make dev` will start a dev server.

Or run only one module at the time:
- `make dev-api` to run only api module
- `make dev-live` to run only live auctions module

### Deploy
`make deploy` will deploy the app to the server.

## Decisions
### Hexagonal Architecture
This project is built on hexagonal architecture.
Service.go file in services contains core business logic and
defines ports (interfaces), which are implemented in
corresponding files in that service.

![Hexagonal Architecture Image](assets/images/hexagonal-architecture.png)

### Folder structure
- assets - resource files
- cmd - main entry points for programs
  - api - api module
  - live - live auctions module
  - seed - seed data
- configs - configuration files
- deployments - docker and k8s
- internal - project source code
  - app - shared application starting related code
```

```
  - dto - data transfer objects
  - entity - entities
  - services - core businnes logic
    - service name
      - api.go - http controller
      - config.go - related config
      - constants.go - constants
      - mocks.go - mocked adapters
      - repository.go - data persistency
      - service.go - business logic
      - service_test.go - unit tests for business logic
  - utils - helper functions
- scripts - support scripts
```

A.3 Administrátor

Dokumentace pro administrátora je umístěna v rámci backendového repozitáře jako soubor ‘cloud.md‘.

Výpis kódu A.3 Dokumentace pro administrátora

```
# Cloud deployment
This document is about how to deploy and maintain
the web application in the cloud.

## Cloud hosting
Application is hosted at the Google Cloud Platform.
It’s deployed on Kubernetes cluster, splitted into
development and production environments.

Each environment has also its own managed MySQL database.
Image storage is shared and managed by Google Cloud Storage.

![Hexagonal Architecture Image](assets/images/gcp-architecture.png)

## How to deploy?
Frontend and backend are deployed separately.
Frontend contains ‘deploy.ENVIRONMENT.sh‘ scripts,
which deploy to specific environment. Backend is deployed
via ‘make deploy‘ command.

### Deployment configurations
- Dockerfile - Dockerfiles do not need to be changed
  (only when needed by developer).
- K8s manifests - K8s manifests contain
- deploy scripts - contains specification where to deploy
- ENV files - ENV files contain environment specific settings

### Environments
- **production** - prod - THIS MUST WORK AT ALL COSTS
- **staging** - prepro - here are new features tested
  before going to production
- **development** - dev - for development and new feature
  testing purposes
- **vercel** - vercel - for prototyping, not on GCP
```

Bibliografie

1. WORKER, Mock Service. *Mock Service Worker* [online] [cit. 2022-04-15]. Dostupné z: `https://mswjs.io/`.
2. NATURE. *Homo erectus made world's oldest doodle 500,000 years ago* [online] [cit. 2022-04-09]. Dostupné z: `https://www.nature.com/articles/nature.2014.16477`.
3. LIDI, Zákony pro. *Zákon č. 26/2000 Sb.* [Online] [cit. 2022-04-15]. Dostupné z: `https://www.zakonyprolidi.cz/cs/2000-26#`.
4. GETID. *KYC Identity Verification for Global Businesses* [online] [cit. 2022-05-06]. Dostupné z: `https://getid.com/`.
5. GURU99. *What is a Functional Requirement in Software Engineering? Specification, Types, Examples* [online] [cit. 2022-04-15]. Dostupné z: `https://www.guru99.com/functional-requirement-specification-example.html`.
6. GURU99. *What is Non-Functional Requirement in Software Engineering? Types and Examples* [online] [cit. 2022-04-15]. Dostupné z: `https://www.guru99.com/non-functional-requirement-type-example.html`.
7. FACEBOOK. *ReactJS* [online] [cit. 2022-04-05]. Dostupné z: `https://reactjs.org/`.
8. NEXTJS. *The React Framework for Production* [online] [cit. 2022-04-05]. Dostupné z: `https://nextjs.org/`.
9. MICROSOFT. *TypeScript* [online] [cit. 2022-04-05]. Dostupné z: `https://www.typescriptlang.org/`.
10. DAHL, Ryan. *Deno - A modern runtime for JavaScript and TypeScript* [online] [cit. 2022-04-05]. Dostupné z: `https://deno.land/`.
11. STITCHES. *Stitches - CSS-in-JS with near-zero runtime* [online] [cit. 2022-04-15]. Dostupné z: `https://stitches.dev/`.
12. SENTRY.IO. *Application Monitoring and Error Tracking Software — Sentry* [online] [cit. 2022-04-15]. Dostupné z: `https://sentry.io/welcome/`.
13. REDHAT. *What is a REST API?* [Online] [cit. 2022-04-15]. Dostupné z: `https://www.redhat.com/en/topics/api/what-is-a-rest-api`.
14. MESSAGEPACK. *It's like JSON. but fast and small.* [Online] [cit. 2022-04-15]. Dostupné z: `https://msgpack.org/`.
15. IETF. *The WebSocket Protocol* [online] [cit. 2022-04-15]. Dostupné z: `https://datatracker.ietf.org/doc/html/rfc6455`.
16. WIKIPEDIA. *Go (programming language)* [online] [cit. 2022-04-05]. Dostupné z: `https://en.wikipedia.org/wiki/Go_(programming_language)`.

17. GITHUB. *Fiber* [online] [cit. 2022-04-05]. Dostupné z: https://github.com/gofiber/fiber.
18. KINSTA. *What Is MySQL? A Beginner-Friendly Explanation* [online] [cit. 2022-04-15]. Dostupné z: https://kinsta.com/knowledgebase/what-is-mysql/.
19. STACKOVERFLOW. *Stack Overflow Developer Survey 2021* [online] [cit. 2022-04-15]. Dostupné z: https://insights.stackoverflow.com/survey/2021#overview.
20. ALTEXSOFT. *Understanding Object-Relational Mapping: Pros, Cons, and Types* [online] [cit. 2022-04-25]. Dostupné z: https://www.altexsoft.com/blog/object-relational-mapping/.
21. JINZHU. *GORM.io* [online] [cit. 2022-04-25]. Dostupné z: https://gorm.io/.
22. GOLANG-MIGRATE. *Database migrations. CLI and Golang library.* [Online] [cit. 2022-04-25]. Dostupné z: https://github.com/golang-migrate/migrate.
23. LOGIC, sumo. *Software Deployment* [online] [cit. 2022-04-25]. Dostupné z: https://www.sumologic.com/glossary/software-deployment/.
24. DOCKER. *Home - Docker* [online] [cit. 2022-04-05]. Dostupné z: https://www.docker.com/.
25. HETZNER. *Hetzner Cloud* [online] [cit. 2022-04-25]. Dostupné z: https://www.hetzner.com/cloud.
26. DIGITALOCEAN. *Kubernetes in Minutes* [online] [cit. 2022-04-25]. Dostupné z: https://www.digitalocean.com/products/kubernetes.
27. LINODE. *Fast and simple Kubernetes cluster deployments* [online] [cit. 2022-04-25]. Dostupné z: https://www.linode.com/products/kubernetes/.
28. GOOGLE. *Google Cloud overview* [online] [cit. 2022-04-25]. Dostupné z: https://cloud.google.com/docs/overview.
29. HEROKU. *Deploy and run apps on today's most innovative Platform as a Service* [online] [cit. 2022-04-25]. Dostupné z: https://www.heroku.com/platform.
30. MARTINEZ, Pablo. *Hexagonal Architecture, there are always two sides to every story* [online] [cit. 2022-04-29]. Dostupné z: https://medium.com/ssense-tech/hexagonal-architecture-there-are-always-two-sides-to-every-story-bc0780ed7d9c.
31. GO. *Effective Go* [online] [cit. 2022-04-29]. Dostupné z: https://go.dev/doc/effective_go.
32. LOADSTER. *Load Stress Testing for High-Performance Websites* [online] [cit. 2022-05-07]. Dostupné z: https://loadster.app/.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF