FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Optilynx - Informační systém pro optiky
Student:	Bc. Filip Glazar
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Navrhněte a realizujte informační systém pro středně velkou oční optiku.
Analyzujte potřeby oční optiky, jako partnera pro analýzu využijte společnost OČNÍ OPTIKA BENEŠOV s.r.o.
Na základě zjištěných požadavků navrhněte koncept uživatelského rozhraní, při návrhu spolupracujte s vedoucím a s Bc. Oldřichem Malcem.
Dle analýzy a návrhu uživatelského rozhraní diskutujte a zvolte vhodné technologie pro realizaci.
Navrhněte a implementujte backend informačního systému, na frontendu spolupracujte s Bc. Jaroslavem Hrachem.
Aplikaci nasaďte do testovacího provozu, odhalte nedostatky, realizujte vhodné úpravy a opravy.
Zhodnoťte celkový stav aplikace, navrhněte možná budoucí vylepšení se zaměřením na backend a datovou část aplikace.
Zajistěte nasazení produkční verze aplikace alespoň u jedné reálné oční optiky.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 14. února 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

OptiLynx – Informační systém pro optiky

Bc. Filip Glazar

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Hunka

8. května 2018

Poděkování

Především bych chtěl poděkovat panu Ing. Jiřímu Hunkovi za možnost zvolit si toto téma jako svoji diplomovou práci, za jeho zkušenosti a rady, bez kterých by tato práce nevznikla. Dále bych chtěl poděkovat Bc. Jaroslavovi Hráchovi za jeho odvahu podílet se na tomto projektu s námi. Poděkovat chci i všem svým kolegům, kteří se podíleli na vzniku této práce. Jmenovitě děkuji Bc. Oldřichu Malcovi, Bc. Pavlu Kovářovi a Ing. Janu Matouškovi. Sice jako poslední, ale za to obrovské poděkování patří celé mé rodině bez jejíž podpory bych tuto práci nikdy nemohl psát, natož dokončit. Děkuji.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické 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. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 8. května 2018 .....

Odkaz na tuto práci

Glazar, Filip. *OptiLynx – Informační systém pro optiky.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato diplomová práce se zabývá kompletním návrhem a vývojem informačního systému pro oční optiky. Cílem této práce je provést analýzu podle požadavků reálných provozoven a zaměstnanců očních optik. Na základě této analýzy navrhnout a implementovat vhodný informační systém. Dále práce popisuje nasazení systému do provozoven několika očních optik a problémy s tím spojené. Detailně rozebírá tyto skutečnosti a navrhuje jejich řešení. Důraz byl kladen především na kvalitní návrh a implementaci projektu. Aplikace se skládá ze dvou stěžejních částí, které jsou obě navrženy v této práci. Práce se zabývá především implementací backendové části. Tento software je napsán v programovacím jazyce PHP s použitím frameworku Laravel. Při realizaci byly použity moderní a populární technologie, které práce popisuje. Součástí realizace je celý vývojový cyklus komplexního informačního systému. Tento systém jsem uvedl do reálného provozu a pevně věřím, že plní všechny požadavky ze světa očních optik.

Klíčová slova informační systém, UI, PHP, analýza, návrh, provoz, REST

Abstract

This diploma thesis deals with the complete design and development of the information system for opticians. The aim of this work is to analyze the requirements of real ophthalmic facility and their staff. Based on this analysis, design and implement an appropriate information system. Further, the thesis describes the deployment of the system to the premises of several eye optics and related problems. It discusses these facts in detail and proposes their solution. Emphasis was placed on quality project design and implementation. The application consists of two core parts, both of which are designed in this work. The thesis deals mainly with the implementation of the backend part. This software is written in the PHP programming language using the Laravel framework. During the implementation, modern and popular technologies have been used.Part of implementation is complete the entire development cycle of a complex information system. I have put this system in real operation and I firmly believe that it meets all the requirements of the world of opticians.

Keywords information system, UI, PHP, analyse, design, operation, REST

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Úvod

Nepochybuji o tom, že každý z nás na sobě měl někdy brýle. Tato zdravotnická pomůcka již dávno překonala svůj původní účel a dnes se používá i jako módní doplněk. Brýle a jiné zdravotnické pomůcky se prodávají v očních optikách, pracovní náplň zaměstnanců těchto prodejen je ovšem daleko rozmanitější. Běžně prodávají příslušenství k brýlím, vyřizují opravy, reklamace a vykonávají mnoho dalších úkonů. V dnešní době se libovolná prodejna neobejde bez informačního systému. Právě vývojem informačního systému pro oční optiky, který co nejvíce ulehčí práci jejím zaměstnancům a zároveň bude moderní, rychlý a snadno udržovatelný, se zabývá tato diplomová práce.

Tato práce mi byla nabídnuta vedoucím, panem Ing. Jiřím Hunkou. Splnit všechny cíle této práce pro mě bylo velkou výzvou. Jedná se totiž o kompletní proces vývoje poměrně komplikovaného softwaru od analýzy až po nasazení systému do reálného provozu, což přesahuje běžnou agendu softwarového inženýra. Na druhou stranu právě díky volnosti při návrhu a možnosti obrovské seberealizace jsem toto zadání přijal. Zásadním faktorem této práce je i spolupráce s reálnou oční optikou, která bude tento systém používat.

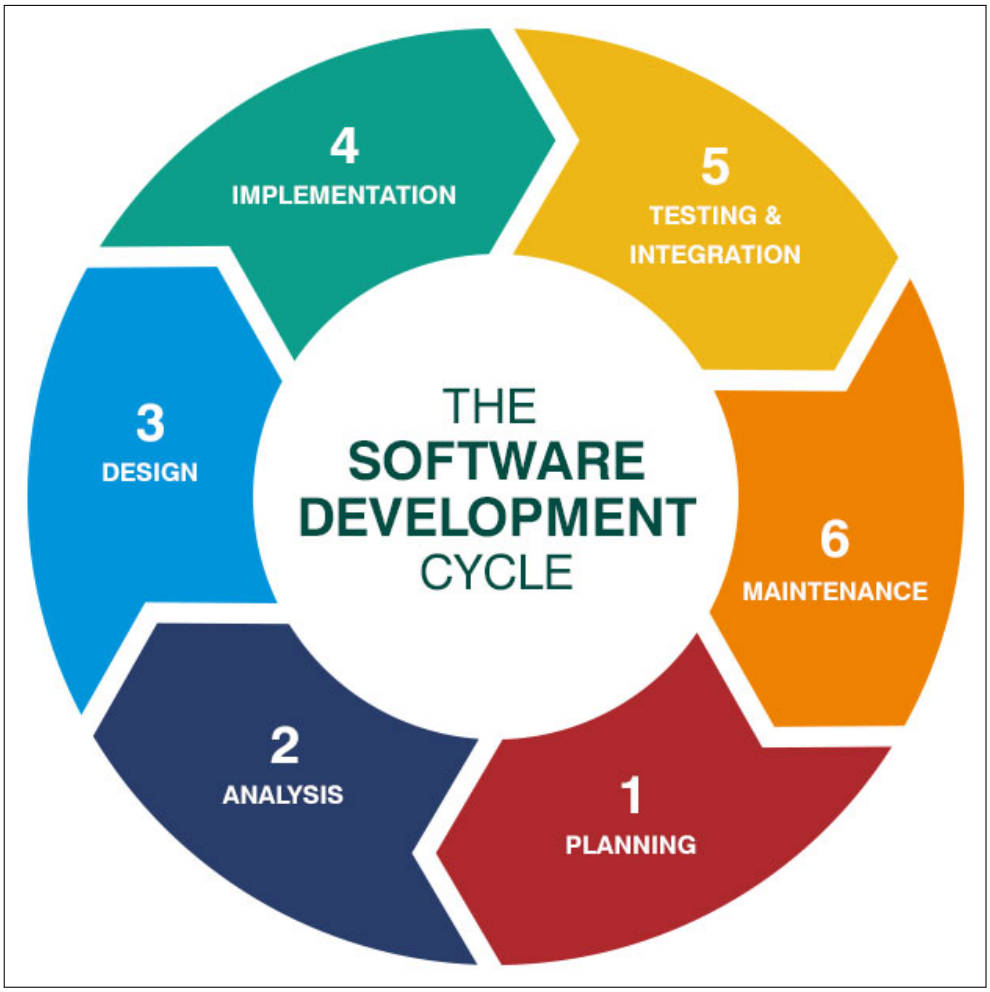
Systém se skládá z několika základních modulů, jako je standardní prodejní pokladna, která slouží i k výdeji zakázek. Další modul umožňuje tvorbu zakázky na brýle. Nechybí ani kompletní skladový systém a mnoho dalšího. Systém musí být navržen tak, aby umožňoval provoz na více pobočkách jedné oční optiky a byl natolik univerzální, aby ho mohla používat jakákoliv oční optika.

Strukturu této diplomové práce jsem zvolil následovně: V první kapitole Cíl práce podrobně rozeberu cíle této práce. V následující kapitole Analýza a návrh popíši, jak probíhala analýza, sběr požadavků a výstupem bude návrh celého informačního systému. Plynule se v kapitole Realizace přesuneme k popisu a ukázkám vlastní implementace, abych mohl v poslední kapitole Nasazení a provoz systému popsat nasazení a průběh ostrého provozu systému. V této části je k nalezení i shrnutí výsledků celého projektu a nahlédneme do blízké budoucnosti systému.

Kapitola 1

Cíl práce

Prvním cílem této práce je navrhnout a realizovat informační systém pro oční optiku. Splnění tohoto cíle je samozřejmě podmíněno i dalšími dílčími kroky, které jsou v této práci zahrnuty. K těmto krokům patří analýza potřeb očních optik, návrh uživatelského rozhraní a samotná implementace systému. Tato práce se věnuje hlavně implementaci backendové části IS. Zmíněna je i realizace uživatelského rozhraní, ale pouze okrajově a to z toho důvodu, že samotné UI řeší paralelně diplomová práce Bc. Jaroslava Hrácha. Druhým a neméně důležitým cílem této práce je nasazení systému do testovacího provozu, následné odstranění chyb a optimalizace systému jako takového. Posledním cílem je nasazení produkční verze systému do reálné provozovny oční optiky. Můžeme říci, že tato práce pokrývá kompletní cyklus vývoje softwaru.



Obrázek 1.1: Životní cyklus vývoje softwaru [1]

1.1 Analýza

Výstupem z této části práce bude souhrn požadavků, dle kterého budeme postupovat při návrhu systému. Z této analýzy vyplynou jak požadavky funkční, tak nefunkční. V rámci této analýzy vznikne také prototyp systému. Tento prototyp bude plnit i dílčí části ostatních cílů, jako je Návrh a Analýza.

1.2 Návrh

V návrhu bude popsána kompletní platforma a jednotlivé moduly společně s návrhem realizace a komunikace těchto modulů mezi sebou. Podrobně rozebereme volbu jednotlivých koncepcí návrhu a její vhodnost či nevhodnost pro použití v tomto konkrétním systému.

1.3 Uživatelské rozhraní

Cílem je pouze návrh UX bez konkrétní implementace, kterou řeší paralelní diplomová práce. Bude zde kladen důraz pouze na výměnu dat mezi systémy a jejich vzájemná kooperace. Tento návrh bude sloužit i jako podklad pro paralelní diplomovou práci Bc. Jaroslava Hrácha. Zároveň na tomto návrhu ukážeme, jak jsou pokryty funkční požadavky na systém, které jsme specifikovali v kapitole Analýza.

1.4 Realizace backendu

Výsledkem je realizace kompletního řešení pro datovou část aplikace. Jedná se jak o samotné uložení dat, tak jejich distribuce ostatním vrstvám platformy. Dále bude zahrnut vývojový proces, distribuce a dokumentace systému. Výstup bude samozřejmě obsahovat i konkrétní dokumentaci, která poslouží pro orientaci při budoucím vývoji systému.

1.5 Reálný provoz

Jedná se o popis nasazení systému jako celku do reálného provozu a reakce na nalezené nedostatky či chybějící části. V případě nalezení takovýchto částí bude realizována jejich implementace nebo navrženo řešení. Zároveň zde nalezneme i zhodnocení celého vývoje jako takového. Speciálně se zaměříme na části systému, které neodhalila analýza.

1.6 Doporučení

Závěrem se budeme zabývat budoucností celé platformy a jasným vymezením následujícího postupu pro evoluci celého systému.

Kapitola 2

Analýza a návrh

2.1 Členové týmu a jejich role

Vývoj takto komplexního systému, který je předmětem této práce, není možné uskutečnit pouze jednou osobou. Minimálně to je časově velmi neefektivní. Na úvod této kapitoly vám představíme členy našeho vývojového týmu a jejich role při vývoji a provozu systému samotného. Vzhledem k agilnosti všech členů našeho týmu se v jednotlivých rolích objevují průniky téže osoby.

2.1.1 Vznik týmu

Tým, který se podílel na vývoji, vznikl pod záštitou firmy Jagu s. r. o. Jednatelem této firmy je Ing. Jiří Hunka, který je vedoucím této práce. Se členy týmu spolupracujeme už na několikátém projektu, a díky tomu máme již připravené všechny standardní procesy vývoje a rozdělené role v týmu. K organizaci činnosti jednotlivých pracovníků jsme používali specializovaný systém pro správu projektu Redmine.[2]

2.1.2 Analytik

Hlavní zodpovědnost analytika je tvořit most mezi vývojáři a klientem. Úkolem je především zjistit požadavky klienta a předat je ostatním členům týmu. Tento informační tok by měl být zprostředkován i opačným směrem. Část této zodpovědnosti by měl nést i doménový specialista, kterého v tomto případně nahrazoval klient samotný. Hlavním analytikem v rámci tohoto projektu je vedoucí této práce, Ing. Jiří Hunka. Ten v pozici supervizora dozoroval nad výsledky analýzy, kterou jsem prováděl společně s kolegou Bc. Oldřichem Malcem.

2.1.3 Architekt

Úkolem architekta je přenést požadavky systému, a to jak funkční tak nefunkční, na konkrétní návrh samotné implementace a její technickou specifikaci. Této role jsem se ujal já a moje činnost je podrobně rozepsána v následujících kapitolách. Velmi specifická pro tento projekt byla potřeba vytvořit architekturu úplně od začátku. Toto mi sice umožňovalo obrovskou volnost, ale každá chyba v návrhu by se velmi negativně projevila při reálném provozu systému.

2.1.4 Designér

Design je velmi široký pojem, můžeme se bavit o designu jako návrhu samotné softwarové implementace, o návrhu UX nebo o konkrétním vzhledu GUI. Role designéra je v tomto projektu rozdělena do dvou konkrétních rovin. První rovinou je základní návrh UX, jako jsou jednotlivé obrazovky aplikace a rozmístění konkrétních ovládacích prvků. Této role jsem se ujal já společně s kolegou Bc. Oldřichem Malcem a konzultantem byl samozřejmě vedoucí. Druhou rovinou byl konkrétní návrh frontendu aplikace, kterého se zhostil Bc. Jaroslav Hrách v rámci své diplomové práce.

2.1.5 Developer

Samotnou implementaci dle požadavků architekta jsem jako vedoucí vývojář z velké části realizoval já s pomocí mnoha kolegů, kteří pracovali na okrajových částech systému mnou navržených. Tuto spolupráci a její výsledky jsem dozoroval. Výjimku tvoří frontendová část systému, kterou realizoval Bc. Jaroslav Hrách a já pouze vypomáhal s některými konkrétními částmi. Dílčí implementace byly provedeny Bc. Pavlem Kovářem a Ing. Janem Matouškem. O konkrétních částech se rozepíšeme v kapitole Realizace.

2.1.6 Projektový manažer

Projektový manažer je nepostradatelnou části týmu. Můžeme říci, že drží celý projekt pohromadě. Jeho hlavní činností je organizace kooperace jednotlivých vývojářů a hlídání dodržovaní termínů. Této role se ujal Bc. Oldřich Malec a zpětně mohu konstatovat, že svoji úlohu plnil na výbornou.

2.1.7 Tester

Tuto pozici zastávají všichni členové týmu. Systém byl jak v průběhu celého vývoje, tak po nasazení do provozu neustále testován. Plánujeme tuto pozici obsadit specializovanou osobou na tuto problematiku a pro ulehčení práce implementovat automatizované testy.

2.1.8 Podpora

Protože systém se v jisté fázi vývoje nasadil do reálného provozu, vznikla potřeba existence podpory pro klienta. Tuto roli jsme vzhledem k časové náročnosti museli zastat ve třech lidech, a to konkrétně Ing. Jiří Hunka, Bc. Oldřich Malec a já. Tato situace je dále neudržitelná a je potřeba pro budoucí provoz systému při více klientech vytvořit samostatnou pracovní pozici.

2.2 Oční Optika Benešov s. r. o.

Podle [3] *„Historie firmy Oční optik Šafanda začíná rokem 1991 resp. 15. 6. 1991, kdy její zakladatel pan František Šafanda vydražil státní provozovnu oční optiky ve Vlašimi. Následovala dražba státní provozovny v Benešově u Prahy 17. 10. 1992 a poté otevření další optiky v Sedlčanech a provozování ambulantních služeb v Týnci nad Sázavou a Voticích. Poslední změna, spíše kosmetického rázu, byla přeměna firmy v právnický subjekt s názvem OČNÍ OPTIKA BENEŠOV s.r.o. v roce 1999. Zaměstnáváme 15 pracovníků, kteří splňují nejvyšší odborná kriteria pro práci v oboru oční optiky.“*

Tato firma je klíčovým partnerem při vývoji tohoto systému. Dle potřeb a znalostí zaměstnanců byla provedena analýza. Dále pobočky této firmy poslouží jako subjekt pro nasazení systému do reálného použití a zdroj informací pro následné úpravy softwaru.

2.3 Popis problémové domény

Před samotnou analýzou a návrhem jakéhokoliv softwarového řešení je nezbytně nutné jasně definovat problémovou doménu. Problémovou doménou můžeme nazvat oblast, ve které bude systém používán, respektive systém řeší problémy z této oblasti. Informační systém nabízí uživateli prostředky k dosažení nějakého cíle. Nejlepším způsobem jak zjistit tuto doménu, je položit uživatelům otázku *„K jakému účelu vám má systém sloužit?“* Za druhé je potřeba vědět, komu tuto otázku položit.[4]

Na všechny tyto otázky jsme mohli najít odpovědi díky firmě Oční Optika Benešov s. r. o. Ve spolupráci s touto firmou jsme společně s Bc. Oldřichem Malcem a vedoucím této práce Ing. Jiřím Hunkou navštívili jednu z provozoven této firmy, kde jsme přímo od zaměstnanců zjišťovali jejich konkrétní požadavky systému a zadefinovali jednotlivé procesy.

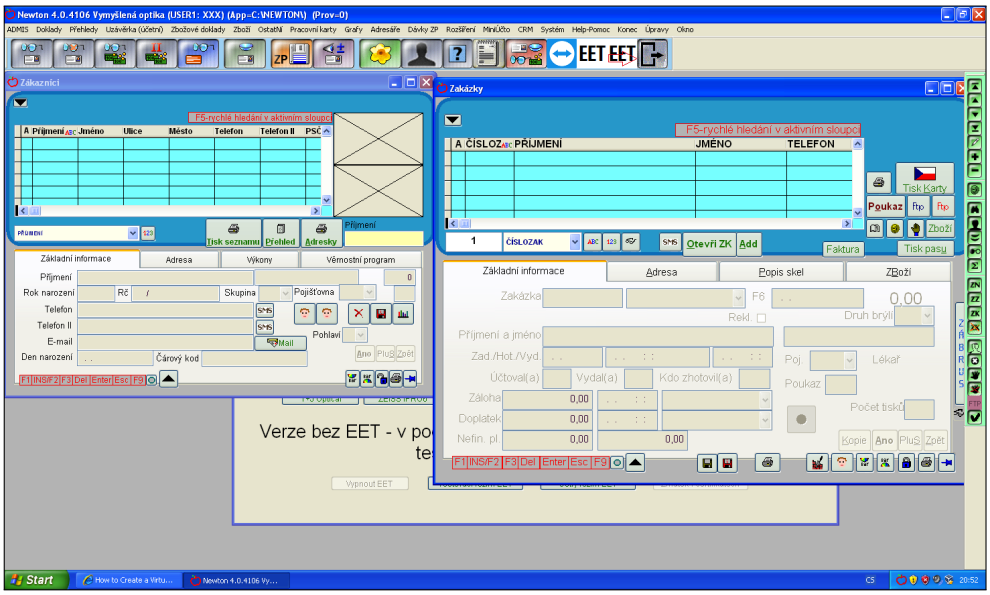
Po náročné diskuzi a následných konzultacích o skutečnostech zjištěných při komunikaci jsme byli schopni definovat problémovou doménu. Systém má tedy v kostce umožňovat prodej a příjem zboží, tvorbu a vydávání zakázky, která obsahuje jednotlivé zdravotní pomůcky pro zákazníka, a spravovat s tímto spojené skladové hospodářství. Dále je nutné myslet na to, že

systém nemá být určen pouze pro jednu prodejnu, ale pro celou organizaci, která sdružuje více prodejen.

Tento popis je velmi stručný. Další konkretizace bude probíhat v následujících kapitolách tohoto textu. Pokud bychom se snažili popsat celou problémovou doménu na jednom místě, čtenář by se mohl vzhledem k obsáhlosti projektu velmi rychle ztratit.

2.4 Aktuální řešení

Při definici funkčních požadavků systému jsme mohli vycházet ze zkušeností očních optik se stávajícími softwarovými řešeními. Konkrétně Oční Optika Benešov s.r.o. využívala program Newton[5]. Tento program sice splňuje všechny funkční požadavky, ale je velmi zastaralý a složitý. Aktuální verze tohoto programu instalovaná na pobočkách neumožňovala synchronizaci mezi jednotlivými provozovnami, což velmi komplikovalo provoz této středně velké optiky. Cílem této práce není hodnotit tento software, ale je potřeba zmínit, že tato aplikace postrádá jakýkoliv smysluplný návrh UI a orientaci na pohodlné ovládání uživatelem.



Obrázek 2.1: Aktuální systém oční optiky

2.5 Proces analýzy

Existuje několik různorodých přístupů, jak postupovat při jasném vytyčení požadavků a hranic systému. V našem případě jsme nezvolili obvyklý po-

stup, který se skládá z rešerše konkurenčních řešení, ale veškerou zodpovědnost jsme nechali na budoucích uživatelích, kteří nám posloužili jako zdroj všech potřebných informací pro návrh systému. Abychom předešli nesrovnalostem při takovémto postupu, vytvořili jsme prototyp systému, který byl opakovaně konzultován s klientem, a dle potřeby na něm byly aplikovány úpravy. Tento prototyp poté posloužil i jako návrh uživatelského rozhraní a jasně definoval všechny potřebné části systému. Bohužel musíme konstatovat, že tato slepá důvěra pouze ke klientovi jako zdroji informací nebyla šťastnou volbou více v kapitole Nasazení a provoz systému.

2.6 Funkční požadavky

Funkční požadavek na systém či software je požadavek, který systém musí být schopen provést. Jedná se o základní procesy nebo transformace, které systém z informací na vstupu převede na konkrétní výstup. Všechny tyto požadavky na systém definují kompletní chování a možnosti softwaru.[6]

Základním funkčním požadavkem na systém je mimo jiné vytvořit, připravit a vydat zakázku, která se ordinérně skládá z optických očních skel, obrub a doplňkových služeb. Takto jednoduchý proces ovšem vyžaduje mnohé velmi složité podprocesy. Nejdříve je potřeba přijmout zboží, ze kterého se zakázka skládá. Dále je potřeba připravit dokumentaci k fyzické realizaci zakázky. Po její kompletaci může být vydána, což skrývá prodej zboží v této zakázce. Tento klíčový proces systému nám přirozeně definuje základní části systému, dále jen moduly.

Systém Nadřazený modul, který využívají všechny podmoduly.

Administrace systému

Pokladna Prodej zboží a výdej zakázek

Zákazníci Správa zákazníků

Zakázky Administrace zakázek

Sklad Příjem a evidence zboží

Export Rozhraní pro komunikaci s externími systémy

2.6.1 Systém

Sdružuje obecné požadavky v podstatě jakéhokoliv informačního systému: Přihlášení a správa uživatelů, rozčlenění na jednotlivé pobočky a přiřazení zodpovědnosti a práv uživatelům, evidence terminálů na jednotlivých prodejnách a dalších doplňujících zařízeních, jako jsou tiskárny, skenery apod. Velmi specifickým požadavkem v tomto systému je rozčlenění na jednotlivé organizace

a jejich pobočky. V rámci pilotního provozu byl požadavek pouze na jednu organizaci a pět jejích poboček. Nemá ovšem smysl se při návrhu systému takto omezovat. a proto jsme jako funkční požadavek zvolili možnost spravovat neomezený počet organizací a jejich poboček.

2.6.2 Administrace systému

Každý plnohodnotný systém potřebuje svoji administraci. Rozdíly jsou jen v možnostech, které administrace nabídne. Zde požadujeme pouze základní funkcionalitu, jako je přidávání a úprava uživatelů, úprava údajů organizace či jejích poboček a zobrazení různorodých číselníků. Jejich úprava je nežádoucí, protože tyto číselníky jsou často poskytovány externími subjekty, jako jsou pojišťovny a jiné.

2.6.3 Pokladna

Tento modul slouží primárně k prodeji zboží ze skladu a výdeji zakázek. Uživatel tedy vybere položky, které chce prodat, a poté zvolí metodu platby. Platby je možné skládat z různých platebních metod, například hotovost nebo stravenka, což vynucuje informovanost uživatele, o tom jak přesně platbu složil. Po úspěšném provedení platby se vytiskne účtenka, která je předána zákazníkovi. Pokladna dále vyžaduje vést přehledy o uskutečněných transakcích ve formě pokladního přehledu a deníku. Požadavky na jednotlivé přehledy jsou velmi striktní, protože slouží jako účetní evidence a je potřeba podpora tisku těchto souhrnů. Vzhledem k prvotnímu cílení systému na český trh je legislativou České republiky vyžadována podpora systému elektronické evidence tržeb. Sekundárně tento modul slouží jako vstupní brána do ostatních částí systému.

2.6.4 Zákazníci

Tato část systému slouží jako administrace zákazníků. Jsou zde uvedeny kontaktní údaje zákazníků a jejich zakázky. Je zde vyžadována jen základní funkcionalita, kterou je přidání, úprava konkrétního zákazníka a zobrazení všech jeho zakázek.

2.6.5 Zakázky

Jedná se o nejkomplexnější část informačního systému. Obsluha zde vytváří zakázky pro jednotlivé zákazníky. Zakázka se skládá z několika částí. Základní jednotkou jsou brýle, které se skládají z optických čoček a obrub. Samotné čočky mají mnoho různých parametrů a možností, které musí být systém schopen editovat a zobrazit. Dále je možno do zakázky vložit službu nebo produkt. Specialitou pro zakázky realizované v očních optikách jsou zdravotní poukazy předepisované očními lékaři, vzhledem k tomuto požadavku systém umožňuje správu těchto poukazů, které budou blíže popsány v sekci Exporty

pro pojišťovnu. Dále je vyžadován tisk průvodní karty k zakázce. Tato karta slouží přímo zaměstnancům prodejny, kteří zakázku realizují.

2.6.6 Sklad

Prvotním požadavkem skladu je evidence produktů. Základním stavebním kamenem jsou skladové karty produktu, které obsahují detailní informace o konkrétním výrobku. Tyto produkty je možno naskladnit na určitou pobočku nebo provést přesun již naskladněných produktů na pobočku jinou. Naskladnění bývá provázeno dvěma doklady, dodacím listem a fakturou. Tyto dokumenty nemusí být vždy ihned dostupné, a je tedy potřeba umožnit párování dodacích listů, faktur a konkrétních naskladněných produktů kdykoliv nezávisle na tom jaký, dokument je již v systému vložen.

2.6.7 Export

V rámci této práce potřebujeme implementovat komunikaci s datovým rozhraním pojišťoven. Celý proces je založen na exportu dat v přesně specifikovaném formátu, který obsahuje informace o zákaznících a jim poskytnutých slevách lékařem, respektive pojišťovnou. Dále je potřeba k tomuto datovému souboru přiložit i průvodní listy v tištěné podobě. Předpokládáme, že v budoucích verzích systému bude potřeba integrace s účetními systémy a externími dodavateli zboží.

2.7 Nefunkční požadavky

Nefunkční požadavek v softwarovém inženýrství je nárokem, který nepopisuje, co systém dělá a umožňuje, ale jak to dělá. Respektive za jakých podmínek a okolností je dosaženo požadavků funkčních. Tyto požadavky jsou velmi složité na definování a testování. Většinou není tyto požadavky možné vyhodnotit objektivně. Například se jedná o výkon, škálovatelnost, bezpečnost a kvalitu softwaru.[6]

2.7.1 Výkon

Výkon se dá vyjádřit dvěma možnými postupy. Buď jako doba reakce systému na požadavek klienta, nebo počet požadavků zpracovaných za jednu sekundu. Existuje i mnoho dalších metod jak výkon změřit, ovšem pro nás je důležitá hlavně dobrá odezva systému směrem k uživateli a co nejmenší zátěž na hardware, kde systém běží.

2.7.2 Škálovatelnost

Systém je prvoplánově navržen tak, aby mohl být používán více subjekty najednou, a to jak na úrovni organizací, tak jednotlivých poboček. Je tedy nutné,

aby systém v případě potřeby umožňoval škálování, a to ideálně jak horizontálně, tak vertikálně.

Horizontální škálování Jedná se o zvyšování počtu serverů či výpočetních uzlů. Například pokud máme jeden webový server, který obsluhuje 100 klientů, přidáme druhý a zátěž rozdělíme 50 na 50.

Vertikální škálování Ve zkratce se jedná o zvyšování výkonu jednoho serveru. K tomuto škálování dochází při zvyšování kapacity RAM, zvyšování frekvence či počtu CPU. Tento výčet není kompletní a existují další způsoby jak škálovat v tomto směru.

[7]

2.7.3 Dostupnost

Uživatelé systému nutně potřebují co nejvyšší dostupnost. Tento požadavek vzniká na základě vyřizování a vydávání zakázek pro zákazníky. Pokud by došlo k výpadku systému, jednotlivé pobočky očních optik nejsou schopny provozu. Tento požadavek jsme zredukovali nutností neustálého připojení k internetu.

2.7.4 Bezpečnost

Systém obsahuje velmi citlivá data. Jedná se jak o data jednotlivých firem, které systém provozují, tak například o rodná čísla, která jsou poskytována zákazníky jednotlivých optik. Vzhledem k těmto skutečnostem je potřeba věnovat zvýšené úsilí zabezpečení celého systému.

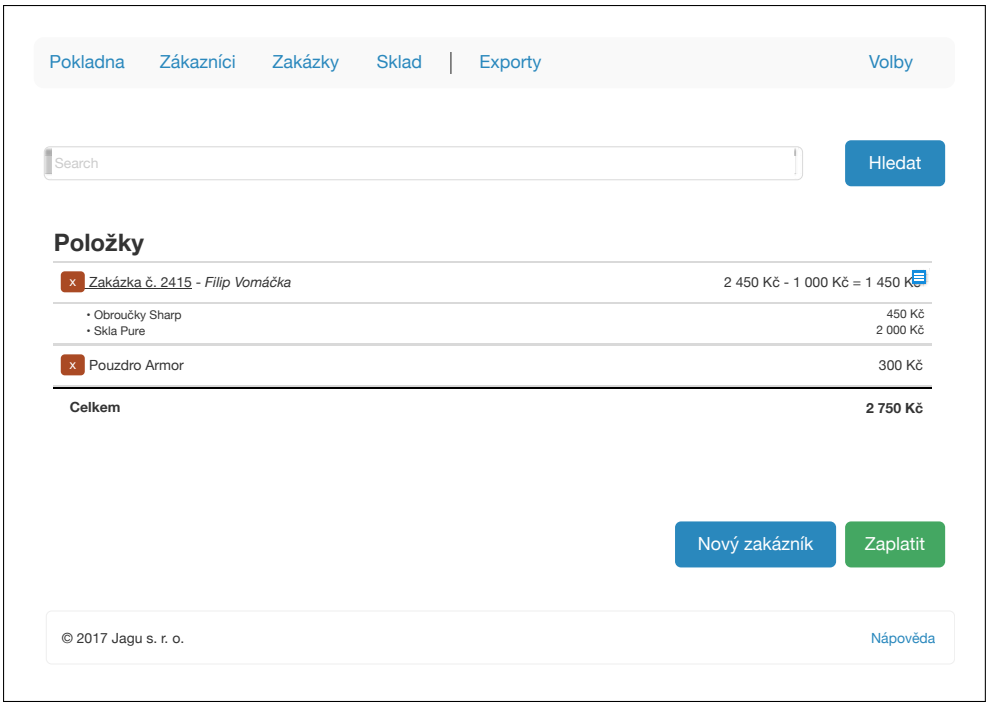
2.8 Návrh uživatelského rozhraní

Pro konkretizaci funkčních požadavků a snadnější komunikaci s budoucími uživateli systému jsme s kolegou Bc. Oldřichem Malcem vytvořili papírový model a prototyp systému. Postupovali jsme standardním způsobem. Nejdříve jsme vytvořili papírový model, který definuje jednotlivé rozložení prvků celého grafického prostředí. Následně jsme z papírových modelů vytvořili prototyp aplikace. Prototyp aplikace již umožňuje modelovat samotné procesy a zjednodušený průchod systémem. Je potřeba zdůraznit, že cílem tohoto návrhu nebylo a ani nemá být definování exaktního vzhledu jednotlivých grafických prvků. Tento postup je obecně přijímaný a věnuje se mu například přednáška[8] pana Ing. Pavla Žikovského PhD.

Existuje nepřeberné množství nástrojů na vytvoření návrhu zmíněného výše. Dokonce k vytvoření takového návrhu stačí i obyčejná tužka a papír. Tento postup je ovšem velmi neefektivní pro větší projekty. Pro modelování jsme tedy použili specializovaný nástroj Axure RP[9]. Hlavní výhodou tohoto

nástroje oproti konkurenci je snadný vývoj ve více osobách, což byl jeden z našich požadavků. Dále jsme od nástroje požadovali možnost snadného sdílení výsledného návrhu, který je dostupný jak všem kooperujícím osobám, tak klientovi. Tyto funkce námi zvolený nástroj měl, a tak bylo možné velmi rychle reagovat na požadavky a připomínky.

Po vymodelování návrhu jsme výsledek konzultovali s cílovou skupinou uživatelů. Tento postup se ukázal jako velmi užitečný, protože nám umožnil specifikovat všechny základní procesy v systému. Provedli jsme několik menších úprav a výsledek naší práce je dostupný na adrese https://agzlzg.axshare.com/. Jeho velká část je součástí této práce v rámci přílohy Prototyp aplikace. V rámci této diplomové práce jsem vytvořil tento návrh a následujícím krokům návrhu uživatelského rozhraní nebudeme zabývat, protože tomuto se věnuje paralelní diplomová práce Bc. Jaroslava Hrácha.



Obrázek 2.2: Prototyp pokladny v systému

2.8.1 Pokrytí funkčních požadavků prototypem

Všechny funkční požadavky jsme pokryli návrhem uživatelského rozhraní, abychom si mohli ověřit, že systém bude umět všechny operace, které umět má. V následující tabulce naleznete podrobný rozpis těchto požadavků a jejich pokrytí danými obrazovkami aplikace v prototypu.

Tabulka 2.1: Pokrytí funkčních požadavků

Akce	**Popis**	**Pokrytí**
Otevřít pokladnu	Prodavač nebo prodavačka musí každé ráno otevřít pokladnu a zadat kolik, mají v pokladně hotovosti. Bez otevřené pokladny není možné prodávat.	B.4
Zavřít pokladnu	Po ukončení směny se pokladna zavírá. Vyplní se, kolik je v pokladně hotovosti a kolik se z pokladny dává pryč. Po uzavření pokladny se vypíší základní informace o stavu pokladny dle proběhnutých transakcí, které slouží pro kontrolu.	B.5
Přidat položky do pokladny	Do pokladny je možno přidat několik základních druhů položek: zakázku, produkt a službu. Při přidání zakázky je ještě potřeba rozlišit, jestli zákazník platí pouze zálohu nebo doplácí celou zakázku, a tato zakázka mu je vydána. Produkt lze přidat jakýkoliv, ale při prodeji se musí zkontrolovat jeho dostupnost na skladě. Služby nejsou skladové položky, takže jdou přidat vždy.	B.2
Zaplatit obsah pokladny	Pro zaplacení obsahu pokladny je potřeba vybrat způsob platby. V tuto chvíli lze platit pouze hotově, kartou nebo stravenkou.	B.3
Zobrazit skladovou kartu	Výpis detailních informací o produktu a počet kusů skladem. Jak kusů naskladněných na pobočce, tak externě v rámci celé organizace.	B.15
Zobrazení historie naskladnění	Zobrazí výpis všech naskladnění na dané pobočce.	B.7
Přidání produktu pro naskladnění	Výběr a přidání produktu k naskladnění. Po přidání produktu se ještě navolí počet kusů produktu v tomto naskladnění.	B.9
Zobrazit historii naskladnění produktu	Kompletní zobrazení pohybu zboží, včetně prodeje či přesunu na jinou pobočku.	B.8

Tabulka 2.1: Pokrytí funkčních požadavků

Akce	**Popis**	**Pokrytí**
Spárovat naskladněné zboží s fakturou	Po naskladnění zboží je potřeba spárovat toto zboží s fakturou. Nelze to provádět hned, protože jak dodací listy, tak faktury nechodí ve stejný čas nebo můžou chodit jen na jednu pobočku z celé organizace.	B.12
Přesunout zboží na jinou pobočku	Umožňuje přesunout naskladněné zboží z jedné pobočky na druhou.	B.11
Zobrazit detail naskladnění	Zobrazí kompletní informace o naskladnění. Jaké produkty byly tímto pohybem ovlivněny a jestli je k němu nějaký dodací list.	B.15
Přidat novou kartu produktu	Přidá nový typ nebo variantu produktu do skladu.	B.14
Upravit kartu produktu	Možnost upravit již existující kartu produktu.	B.14
Zobrazit obsah skladu	Vypíše všechny skladové karty produktu a to jak produktů skladem, tak těch, které mají pouze skladovou kartu. Ve skladu je také možné vyhledávat.	B.16
Zobrazit detail zakázky	Zobrazí detail již vytvořené zakázky.	B.17
Vytvořit novou zakázku	Umožní vytvořit kompletní zakázku pro zákazníka. Zakázka obsahuje brýle, které se skládají z čoček a obrub. Dále zakázka může obsahovat produkty nebo služby. Zakázka má spoustu doplňujících atributů, které je možno vyplnit nebo se vyplňují automaticky po nějaké jiné akci.	B.18
Upravit existující zakázku	Editace již existující zakázky.	B.17

Tabulka 2.1: Pokrytí funkčních požadavků

Akce	**Popis**	**Pokrytí**
Vyplnit v zakázce poukaz pro pojišťovnu	Zákazník může dostat od lékaře předpis na brýle, který obsahuje slevové kódy na různé komponenty brýlí nebo služby. Tyto slevy jsou vyplněny u zakázky a později předloženy pojišťovně k vyplacení.	B.19
Zobrazit všechny zakázky	Výpis všech zakázek konkrétní pobočky s možností hledání.	B.20
Zobrazit zákazníky	Výpis všech zákazníků pobočky a jejich vyhledávání.	B.21
Zobrazit detail zákazníka	Detail již existujícího zákazníka.	B.22
Upravit detail zákazníka	Editace zákazníka	B.22
Přidat nového zákazníka	Formulář pro přidání nového zákazníka do systému	B.23
Přihlášení uživatele	Jednoduchý přihlašovací formulář	Nepokryto
Odhlášení uživatele	Odhlásí uživatele a zobrazí pouze přihlašovací formulář. Systém může používat pouze přihlášený uživatel.	Nepokryto
Vytisknout kartu k zakázce	Ke každé zakázce se tiskne zakázková karta, která obsahuje všechny potřebné informace pro pracovníka, který zakázku kompletuje.	Nepokryto
Zobrazit pokladní deník	Souhrnný přehled o všech transakcích, příjmech a odvodech za celý měsíc	Nepokryto
Zobrazit přehled pokladny	Podrobný přehled všech transakcí, příjmů a odvodů za vybrané období	Nepokryto
Zobrazit přehled poukazů pro pojišťovnu	Zobrazí přehled dokladů, které se musí odevzdat pojišťovně.	Nepokryto

2.9 Návrh architektury systému

„Architektura je velmi abstraktní pojem. Vždy záleží na konkrétním oboru či doméně, ve které se o architektuře bavíme.“ [10] Vzhledem ke stěžejnímu tématu této práce budeme slovem architektura myslet architekturu softwarovou. Softwarová architektura má několik různých definic.

Jedním z cílů této práce bylo navrhnout architekturu celého systému. Architektur jako takových existuje velké množství a výběr té správné vždy závisí na konkrétním typu informačního systému. Velmi důležitým krokem bylo stanovení typu systému jako takového. Tím máme na mysli rozhodnutí o tom, jestli bude systém navržen jako desktopová, chcete-li nativní aplikace, nebo bude postaven jako v poslední době stále populárnější webová aplikace. Oba tyto přístupy můžeme označit za korektní a každý má své klady a zápory. Architektury a postupy při návrhu jsou popsány v [11].

Tabulka 2.2: Srovnání desktopových a webových systémů

	Desktopová aplikace	**Webová aplikace**
Výhody	• Rychlost • Bezpečnost • Lepší přístup k HW	• Multiplatformost • Dostupnost
Nevýhody	• Distribuce • Údržba	• Závislost na prohlížeči • Provoz cloudu

Na první pohled se může z výše zmíněného přehledu zdát, že klasická desktopová aplikace bude pro tento informační systém ideální volbou. Hlavním argumentem pro tuto volbu byla potřeba tisknout účtenku na termotiskárně, což není v klasické webové aplikaci možné. Všechny ostatní požadavky splňovala webová aplikace a přinášela mnohé další výhody, jako je snadná údržba aplikace, přístup k datům odkudkoliv, kde je internetové připojení, a především vývoj pouze jedné verze systému, která bude funkční na všech běžných platformách, jako je Windows, Linux a MacOS.

Touto analýzou jsme zjistili, že ideální by byla hybridní aplikace navržená tak, aby si vzala to nejlepší z yěchto přístupů. První variantou bylo upravit si open-source prohlížeč Chromium, který je aktivně vyvíjený a stojí za ním mnoho velkých firem, jako je např. Google. Myšlenka byla taková, že se v upraveném prohlížeči otevře naše aplikace jako webová stránka a pokročilejší funkce budou přímo implementované na straně Chromia, které je klasickou nativní aplikací a umožňuje přístup přímo k operačnímu systému. Nevýhodou byla obrovská komplexnost zdrojových kódů tohoto softwaru a nikdo z nás

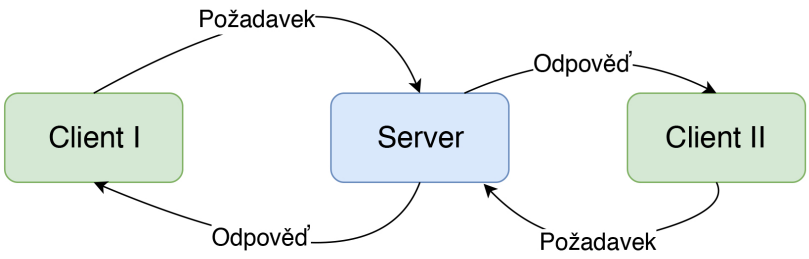
neměl se strukturou tohoto projektu žádné zkušenosti. Specifické znalosti by byly poté vyžadovány i při dalším vývoji systému a vývoj by se velmi prodražil. Vzhledem k tomu, že jinak tento přístup splňoval všechny naše požadavky, provedli jsme hlubší analýzu možností a objevili projekt Electron.[12]

Electron je framework, který umožňuje vytvářet nativní desktopové aplikace pouze za pomoci webových technologií, jakými jsou JavaScript, HTML a CSS. Tento framework podporuje všechny tři výše zmíněné operační systémy a je tedy multiplatformní. Další informace a podrobnosti o tomto frameworku jsou popsány v kapitole Realizace. Poté co tento chybějící díl skládačky zapadl, mohli jsme kompletně dokončit návrh softwarové architektury.

Systém bude rozdělen do dvou hlavních komponent, které jsme pracovně nazvali BackLynx a FrontLynx. Zatímco první zmiňovaná část se stará o samotná data aplikace a jejich integritu, FrontLynx zodpovídá za obsah zobrazený uživateli. Každý z těchto modulů je ještě interně rozdělený na jednotlivé komponenty, které jsou plně nahraditelné. Tímto návrhem byla jednoznačně rozdělena odpovědnost jednotlivých systémů bez závislosti na použitých technologiích. Jednotlivým modulům a jejich implementaci v konkrétních technologiích se budeme věnovat v následující kapitole. Z předchozího textu vyplývá, že jsme jako základní architekturu systému zvolili koncept Client–Server.

2.9.1 Client–Server

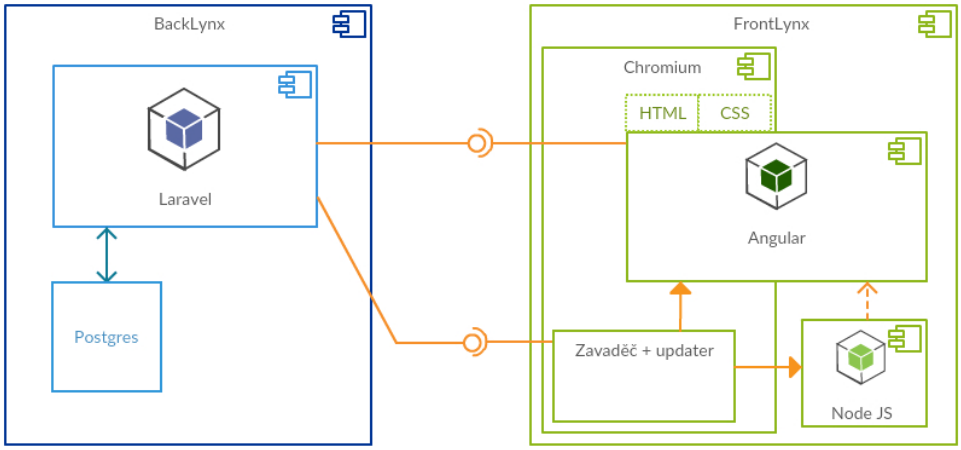
Celý ekosystém systému je postaven na konceptu Client–Server. Pojem Client–Server je dnes zmiňován hlavně v kontextu softwarové architektury. Server je část aplikace, která je společná pro všechny klienty. Serverová část tedy obsahuje bussines logiku a přístup k externím zdrojům. Klient je často nenáročná aplikace, která se připojuje k serveru.[13]



Obrázek 2.3: Client–Server

Konkrétně v našem případě je serverem BackLynx a jako klient fungují jednotlivé instance Electronu, ve kterých běží FrontLynx. Pokud se podíváme více

do hloubky, nalezneme v návrhu další využití této koncepce. Samotný BackLynx využívá například Redis, který je serverem pro kešování dat, a Laravel se k němu připojuje jako klient. Tento koncept je tedy základním stavebním kamenem našeho návrhu.



Obrázek 2.4: Backlynx - Frontlynx

2.10 BackLynx

Hlavním úkolem této části systému je správa dat a jejich distribuce ostatním částem systému. Existuje několik přístupů, jak toto realizovat. Tuto úlohu je možné separovat na dvě disjunktní funkcionality. První z nich je uložení dat do datového úložiště a druhá je výměna dat s frontendovou částí aplikace. Samozřejmě tyto části musí být schopné mezi sebou komunikovat.

2.10.1 Datové úložiště

Způsobů, jak uložit data, existuje mnoho. My potřebuje ukládat data strukturovaná a nějakým způsobem zaručit jejich integritu, proto je například primitivní ukládání do nestrukturovaných textových dokumentů nevhodné. Vznikl nám tedy první požadavek a to takový, že data musí mít pevnou strukturu. Tomuto požadavku vyhovuje několik nejběžnějších datových modelů.

Relační databáze Jedná se v dnešní době o nejvyužívanější typ databáze v informačních systémech. Data jsou reprezentována dvourozměrnými tabulkami. Jednotlivé sloupce vyjadřují konkrétní atribut, neboli prvek záznamu a řádky záznam samotný.[14]

Objektová databáze Tento typ databáze vychází z objektově orientovaného paradigmatu. V databázi jsou uloženy celé objekty i se svými atributy.

Výhodou může být adopce dalších prvků z OOP jakými jsou vícenásobná dědičnost, zapouzdření a polymorfizmus.[15]

Grafová databáze Vychází z teorie grafů a je zde kladen větší důraz na výkon než na konzistenci a svázanost struktury dat. Základní entitou je vrchol grafu a přechody neboli hrany mezi takovými vrcholy. Výhodou je přirozená reprezentace vazeb mezi daty.

Každý z výše zmíněných modelů je vhodný na jiný typ dat. Grafová databáze nebyla vhodným kandidátem, protože systém bude obsahovat pouze minimum dat, která se dají efektivně vyjádřit ve struktuře grafu. Objektová databáze by nebyla špatnou volbou, ale vzhledem k mladému věku této technologie a její neodladěnosti jsme toto řešení zavrhli. Zbyla nám tedy relační databáze, která je již velmi stabilní a léty prověřenou technologií. Její kořeny podle [16] sahají až do roku 1970.

Na druhou stranu nevýhodou relačního přístupu je zdánlivá nekompatibilita dat a dnešním obvyklým přístupem k nim. Drtivá většina dnešních programovacích jazyků je objektově orientovaná a musíme tedy mapovat objekty na relační data a naopak.

Naštěstí existuje programovací technika Object Relational Mapping, zkráceně ORM, která tuto problematiku řeší. ORM je technika, která umožňuje mapovat data z relační databáze na objekty. V ideálním případě umožňuje úplně se vyhnout SQL dotazům a přímé práci s databází. Místo toho, abychom pracovali s řádky v tabulkách, pracujeme s komplexními a často i složenými objekty. Tato technologie je využívává DBAL. Database Abstraction Layer je unikátní vrstva, která tvoří interface pro přístup do běžných DBMS. Můžeme tedy používat univerzální DBAL API a poté si vybrat podporovaný databázový stroj a nemusíme měnit zdrojový kód.[17]

Na základě těchto informací jsme jako datové úložiště zvolili relační databázi. Díky ORM se zbavíme většiny komplikací s tímto typem stroje, zachováme si možnost rychlého a efektivního prototypování. Diskuze o našem výběru konkrétní relační databáze je v sekci PostgreSQL.

2.10.2 Distribuce dat

Abychom mohli přistupovat k datům v úložišti, je potřeba vytvořit rozhraní, které toto umožňuje. Pokud by naše platforma byla klasická desktopová, nemáme moc na výběr a k datům přistupujeme přímo. V předešlé kapitole jsme ovšem zvolili přístup webový a zde se nám naskýtají dva různorodé koncepty, SOAP A REST.

SOAP Simple Object Access Protocol byl vytvořen v roce 1998 Davidem Winerem ve spolupráci s firmou Microsoft. Vývoj pod takto obrovskou softwarovou společností předurčovalo využití tohoto konceptu v enterprise prostředí. Můžeme říct, že celý tento koncept je založen na techno-

logii XML. Server a klient si mezi sebou vyměňují zprávy, které jsou ve formátu XML. Tyto zprávy mají definované schéma a lze je tedy velmi snadno strojově validovat.[18]

REST Representational State Transfer byl vytvořen Royem Fieldingem na Kalifornské univerzitě v Irvine. Z důvodu vzniku tohoto konceptu na akademické půdě tento protokol respektuje filosofii otevřeného webu. Nejedná se o konkrétní technologii či implementaci, ale o koncept. Jedná se o stateless komunikaci, to znamená, že server neudržuje žádnou informaci o klientovi a není schopen mezi klienty rozlišit. Základem RESTu je zdroj, který je jednoznačně identifikován. Každý zdroj má poté vlastní reprezentaci, která může být například HTML dokument nebo JPEG obrázek. [19]

Jako RESTful službu označujeme takovou, která implementuje REST architekturu. Základem jsou tedy zdroje, pod těmito zdroji si můžeme představit v podstatě jakoukoliv informaci, kterou dokážeme pojmenovat. Operace s těmito zdroji jsou definované pouze několika málo metodami přímo z protokolu HTTP, jako je GET, POST, PUT a další. Žádné jiné akce není možné přidat.[20]

Může se zdát, že RESTful služby nabízejí velmi omezené výrazové schopnosti, ale právě díky své jednoduchosti nabízejí téměř nekonečné možnosti a velmi jednoduchou implementaci. Podrobněji se této problematice budeme věnovat dále v této kapitole.

Tabulka 2.3: Srovnání SOAP vs. REST [21]

	SOAP	**REST**
Koncept:	Data jsou dostupná jako služba a je k nim dokonce i takto přistupováno. Například getAddress nebo IssueOrder.	Data jsou dostupná jako zdroje tzv. resources. Například Address nebo Order
Klady:	• Formální enterprise přístup • Není vázáno na žádný konkrétní komunikační protokol • Klient má kompletní informace o objektech • Řešení bezpečnosti a autorizace jsou přímo součásti protokolu • Formální specifikace a dokumentace pomocí WSDL	• Otevřený přístup • Velmi jednoduché na implementaci, údržbu a rozšiřování • Oddělená implementace klienta a serveru • Snadné cachování na straně klienta • Nezávislost na formátu datové komunikace
Zápory:	• Velká datová režie při komunikaci • Složitější implementace • Velmi robustní a svázané	• Podpora pouze jediného protokolu, a to HTTP • Není vázáno na žádný konkrétní komunikační protokol • Klient má kompletní informace o objektech
Vhodné pro:	• Pokud vzniká potřeba přímého přístupu přímo k objektům na serveru • Složitou formální a transakční komunikaci mezi klientem a serverem	• Pokud klient i server pracují ve webovém prostředí • Nejsou potřeba žádné konkrétní přístupy k objektům nebo vzdáleným metodám
Nevhodné pro:	• Požadujeme jednoduché použití a rozvoj API • Omezenou šířku pásma komunikace	• Volání procesů, které se skládají z vícenásobného počtu volání

Na základě těchto informací jsme vybrali konkrétní přístup. Na první pohled se může zdát, že SOAP přístup bude to pravé kvůli složitosti systému, ale vzhledem k agilnímu vývoji aplikace bude lepší jednodušší a transparentnější REST. Chybějící dokumentaci vyřešíme specializovanými nástroji popsanými níže.

2.10.3 Návrh

Pro návrh jsme nezvolili standardní postup, který se skládá z konceptuálního a následně datového modelu relační databáze, ale primárně jsme navrhli struktury jednotlivých entit pomocí dokumentace API. Toto řešení jsme zvolili hlavně z toho důvodu, že bylo potřeba záhy po implementaci základu backendu paralelně vyvíjet i frontend. Datový model by byl pro vývoj uživatelské části systému nedostačující. Zatímco při dodržení tohoto postupu vývojář frontendu ihned věděl, jaká data dostane po zavolání konkrétního zdroje a jakou množinu dostupných operací systém podporuje. Dalším argumentem pro založení dokumentace na samotné dokumentaci API byla neexistence standardu pro dokumentaci RESTful webové služby, na rozdíl od již zmíněného SOAPu, který má WSDL.

Pro dokumentaci jsem zvolili OpenAPI format verze 2.0 [22]. V době návrhu již byla kompletní specifikace verze 3.0, ale tu jsme nepoužili z důvodu malé podpory externích nástrojů pro práci s tímto formátem, jako je například exportování do jiných formátů. Jako nástroj pro tvorbu jsme vybrali Swagger.[23]

Tento nástroj umožňuje realizaci celého vývojového cyklu API od návrhu a dokumentace, až po testování a nasazení. My jsme pro tento projekt využili jen polovinu možností této platformy a testovaní společně s nasazením jsme si řešili sami. Celý návrh je realizován pomocí velmi jednoduchého textového formátu YAML. Příklad definice zdroje adresy z návrhu:

Zdrojový kód 2.1: Definice zdroje adresy

```
/address:
    post:
      tags:
      - address
      summary: Add a new address to the system
      operationId: addAddress
      consumes:
      - application/json
      produces:
      - application/json
      parameters:
      - in: body
        name: body
        description: Address object
        required: true
        schema:
          $ref: '#/definitions/Address'
      responses:
        405:
          description: Invalid input
```

Z takto jednoduchého zápisu je hned jasná URI zdroje */address*, HTTP metoda *POST*, že server přijímá i vrací data ve formátu JSON a díky *#/definitions/Address* jasně specifikujeme, jak tato data vypadají. Toto jsou jen základní možnosti, které nám formát OpenAPI nabízí. Můžeme definovat i oprávnění pro jednotlivé metody a cesty ke zdrojům. Tímto způsobem lze tedy vytvořit kompletní dokumentaci. Platforma také umožňuje generování základní struktury zdrojového kódu pro různé programovací jazyky a jejich frameworky. Této možnosti jsme využili a byli jsme tedy velmi rychle schopni vytvořit základ API, podle kterého mohla být vyvíjena i frontendová část systému.

Možnost definovat schéma entit bylo hlavním argumentem pro volbu této dokumentace. Umožnilo nám to vynechat klasický konceptuální model při návrhu a implementaci relační databáze. Definice schématu takovéto entity vypadá následovně:

Zdrojový kód 2.2: Dokumentace entity

```
Address:
    type: object
    required:
    - id
    - street
    - zip
    properties:
      id:
        type: integer
        format: int64
      street:
        type: string
      houseNumber:
        type: string
      zip:
        type: integer
        format: int32
      city:
        type: string
      country:
        type: string
```

2.11 FrontLynx

FrontLynx, jak již název napovídá, plní pouze úlohu frontendu aplikace. Zodpovídá za prezentaci informací získaných z BackLynxu uživateli. Díky návrhu architektury systému je tento modul kdykoliv nahraditelný za implementaci v jakékoliv technologii, která umí komunikovat pomocí protokolu HTTP. Vývojem této části systému se zabývá diplomová práce mého kolegy Bc. Jaroslava Hrácha.

V rámci mé práce jsem navrhl pouze použité technologie pro stávající realizaci. Při takovémto návrhu je velmi důležité specifikovat, která část validuje uživatelem zadaná data. Na základě diskuze s kolegou jsme se rozhodli, že za validaci těchto dat zodpovídá FrontLynx a backend je pouze zkontroluje a ve výjimečných případech se je pokusí opravit. Pokud se to nepovede, vrátí stavový kód, který signalizuje chybu vstupních dat.

Takto navržený frontend má obrovskou výhodu v tom, že veškerá zátěž a režie spojená s vykreslováním obsahu je přenesena na zařízení kde FrontLynx běží a nezatěžuje naše servery, které se starají pouze o data.

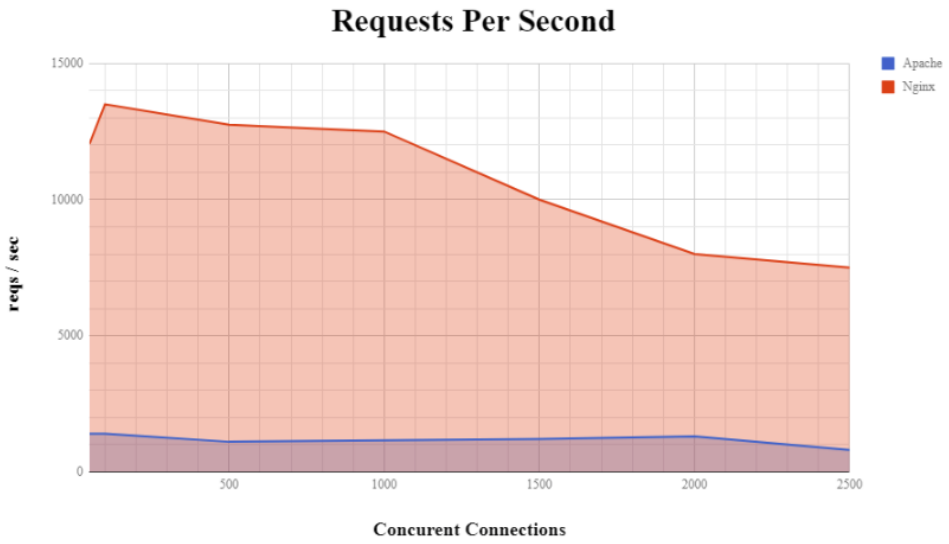
2.12 Použité technologie

V této sekci se budeme věnovat použitým technologiím. Tyto technologie jsme volili na základě mých zkušeností a vhodnosti pro tento typ projektu. U některých stěžejních technologií rozepíšeme důvody, proč jsme zvolili zrovna tyto, a představíme jejich konkurenty.

2.12.1 NGINX

NGINX je především HTTP a reverzní proxy server. Nabízí i funkčnost proxy serveru a generického TCP/UDP proxy serveru. Pro náš projekt ho využijeme převážně jako klasický HTTP server. Tento program byl původně napsán Igorem Sysoevem a je dále komunitně rozvíjen pod 2bodovou BSD licencí. V březnu 2018 obsluhoval 23,76 % nejvytíženějších webových služeb na světě. Těší se velké oblibě i mezi velkými softwarovými projekty a službami, jako je Dropbox, Netflix, Wordpress.com a mnohé další.[24]

Jeho největším konkurentem je server Apache. Tento server jsme také brali v úvahu, ale vzhledem k lepší výkonnosti při více připojeních, která rozhodně vychází lépe pro NGINX, jsme zvolili tento server. Počet požadavků za sekundu při vysokém počtu paralelních připojení je pro nás zásadní a to z toho důvodu, že plánujeme do budoucna rozšířit počet našich klientů.



Obrázek 2.5: NGINX versus Apache [25]

2.12.2 PHP

Tento programovací jazyk jsem zvolil jako hlavní už ve své bakalářské práci [26], podle které je jazyk PHP *„Následník Perlu navržený pro webové skriptování na straně serveru. Skripty jsou typicky vloženy ve webových stránkách. Byl vyvinut Rasmusem Lerdorfem v roce 1995 za účelem správy jeho osobní domovské internetové stránky. Nyní je jako oficiální název používán rekurzivní akronym (PHP: Hypertext Preprocessor). Novější verze vyvíjí Andi Gutmans a Zeev Suraski ve spolupráci s Lerdorfem. PHP obsahuje vestavěnou podporu pro široké spektrum internetových protokolů a pro přístup do více než tuctu komerčních databázových řešení. Verze 5 z roku 2004 přidala podporu pro objektově orientované programování, jako je složená dědičnost, iterátory, automatické načítání, strukturované zpracování výjimek, reflexi, přetěžování metod a volitelnou deklaraci typu parametru.“*

Můžeme říct, že tento programovací jazyk i s technologiemi okolo postupuje mílovými kroky a adoptuje mnohé moderní přístupy z konkurenčních jazyků. Navíc je pro něj dostupných mnoho vynikajících frameworků, které rozšiřují možnosti tohoto jazyka a velmi urychlují vývoj jakékoliv webové aplikace. Jedním z těchto frameworků je i Laravel.

2.12.3 Laravel

Laravel je velmi populární PHP framework. Vybírali jsme z několika frameworků, jako je třeba Nette či Symfony, ze kterého Laravel dokonce vychází. Hlavním argumentem pro volbu tohoto řešení byla jeho skvělá podpora RESTful architektury, která je základem pro modul BackLynx. Laravel podporuje, až přímo vybízí k psaní jednoduchého a srozumitelného kódu, který je snadno udržitelný. [27]

Mezi jeho velmi dobře implementované vlastnosti patří například ORM modul, kterému se více věnujeme v kapitole ORM. Analýzou jsme zjistili, že většina potřebných funkcionalit pro BackLynx je přímo obsažena v základní instalaci tohoto frameworku.[28]

2.12.4 PostgreSQL

V sekci Datové úložiště jsme jako základní datové úložiště zvolili relační databázi. Samozřejmě jsme měli na výběr z několika nejrozšířenějších databázových strojů, jakými jsou například MySQL, MariaDB, Oracle nebo PostgreSQL. První dva zmiňované jsme zavrhli z důvodu jejich jednoduchosti a omezeným možnostem. Sám tyto databáze používám především na menší projekty. Jasnou volbou by byl Oracle, bohužel ten je placený a vzhledem k rozpočtu tohoto projektu je zakoupení licence nemožné. Zbývá tedy PostgreSQL.

PostgreSQL je open-source databázový systém a jeho vývoj probíhá více než 30 let. Implementace pokrývá skoro celou specifikaci SQL:2011, takže jeho možnosti jsou opravdu rozsáhlé. K jeho hlavním vlastnostem patří:

Datové typy Obrovský rozsah datových typů od primitivního integeru až po dokumentový XML s možností definovat vlastní datové typy.

Integrita dat Obsahuje všechny základní implementace technik pro zachování datové integrity, jakými jsou primární klíče, cizí klíče či explicitní zámky.

Procedury Používá vlastní jazyk pro psaní procedur PL/PGSQL, ale umožňuje psaní i v běžných skriptovacích jazycích, jako je Perl nebo Python.

[29]

2.12.5 Angular

Jedná se o JavaScriptový framework, který umožňuje tvořit velmi komplexní single page aplikace. Single page aplikace je taková webová aplikace, která na změnu nereaguje přenačtením celé stránky, ale upraví pouze ty části obsahu, které byly akcí ovlivněny. Angular má dvě zásadní vlastnosti, kvůli kterým jsme se rozhodli pro jeho využití v našem projektu. Je naprogramovaný v TypeScriptu a ne v čistém JavaScriptu. O tomto jazyku více v následující sekci. Druhou velmi důležitou vlastností je adopce MVC architektury. Ta odděluje logiku aplikace napsanou v TypeScriptu od vzhledu aplikace, který je definován v HTML šabloně.

2.12.6 TypeScript

TypeScript je skriptovací programovací jazyk, který vytvořila firma Microsoft. Jeho syntaxe je velmi podobná JavaScriptu, ovšem přidává do tohoto jazyka standardní prvky z OOP paradigmatu, jako jsou třídy, rozhraní a typová kontrola. Kompatibilita je zajištěna tím, že kód napsaný v TypyScriptu se překompiluje do klasického JavaScriptu.[30]

Díky těmto rozšířením se snadněji udržuje a provádí přepracování obsáhlého zdrojového kódu. Do tohoto jazyka je možné vkládat i čistě JavaScriptové zdrojové kódy, takže nevzniká žádné omezení s kompatibilitou a použití tohoto jazyka nám přináší jen výhody.

2.12.7 Electron

Electron je open-source knihovna vyvinutá společností GitHub. Jejím hlavním účelem je vývoj multiplatformních desktopových aplikací za pomoci pouze webových technologií, jakými jsou HTML, CSS a JavaScript. Této funkcionality je dosaženo díky kombinaci dvou stěžejních technologií Chromium a Node.js.[31]

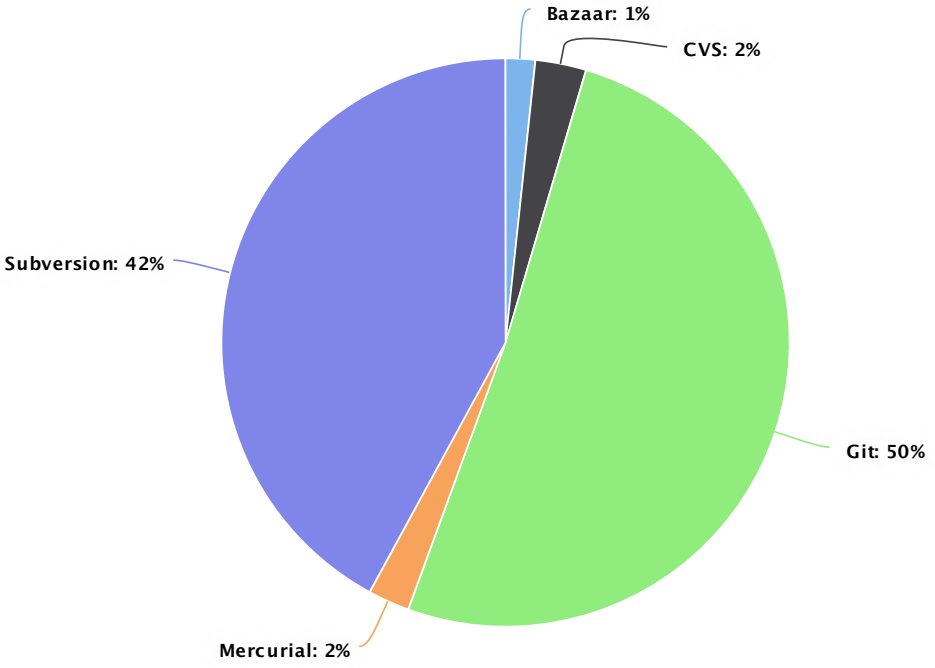
Chromium Chromium je kompletní otevřený webový prohlížeč.[32] Jak jsme již zmínili výše, úprava celého Chromia je velmi náročná. Proto je pro

nás výhodou použít Electron, který si z celého projektu bere pouze vykreslovací jádro. Díky tomu může naše aplikace velmi rychle reagovat na moderní trendy, protože toto jádro je vyvíjeno mnoha vývojáři z celého světa.

Node.js Jedná se o běhové prostředí pro JavaScript. Obsahuje i vlastní balíčkovací systém npm, takže je velmi jednoduché rozšířit jeho možnosti pomocí externích knihoven. Toto prostředí je asynchronní a řízené událostmi a to je dnes velmi moderní a efektivní způsob implementace.[33]

2.12.8 Gitlab

K vývoji většího softwaru navíc v mnohačlenném vývojářském týmu se hodí verzovací systém. Verzovací systém eviduje všechny změny ve zdrojových kódech systému. Skupiny těchto změn či jejich jednotky můžeme spojovat a vytvářet tak různé verze softwaru. Tyto verze nazýváme branch a jednotlivé změny commity. V předchozích řádcích jsem popsal distribuovaný typ verzování. Tento typ se v dnešních dnech stává používanějším než centralizované systémy, jakým je například Subversion. Jedním z tohoto typu systému je Git a tento systém využívá aplikace Gitlab.



Obrázek 2.6: Subversion versus Git [34]

Gitlab je komplexní webová aplikace, která kromě serveru pro verzování zdrojového kódu nabízí i mnoho dalšího. Obsahuje tiketovací systém na řešení chyb aplikace, obsahuje continuous integration řešení pro vydávání a sestavování softwaru, které také využíváme v našem projektu, a mnohé další. Gitlab jsme zvolili hlavně kvůli tomu, že většina členů našeho týmu s ním má dlouholeté zkušenosti.

Kapitola 3

Realizace

V rámci celé této kapitoly budeme hojně uvádět úryvky zdrojových kódů systému a to z toho důvodu, že v rámci této diplomové práce nebudou zdrojové kódy zveřejněny.

3.1 Vývojové prostředí

Jednotné vývojové prostředí zjednodušuje adaptaci a začátek vývoje nově příchozím členům vývojářského týmu. Zároveň také sjednocuje prostředí pro běh aplikace, které se dá později přenést i na produkční prostředí. K realizaci vývojového prostřední jsme použili Vagrant obraz. Jedná se o kompletní obraz virtuálního stroje, který je spuštěn na hostitelském počítači vývojáře. Tento obraz jsme vytvořili společně s Bc. Pavlem Kovářem. Já jsem navrhl jeho podobu, specifikoval požadavky a kolega Kovář ji realizoval dle mých představ.

Na tomto obrazu se nachází jak BackLynx, tak FrontLynx. Pro běh BackLynxu je nainstalován NGINX, PHP, PostgresSQL a další podpůrný software, jako je balíčkovací systém Composer. FrontLynx využívá nainstalovaný NPM balíčkovací systém a Node.js. Po stažení tohoto balíčku může vývojář okamžitě začít vyvíjet a pomocí tohoto obrazu i aplikace spustit.

3.2 Datový model

Samotnou realizaci jsme začali implementací struktury databáze. Jako návrh pro tuto implementaci posloužila dokumentace entit z API. Vzhledem k tomu, že v podstatě od začátku implementace backendu bylo potřeba realizovat i frontend, musel být vývoj velmi agilní. K tomu nám posloužily metody a jejich implementace z následujících kapitol.

3.2.1 Migrace

Nevýhodou použití relačních databází u velkých projektů je jejich svázanost se schématem. Tato nevýhoda se projeví při vývoji v týmu, kdy všichni členové týmu musí být informování o konkrétní změně v určité entitě pro danou verzi zdrojového kódu. Na těchto změnách se musí domlouvat, a dokonce minimální změna například v povolené délce textového řetězce může zapříčinit nekompatibilitu vašeho zdrojové kódu.

Tento problém velmi elegantně řeší migrace. Každá jednotlivá migrace je třída, která obsahuje dvě metody `up()` a `down()`. První metoda obsahuje definice změn v databázi. Druhá zmiňovaná metoda obsahuje tyto změny v opačném pořadí, takže pokud nejdříve zavoláme `up()` a následně `down()`, v databázi neproběhnou žádné změny.

Tento přístup nám umožňuje zároveň s aktualizací zdrojového kódu aplikace vydat i novou migraci, která vytvoří či jen upraví existující tabulku v databázi. Všichni vývojáři tedy přesně vědí, kdy a proč se databáze změnila. Zároveň si migrace samy hlídají integritu dat a nedovolí vám při úpravách jakkoliv porušit integritní omezení definované současnou strukturou databáze. Příklad migrace, která vytvoří tabulku položek na účtence:

Zdrojový kód 3.1: Migrace pro vytvoření položky na účtence

```
1  public function up() {
2   Schema::create('receipt_items', function (Blueprint $table) {
3     $table->increments('id');
4     $table->integer('receipt_id');
5     $table->integer('order_item_id');
6     $table->timestamps();
7   });
8 }
9
10 public function down() {
11   Schema::dropIfExists('receipt_items');
12 }
```

Tyto migrace jsme používali od začátku vývoje, takže můžeme zpětně identifikovat, kdy došlo ke změnám a z jakých důvodů. Migrace jsou velkým přínosem i při úpravách na produkčních datech. K datu 1. 5. 2018 se pro implementaci a následné úpravy API vytvořilo 114 migrací. V tuto chvíli je tedy vhodné označit danou verzi za stabilní, vytvořit jednu migraci, která vytvoří databázi v současné struktuře, a dále držet jen nové migrace.

3.2.2 Seeder

Mít dobře navržené a implementované datové struktury je nutnou podmínkou pro testování a další vývoj softwaru. Naneštěstí pokud v těchto strukturách

nejsou data, lze je jen velmi obtížně testovat. Vytvoření takovéto testovací sady dat není triviální úkol a každá změna struktury vyžaduje změny i v těchto datech. Abychom se tomuto vyhnuli, implementovali jsme takzvaný Seeding systém.

Implementace seederu spočívá ve vytvoření jednoduché třídy, jejíž obsahem je definice objektu, který se má vytvořit. Konkrétní atributy objektu jsou vybírané náhodně. Pokud by data byla úplně náhodná, nebylo by při testování zřejmé, co reprezentují. Z tohoto důvodu jsme použili existující implementaci knihovny[35], která vrací náhodná data se specifickými parametry. Jedná se například o ulice, města, barvy, jména, e-maily a mnoho dalšího. Jednoduchá implementace seederu, který vloží náhodně vytvořenou úpravu skel, vypadá takto:

Zdrojový kód 3.2: Seeder pro vytvoření náhodných dat

```
1 $code = $faker->numberBetween(100, 999);
2 $code = $faker->numberBetween(100, 999);
3 $name = $faker->colorName;
4 GlassesFinish::create([
5   'code' => $code,
6   'name' => $name,
7   'price' => $price,
8   'vat' => 0.21,
9 ]);
```

Díky tomu, že jsme implementovali podobné třídy pro všechny základní typy entit, byli jsme ihned od začátku vývoje schopni testovat a daleko lépe kooperovat požadavky frontendu na backend a obráceně. Tento postup jsme využili pouze při vývoji, na produkčních datech se již tyto seedery samozřejmě nepoužívají.

3.2.3 ORM

Všechny postupy popsané v předchozích kapitolách jsme mohli elegantně použít díky ORM. Koncepci Object Relational Mapping jsme popsali v části Datové úložiště. Nyní se podíváme na konkrétní implementaci v našem systému. Pro implementaci jsme zvolili Eloquent, který je přímo součástí frameworku Laravel. Pomocí této knihovny jsme zadefinovali všechny entity v naší aplikaci. Implementace entit obsahuje i specifikaci vazeb mezi ostatními objekty. Jsou podporovány všechny standardní relační vazby a operace s nimi. Za standardní typy vazeb považujeme 1:1, 1:N a M:N. Jednoduchá definice entity reprezentující čárový kód vypadá takto:

Zdrojový kód 3.3: ORM definice čárového kódu

```
class Ean extends Model {
  protected $fillable = [
    'number',
    'product_id',
    'deleted'
  ];
  protected $hidden = [
    'created_at',
    'updated_at'
  ];
  public function product() {
    return $this->belongsTo('App\Product');
  }
}
```

Pomocí pole `$fillable` definujeme, které atributy umožníme nastavovat či editovat. V poli `$hidden` se nachází definice atributů objektu, které nechceme vypisovat například při převodu objektu na JSON nebo řetězec. Pomocí funkce `product()` říkame, že čárové kódy mají relaci N:1 k entitě produktu. Díky této funkci můžeme poté zavolat `$ean->product()` a dostaneme objekt produktu, ke kterému se váže tento čárový kód.

Použití této knihovny je velmi jednoduché, ale přitom umožňuje většinu operací s databází. Mohu konstatovat, že v celém zdrojovém kódu BackLynxu se nenachází jediný dotaz do databáze napsaný v SQL jazyku a všechny databázové operace jsou řešeny přes Eloquent ORM.

3.3 API

Pomocí implementace dílčích částí popsaných výše jsme mohli realizovat hlavní předmět této práce, a to RESTové API. Nejdříve jsme realizovali hlavní části API, které pokrývaly funkční požadavky na systém, a poté jsme prováděli úpravy či přidávali nové zdroje dle požadavků vývojáře FrontLynxu. Tato kapitola je rozdělena na tři základní části. První část popisuje, jakým způsobem a pomocí jakého formátu komunikuje FrontLynx s BackLynxem. Ve druhé části popíšeme jednotlivé zdroje API a jejich metody a v poslední části se budeme zabývat implementací tisku.

3.3.1 JSON

Komunikace mezi hlavními moduly systému probíhá pouze ve formátu JSON. JavaScript Object Notation je textový formát, který slouží k výměně dat mezi různými platformami. Důležitou vlastností tohoto formátu je jeho čitelnost.

Obsah tohoto dokumentu je dobře strojově čitelný a zároveň je čitelný i pro člověka. Díky tomu můžeme přímo analyzovat komunikaci a mít jasnou představu o tom, jaká data si platformy posílají. Syntaxe tohoto formátu je založena na párech klíče a hodnoty. Například `"name":"Filip"` kde `"name"` je klíč a `"Filip"` je hodnota náležící k tomuto klíčí. Dvojtečka slouží jako oddělovač klíče a hodnoty. JSON má pouze pár speciálních znaků.[36]

Levá složená závorka { Začátek objektu

Pravá složená závorka } Konec objektu

Levá hranatá závorka [Začátek pole

Pravá hranatá závorka] Konec pole

Dvojtečka : Odděluje klíč a hodnotu

Čárka , Odděluje jednotlivé páry nebo prvky v poli

S takovouto základní syntaxí, která je ovšem kompletní, můžeme konstruovat složité objekty, které obsahují pole prvků, nebo dokonce objektů. JSON v praxi může vypadat například takto:

Zdrojový kód 3.4: Ukázka formátu JSON

```
[
    {
        "id": 2,
        "name": "Filip Glazar",
        "address_id": 6,
        "pin": "9207306141",
        "phoneNumber": "603702455",
        "email": null,
        "vip": null,
        "discount": "0",
        "insurance_company_id": "1",
        "send_sms": null,
        "address": {
            "id": 6,
            "street": "Pricni",
            "houseNumber": "352",
            "zip": "56169",
            "city": "Kraliky",
            "country": null,
            "phone": null
        },
        "insurance": {
            "id": 1,
            "code": 111,
            "shortcut": "VZP",
            "name": "Vseobecna zdravotni pojistovna"
        }
    },
    ...
]
```

Jedná se o JSON přímo z implementace, který reprezentuje pole objektů zákazník a tento objekt má v sobě další dva objekty, jako je adresa a zdravotní pojišťovna. Komunikace fontendu a backendu probíhá výhradně tímto formátem. Z příkladu je vidět, že backend vrátil přímo složenou entitu zákazník. Toto však není vhodné vždy a někdy nepotřebujeme posílat entitu celou nebo potřebujeme zaslat taková data, která jsou kombinací těchto entit, nebo jakákoliv jiná agregovaná data. Na to slouží takzvané resources, které jsme v rámci API implementovali.

Resources Tento koncept je přímo připraven na situace, kdy potřebujeme předat model nebo jeho část jako JSON. Umožňuje jednoduše transformovat data a nemusíme kvůli tomu upravovat logickou definici modelu. Dokonce můžeme spojit data z více modelů a vrátit jako JSON. Pokud vracíme jednu konkrétní transformaci, mluvíme o zdroji, pokud vracíme pole těchto transformovaných entit, mluvíme o kolekci.

3.3.2 Specifikace

V této sekci představíme metody a zdroje, které podporuje API BackLynx. Nejedná se o úplně kompletní výčet všech metod. Kompletní graficky i textově vyjádřená doukemtace se nachází na https://app.swaggerhub.com/apis/FigaCZ/OptiLynx/1.0.0

Tabulka 3.1: Specifikace API - Lekaři

URI	Metoda	Popis
`doctors`	**GET**	Seznam všech lékařů
`~/refresh`	**GET**	Aktualizace lékařů z externího zdroje
`~/search`	**GET**	Hledání v seznamu lékařů
`~/{icz}`	**GET**	Vrátí lékaře dle zadaného IČZ

Tabulka 3.2: Specifikace API - Autentifikace

URI	Metoda	Popis
`login`	**POST**	Přihlášení uživatele
`logout`	**POST**	Odhlášení uživatele
`register`	**POST**	Registrace nového uživatele

Tabulka 3.3: Specifikace API - Sklad

URI	**Metoda**	**Popis**
Prefix: `stock/`		
`in`	**POST**	Naskladnění zboží na pobočku
`out`	**POST**	Vyskladnění zboží z pobočky
`move`	**POST**	Přesun zboží na jinou pobočku
`{branch}/orders`	**GET**	Pohyb zboží související s objednávkami
`{branch}`	**GET**	Pohyby zboží na dané pobočce
`history/in/{branch}`	**GET**	Historie naskladnění na pobočku
`history/move/{organization}`	**GET**	Historie přesunu mezi pobočkami

Tabulka 3.4: Specifikace API - Stav objednávky

URI	**Metoda**	**Popis**
Prefix: `orders/{order}/`		
`state`	**POST**	Změna stavu objednávky
`card`	**GET**	Tisk zakázkové karty

Tabulka 3.5: Specifikace API - Prodej

URI	Metoda	Popis
Prefix: `cashdesk/`		
`sell`	**POST**	Prodej zboží na pokladně
`resetEet`	**POST**	Odeslání záporné tržby na EET
`transactionDelete/{transaction}`	**GET**	Odstranění platby
`restock/{order}`	**GET**	Opětovné naskladnění již vydané zakázky
`oppositeReceipt/{receipt}`	**GET**	Záporná účtenka

Tabulka 3.6: Specifikace API - Adresy

URI	Metoda	Popis
`addresses`	**GET**	Seznam všech adres
`addresses`	**POST**	Vložení nové adresy
`~/{address}`	**GET**	Vrátí adresu
`~/{address}`	**PUT**	Upraví adresu
`~/{address}`	**DELETE**	Odstraní adresu

Tabulka 3.7: Specifikace API - Účtenky

URI	Metoda	Popis
`receipts`	**GET**	Všechny účtenky
`receipts`	**POST**	Vložení nové účtenky
`~/{receipt}/print`	**GET**	Tisk účtenky
`~/{receipt}`	**GET**	Vrátí účtenku
`~/{receipt}`	**DELETE**	Odstraní účtenku

Tabulka 3.8: Specifikace API - Zdravotní pojišťovny

URI	Metoda	Popis
`insuranceCompanies`	**GET**	Seznam zdravotních pojišťoven
`~/{insuranceCompany}`	**GET**	Detaily o zdravotní pojišťovně
`~/{insuranceCompany}`	**PUT**	Upraví údaje zdravotní pojišťovny
`~/{insuranceCompany}`	**DELETE**	Odstraní zdravotní pojišťovnu

Tabulka 3.9: Specifikace API - Dodavatele

URI	Metoda	Popis
`suppliers`	**GET**	Vypíše všechny dodavatele
`suppliers`	**POST**	Přidá dodavatele
`~/{supplier}`	**GET**	Vrátí konkrétního dodavatele
`~/{supplier}`	**PUT**	Upraví detaily o dodavateli
`~/{supplier}`	**DELETE**	Odstraní dodavatele

Tabulka 3.10: Specifikace API - Služby

URI	Metoda	Popis
`services`	**GET**	Všechny služby
`services`	**POST**	Přidá službu
`~/{service}`	**GET**	Vrátí konkrétní službu
`~/{service}`	**PUT**	Upraví službu
`~/{service}`	**DELETE**	Odstraní službu

Tabulka 3.11: Specifikace API - Pobočky

URI	Metoda	Popis
`branches`	**GET**	Všechny pobočky
`branches`	**POST**	Přidá pobočku do systému
`~/{branch}`	**GET**	Konkrétní pobočka
`~/{branch}`	**PUT**	Úprava konkrétní pobočky
`~/{branch}`	**DELETE**	Odstraní pobočku
`~/{branch}/orders`	**GET**	Zakázky na dané pobočce
`~/{branch}/orders`	**POST**	Přidá zakázku na konkrétní pobočku
`~/{branch}/receipts`	**GET**	Účtenky z jedné pobočky

Tabulka 3.12: Specifikace API - Pokladna a účetnictví

URI	Metoda	Popis
`terminals`	**GET**	Všechny pokladny
`terminals`	**POST**	Vloží pokladnu
`~/terminal`	**GET**	Konkrétní pokladna
`~/terminal`	**PUT**	Úprava pokladny
`~/terminal`	**DELETE**	Odstranění pokladny
`~/terminal/open`	**POST**	Otevře pokladnu
`~/terminal/close`	**POST**	Zavře pokladnu
`~/terminal/closeOverview`	**GET**	Přehled pokladny
`~/terminal/expense`	**POST**	Vloží výdaj do pokladny
`~/terminal/history`	**GET**	Historie pokladny
Prefix: `{terminal}/{date}/`		
`~/register`	**GET**	Pokladní deník za dané období
`~/register/print`	**GET**	Tisk pokladního deníku za dané období
Prefix: `{terminal}/{from}/{to}/`		
`~/summary`	**GET**	Přehled pokladny za dané období
`~/summary/print/simple`	**GET**	Tisk jednoduchého přehledu pokladny za dané období
`~/summary/print/full`	**GET**	Tisk kompletního přehledu pokladny za dané období

Tabulka 3.13: Specifikace API - Zákazníci

URI	Metoda	Popis
`customers`	**GET**	Všichni zákazníci
`customers`	**POST**	Vloží zákazníka
`~/{customer}`	**GET**	Konkrétní zákazník
`~/{customer}`	**PUT**	Úprava zákazníka
`~/{customer}`	**DELETE**	Odstranění zákazníka

Tabulka 3.14: Specifikace API - Pohyby zboží

URI	Metoda	Popis
`productMovements`	**GET**	Pohyby zboží
`productMovements`	**POST**	Přidat pohyb zboží
`~/productMovement`	**GET**	Konkrétní pohyb zboží
`~/filter/`	**GET**	Hledat v pohybech zboží
`~/in/`	**GET**	Pohyb zboží na pobočku
`~/move/`	**GET**	Přesun zboží
`~/out/`	**GET**	Prodej zboží
`~/product/{product}`	**GET**	Detail pohybu konkrétního produktu

Tabulka 3.15: Specifikace API - Organizace a pobočky

URI	Metoda	Popis
Prefix: `organizations/{organization}/branches`		
`/branch/terminals`	**GET**	Pokladny konkrétní pobočky
`/invoices`	**GET**	Faktury organizace
`/deliveryNotes`	**GET**	Dodací listy organizace
`/products/unpaired`	**GET**	Nespárované produkty organizace
`/products/unpairedInvoice`	**GET**	Produkty bez faktury pod určitou organizací
`/deliveryNotes/unpaired`	**GET**	Nespárované dodací listy organizace
`/branch/invoices`	**POST**	Faktury přijaté na pobočce
`/branch/products/product`	**GET**	Produkty na pobočce
`~/branch/date/invoice/vzp/print`	**GET**	Tisk faktury pro VZP
`~/branch/date/invoice/others/print`	**GET**	Tisk faktury pro ostatní pojišťovny
Prefix: `~{branch}/insuranceCompany/{insuranceCompany}/{date}`		
`export`	**GET**	Průvodní list exportu
`download`	**GET**	Datový soubor exportu ke stažení
`list`	**GET**	Etiketa exportu
`print`	**GET**	Tisk průvodního listu exportu

3.3.3 Tisk

Ve funkčních požadavcích, které vyplynuly z analýzy systému, se nacházel i tisk dokumentů. Realizaci tisku můžeme rozdělit na dvě disjunktní implementace. Jedna zajišťuje tisk účtenek a druhá se stará o tisk běžných dokumentů a exportů ze systému.

Tisk účtenek Tisk účtenek probíhá na specializované termotiskárně. BackLynx vygeneruje specifický binární soubor, který je poslán na tiskárnu. O poslání na tiskárnu se stará přímo Electron. Takovouto realizací tisku se nám povedlo udržet systém modulární, protože tisk nezávisí na technologii frontendu. S implementací tisku účtenek mi pomáhal Ing. Jan Matoušek, za což mu velice děkuji.

Tisk dokumentů Typů dokumentů, které umožňuje systém vytisknout na jakékoliv běžné tiskárně, je několik druhů. Patří k nim různé přehledy pokladen a transakcí, faktury pro pojišťovnu, karta zakázky a další. Opět jsme chtěli, aby tento tisk nebyl závislý na frontendu. Toho jsme dosáhli tak, že všechny tiskové dokumenty jsou sesazené jako obyčejný HTML dokument, který je na vyžádání frontendu vygenerován na backendu a zaslán jako obyčejná HTTP odpověď s daným typem obsahu. Díky tomu je tedy velmi jednoduché přidat další typy dokumentů a zodpovědnost je opět čistě na BackLynxu.

3.4 Deploy

Deploy je proces, při kterém je vývojová verze programu nasazena do produkčního prostředí. Celý proces je přímo propojen s repositářem zdrojových kódů v Gitlabu, jehož průběžná integrace zařídí sestavení aplikace. O této technologii jsme psali v sekcích Gitlab a Vývojové prostředí. Stačí tedy nahrát změnu kódu do GITu a systém automaticky nahraje a sestaví novou verzi aplikací na vývojové větvi. Jakmile je tato verze otestována, může být z administrace Gitlabu přesunuta do produkčního prostředí. Toto řešení jsem navrhl já a s jeho realizací mi pomáhal Bc. Pavel Kovář, který ve firmě Jagu s. r. o. působí také jako správce serverů.

3.5 Frontend

Jak jsme již několikrát zmínili, realizací samotného FrontLynxu se zabýval kolega. Naneštěstí měsíc před spuštěním pilotního provozu systému, který byl naplánován na únor 2018, neměl tolik času, aby mohl všechny potřebné části frontendu realizovat sám. Musel jsem tedy přebrat část jeho práce a implementovat i frontend. Tato práce samozřejmě probíhala za jeho plné asistence a zakládala se na jeho návrhu systému.

Bylo to pro mě velmi časově náročné, protože jsem s Angularem, ve kterém je systém implementován, neměl velké zkušenosti a musel jsem se tuto technologii velmi rychle naučit. Ve výsledku byla většina mé implementace přepsána, protože vývoj FrontLynxu byl rozdělen na dvě fáze. V první fázi, které jsem se při vývoji účastnil, byl použit design vycházející z knihovny Bootstrap[37]

a ve druhé fázi se přešlo na Material Design[38], jehož implementace jsem se už neúčastnil.

3.6 Distribuce

Distribuce části systému, kterou používá klient, je realizována pomocí Electronu. Electron je možné zkompilovat na všechny dnes běžné operační systémy a zároveň lze vytvořit instalátor. Distribuce tedy probíhá tak, že uživatel stáhne instalační soubor pro svoji platformu a nainstaluje Electron. Electron si po každém spuštění ověří jestli pro něj není na serveru umístěna aktualizace a pokud ano, nabídne uživateli možnost nahrát novou verzi. Poté si stáhne ze serveru zdrojové kódy FrontLynxu a spustí tuto aplikaci.

Kapitola 4

Nasazení a provoz systému

Nasazení bylo rozplánováno do dvou fází. V první fázi se náš systém používal pouze na prodej produktů a služeb a nejsložitější část systému zakázky se testovala společně s pracovníky optik. Ve druhé fázi již došlo k aktivnímu používání zakázek a systém měl plně nahrazovat nejdůležitější funkcionalitu původního softwaru.

4.1 Produkční prostředí

K datu 2. 5. 2018 je systém provozován na 5 pobočkách očních optik. BackLynx běží na virtuálním serveru, který spravuje firma Jagu s. r. o. Tento server má následující parametry:

CPU 2 x QEMU Virtual CPU 1.5.3 @ 2.40 GHz

RAM 8 GB

HDD 40 GB SSD

OS CentOS Linux 7.4.1708

Web server NGINX/1.12.2

Databáze PostgreSQL 10.1

PHP 7.1.13

Obrázek 4.1: Parametry produkčního serveru

Jakmile se bude zvětšovat počet klientů, budeme i škálovat tento virtuální stroj. Počítáme i s budoucím použitím vlastního dedikovaného serveru, na kterém poběží virtuální stroje pro každou organizaci zvlášť.

4.2 Přechod ze starého systému

Díky rozfázování nasazení systému jsme museli řešit přesun dat ze systému původního. Již před nasazením do omezeného provozu v únoru 2018 jsme provedli automatizovaný přesun skladu z Newtonu do našeho systému. Kvůli účetním uzávěrkám se musely zakázky vydávat v našem systému, ale jejich tvorba a realizace probíhala v původním systému. Díky špatné koordinaci mezi námi a zaměstnanci poboček došlo k vydávání zakázek ve špatných sazbách DPH. Optiky se obecně pohybují v 15 % a 21 % sazbě DPH.

Zaměstnanci ovšem všechny zakázky vydávali s 15 % DPH a tuto situaci jsme museli následně řešit. Řešením bylo ručně projít všechny původní zakázky a v našem systému je znova vydat. Daň z přidané hodnoty nám způsobila daleko více problémů. O tomto tématu se dočtete více v kapitole Elektronická evidence tržeb (EET).

4.3 Problémové oblasti

Při druhé fázi nasazení do produkce, to znamená při začátku používání zakázek, jsme začali zjišťovat, že funkční požadavky na systém nepokrývaly kompletní potřeby systému. K tomuto problému došlo díky analýze, která spoléhala čistě na informace, které nám dodali zaměstnanci optik. Přineslo to mnoho problémů, se kterými jsme se nakonec úspěšně poprali a všechny vyřešili. V této sekci popíšu některé z těchto neočekávaných problémů. Musím konstatovat, že příště bych se touto cestou nevydal, a požadavky předané od klienta bych pro kontrolu ověřil s pomocí externích zdrojů.

4.3.1 Elektronická evidence tržeb (EET)

Dle Zákona o evidenci tržeb 112/2016 Sb.[39] platí pro provozovny očních optik povinnost elektronicky evidovat tržbu. Implementace funkcionality pro plnění této povinnosti proběhla z naší strany bez problému. Komplikace nastaly až při používání systému. Při analýze jsme od klienta zjistili různé případy použitých sazeb DPH v zakázce. Naneštěstí tyto informace byly mylné a sazby DPH se řídily úplně jinými pravidly.

Tržba musí být do systému EET zaslána nejpozději do 48 hodin od uskutečnění platby. Pro vysvětlení přikládáme ukázku dat, která se posílají na API EET v případě, kdy se tržba skládá ze základu daně různých sazeb. Popis kompletní specifikace formátu přesahuje rozsah této práce, a proto nebude uvedena.

Zdrojový kód 4.1: Ukázka zprávy pro EET

```
<Data dic_popl="CZ00000000" id_provoz="111" id_pokl="velka-
    ↪ pokladna" porad_cis="999" dat_trzby="2018-05-05T11:59:59+
    ↪ 02:00" celk_trzba="200" zakl_dan1="79" dan1="21" zakl_dan
    ↪ 2="85" dan2="15" rezim="0" />
```

Tato zpráva říká, že byla přijata tržba ve výši 200 Kč a z toho do 21 % sazby spadalo zboží za 100 Kč a stejně tak do 15 % sazby. Ve chvíli, kdy se na položky v zakázce změnila pravidla pro určování DPH, musí tato zpráva obsahovat jiná data. Na tento problém jsme narazili až po třech týdnech používání systému. Museli jsme tedy zrušit všechny závadné odeslané tržby a odeslat je do systému ve správném tvaru.

V systému bylo v tu chvíli necelých sto zakázek, které jsme označili jako chybné. Museli jsme tedy implementovat podpůrný software pro vrácení zakázky do původního stavu, resetovat tržbu a zakázku zaplatit a vydat znova. Tento proces není vůbec jednoduchý, protože zakázku je možné platit složením různých platebních metod, na zakázku může být zaplacena záloha a při výdeji zakázky probíhá komunikace se skladem. Kompletní vyřešení tohoto problému nám zabralo více než 14 dní. Takže takto drobná chyba, kdy jsme si neověřili informace poskytnuté klientem, při analýze vyšla hodně draho.

4.3.2 Aritmetika decimálních čísel

Tento problém se vymyká ostatním problémům zde zmíněných, protože nevznikl na základě chybné analýzy, ale vznikl špatnou implementací části systému. Ve standardních implementacích většiny moderních programovacích jazyků jsou desetinná čísla reprezentovaná pomocí plovoucí řádové čárky. Dle [40] je tato reprezentace založena na exponenciální notaci. V této notaci je nenulové reálné číslo x vyjádřeno v desítkové soustavě jako

$$x = \pm S \times 10^E, \text{ kde } 1 \leq S < 10, \text{ a } E \text{ je celé číslo,}$$

takže například reprezentace čísla 0.0024 je 2.4×10^{-3} v této notaci. V běžných počítačích reprezentujeme čísla binárně, takže například:

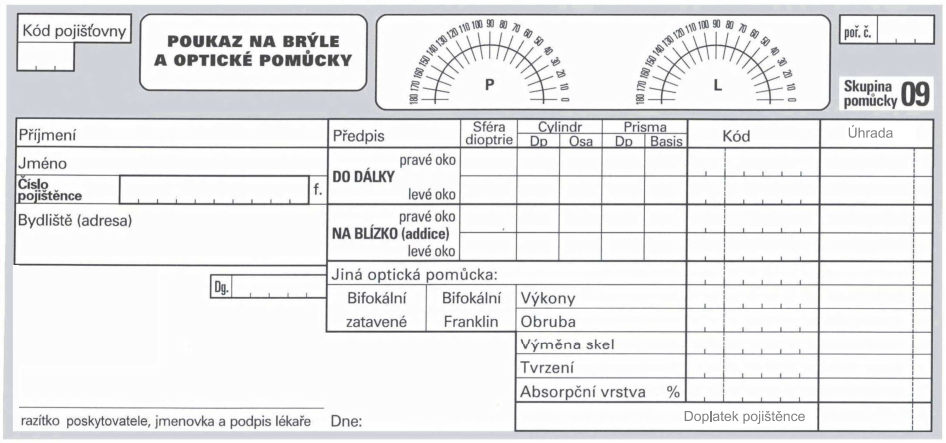
$$\frac{11}{2} = 1.011_2 \times 2^2$$

Z toho je zřejmé, že některá desetinná čísla mají nekonečný rozvoj ve dvojkové soustavě. Protože exponent E má omezený rozsah, musíme čísla zaokrouhlit.

PHP používá tuto reprezentaci čísel, a proto se například $200 - 107.1 - 0 - 92.9 - 0 = -1.4210854715202 \times 10^{14}$ a ne 0, jak bychom předpokládali. Na finanční matematiku se proto hodí aritmetika s libovolnou přesností. Tento problém tedy vznikl na základě mé nezkušenosti s finančním softwarem. Problém jsme vyřešili použitím knihovny Brick Math[41], která implementuje aritmetiku s neomezenou přesností. Tuto knihovnu jsme použili ve všech částech systému, kde se počítá s penězi.

4.3.3 Exporty pro pojišťovnu

Proces vzniku dat, která je potřeba předat zdravotním pojišťovnám, vzniká už při tvorbě zakázky. Zákazník dostane od lékaře poukaz na zdravotní pomůcky. Na tomto poukazu se nachází jednotlivé slevy a jejich kódy dle ceníku zdravotních pojišťoven. Dále poukaz obsahuje rodné číslo, identifikační číslo zdravotního zařízení, které předpis vydalo, a datum vydání.



Obrázek 4.2: Poukaz na brýle a optické pomůcky

O tyto částky je poté ponížena cena celé zakázky. Naneštěstí kvůli různým sazbám daní je potřeba přesně spárovat jednotlivé slevy na konkrétní položky v zakázce. Ze zaměstnanců optik nám nebyl nikdo schopen přesně říct, jak tento proces funguje, a museli jsme se tedy vrátit zpět k rýsovacím prknům. Po komunikaci s pojišťovnami působícími na území České republiky jsme nakonec byli schopni sestavit algoritmus, který řeší tuto problematiku. Bohužel to bylo velmi časově náročné a opět se nám nevyplatilo spoléhat se na analýzu založenou na informacích od klienta.

Tento proces má své pokračování. Optiky předávají pojišťovnám přehled těchto slev za každý měsíc, a to zpětně. Předávají se jak samotná data, tak několik doprovodných dokladů. Data se předávají v přesně daném datovém formátu, jehož popis je nad rámec této diplomové práce. Dále je potřeba vytisknout průvodní list se seznamem dávek pro jednotlivé pojištěnce a faktura. Některé z těchto dokladů si optiky psaly ručně a nám se podařilo je generovat přímo v systému. Dokonce jsme zavedli systémovou kontrolu validity poukazů vložených do systému, a tím pádem je chyba například zadaného kódu odhalena dříve, než probíhá komunikace s pojišťovnou.

4.3.4 Slevy

V analýze jsme zjistili, že by systém měl umožňovat poskytnout slevu na položky přidané do pokladny. Zahrnuli jsme tedy tento požadavek do realizace a v první verzi systému bylo uživatelům toto umožněno. V průběhu provozu ostrého systému ovšem zaměstnanci zjistili, že by systém měl umožňovat dát slevu na jakoukoliv položku ze zakázky. Tento na první pohled jednoduchý požadavek vyžadoval zásadní změnu struktury dat v databázi našeho systému.

Všechny tyto slevy musí být totiž evidované kvůli správnému odečtu ceny z konkrétní položky, která spadá do nějaké sazby DPH. Samotná úprava schématu databáze by nebyla problém, ale v systému již bylo mnoho dat, která se museli podrobit migraci. Tato úprava nám i s kompletní migrací původních dat a úpravou systému zabrala více než týden práce.

4.4 Chybějící části systému

V rámci této diplomové práce nebyly navrženy ani implementovány všechny potřebné části systému. Na některé funkční požadavky se přišlo až při reálném provozu aplikace a jejich odhalení v rámci analýzy bylo velmi nepravděpodobné až nemožné. Musíme však říci, že těchto chybějících částí je pouze malé procento a v žádném případě nebrání provozu systému.

4.4.1 Offline provoz

Celý systém vyžaduje neustále připojení k internetu, aby mohl FrontLynx komunikovat s BackLynxem. Tuto vlastnost jsme přijali už při návrhu systému jako fakt. Ovšem technologie použité při implementaci systému umožňují i offline provoz. Při startu Electronu si FrontLynx stáhne všechna aktuální data do lokální cache u klienta. Při nedostupnosti připojení k internetu se požadavek provede na stažených datech a zapíše se do speciálního logu. Ve chvíli, kdy se klient opět připojí do sítě, se tento log transakčním zpracováním propíše do dat na serveru.

Tento postup se na první pohled zdá jako snadný, ale není tomu tak. Bude potřeba řešit mnohé konflikty. Například co když dva terminály prodají zboží, kterého je ve skladu pouze jeden kus, a tak dále. Proto bude vhodné omezit množinu akcí, které uživatel může provádět v režimu offline.

4.4.2 Automatizace testování

V době psaní této práce není systém pokryt automatickými testy. Důvod tohoto faktu je čistě časová náročnost tohoto úkolu. Při návrhu a realizaci systému se s budoucím pokrytím testy počítalo, a díky tomu bude následná integrace velmi snadná. Připraveno je pokrytí třemi základními kategoriemi testů.

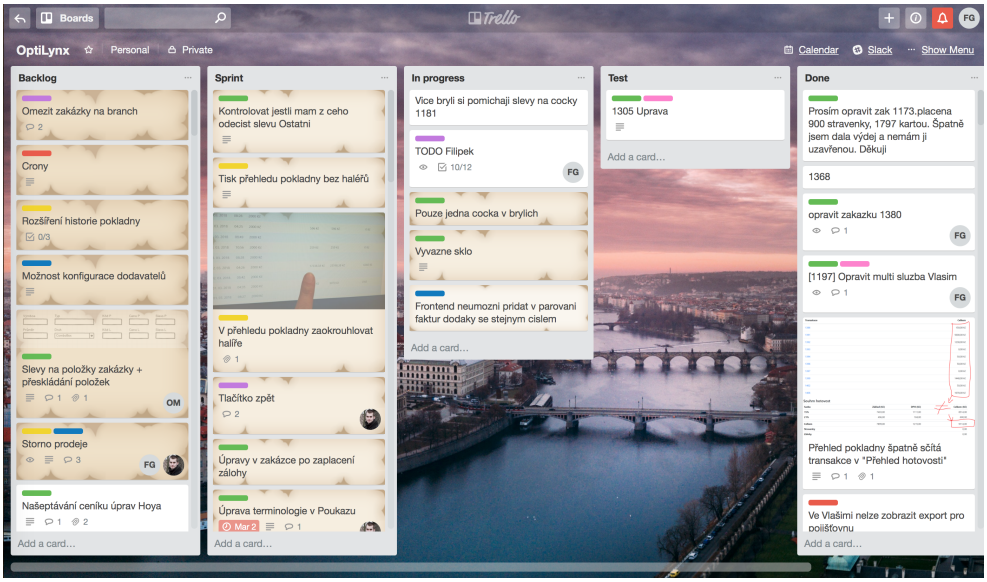
Unit Test Slouží k ověření korektnosti fungování celé implementace systému. Systém je rozdělen do samostatně funkčních jednotek, které je možné otestovat samostatně. Ověřením správnosti všech těchto jednotek můžeme systém považovat za korektní.[42] Námi použitý framework Laravel již obsahuje integraci frameworku pro unit testování PHPUnit.

HTTP Test Jedná se o specializovanou kolekci testů přímo pro testování RESTful API. Těmito testy bude pokryt kompletní modul BackLynx. Můžeme testovat celý obsah odpovědi serveru. To znamená, že můžeme ověřovat samotné HTTP kódy odpovědi nebo část obsahu. Tyto testy jsou opět součástí Laravelu a jsou založeny na výše zmíněných unit testech.[43]

Browser Test Jedná se o test na nejvyšší úrovni. Tento proces simuluje reálný pohyb uživatele systémem. Laravel má přímou podporu pro takovéto testování. Ta je bohužel uzpůsobena přímo na míru view vrstvě tohoto frameworku, kterou používáme pouze pro generování obsahu k tisku. Pro univerzální testování FrontLynxu přímo, plánujeme použít Selenium IDE, se kterým máme bohaté zkušenosti.[44]. Tímto způsobem bude nepřímo otestován i BackLynx. Tuto implementaci využívá nástroj vyvinutý přímo pro testování aplikací implementovaných v Angularu, Protractor, který nám zjednoduší tento typ komplexního testování.

4.4.3 Podpora

Pro podporu poskytovanou klientům jsme využívali několik kanálů. Zaměstnanci poboček zprvu hlásili problémy po telefonu, protože se často stávalo, že problém byl nahlášen někomu, ale zodpovědnost za jeho vyřešení často připadala mně nebo kolegovi Jaroslavovi Hráchovi. Vyhodnotili jsme jako vhodné začít tyto požadavky skladovat v nějakém tiketovacím systému. Systém Redmine, který jsme používali na vývoj není, tak vhodný na tento druh použití. Zvolili jsme Trello.com[45]. V tomto systému jsme jednotlivé úkoly rozdělili do pěti skupin: Backlog, Sprint, In progress, Test a Done, přičemž se úkol postupným přesouváním zleva doprava dokončil.



Obrázek 4.3: Nástěnka projektu OptiLynx

Musím poznamenat, že o pořádek v tomto systému a jeho agendě se řádně staral kolega Bc. Oldřich Malec, za což mu patří velké díky. Ke dni 5. 5. 2018 se v tomto systému nachází 99 úkolů k řešení, 5 úkolu, které se aktuálně řeší, a konečně 124 úkolů vyřešených.

4.4.4 Vzdálená správa

Díky webové koncepci celého systému je možné k datům v systému přistupovat odkudkoliv na světě, kde je internetové připojení. Díky multiplatformnosti celého řešení nejsme omezeni ani použitým zařízením či operačním systémem. Jediné, co nám v tomto ohledu schází, je vzdálená správa terminálu klienta. Tak, abychom se mohli připojit přímo na zařízení, kde běží OptiLynx, a společně s klientem mohli řešit daný problém. Velká výhoda v tomto postupu je, že klient vidí, co dělá podpora, a naopak. Ideální je tento postup spojit i s hlasovou či textovou komunikací.

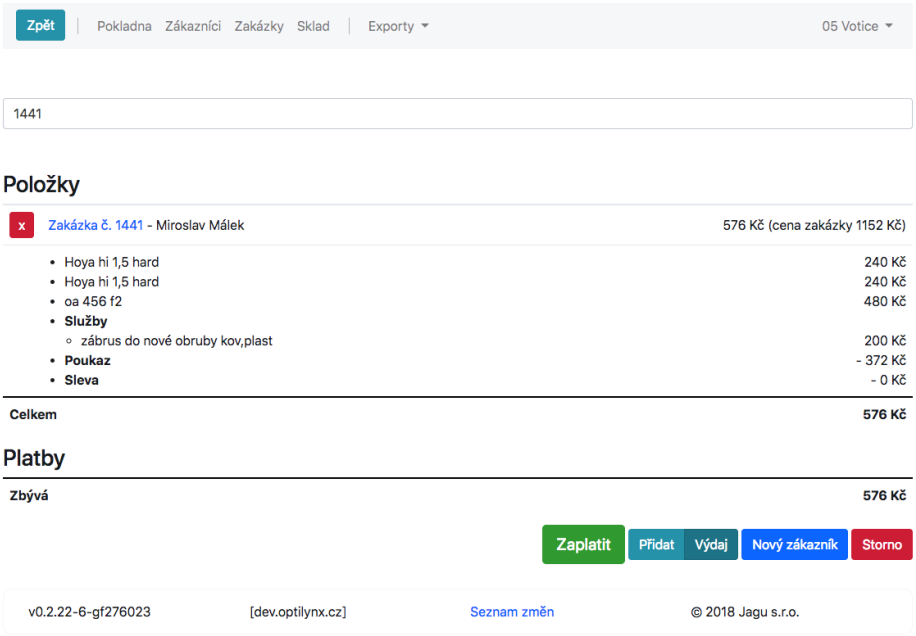
Takovéto aplikace samozřejmě existují a jsou hojně používané. Jejich nevýhodou, alespoň pro naše řešení, je zpoplatněné komerční použití. Z tohoto důvodu jsme minimálně pro pilotní provoz nemohli žádný software použít. Začali jsme se tedy zabývat otázkou, zda by nebylo možné implementovat vlastní řešení přímo v rámci Electronu. Jisté možnosti jsou, ale jejich implementace, přesahuje rámec této diplomové práce. Z tohoto důvodu je pouze nastíníme.

Základní variantou je pouze snímání obrazovky[46] implementované přímo v Electronu. Pokročilejší možnosti by nabízela implementace vlastního programu, který by se distribuoval společně s instalací Electronu. V současné

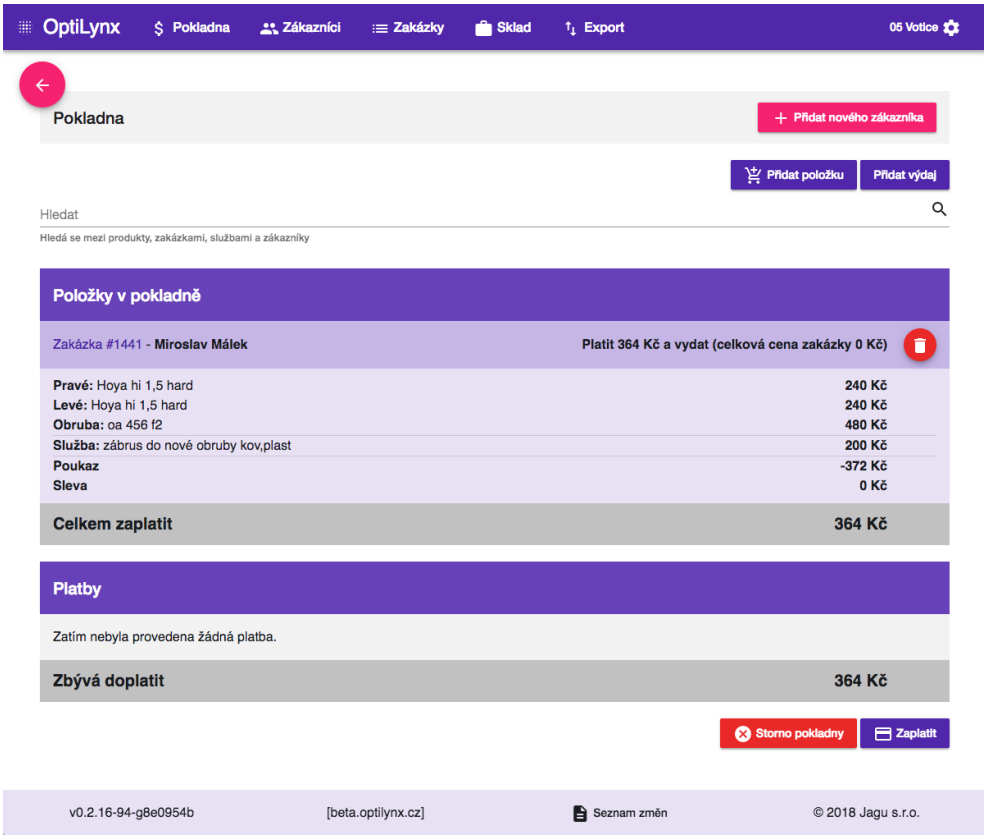
době totiž protokol WebRTC neumožňuje ovládat jiné zařízení. Z hlediska bezpečnosti[47] nelze ani počítat s budoucí implementací.

4.5 Výsledná aplikace

Přestože FrontLynx není předmětem této práce, přijde mi vhodné představit výslednou aplikaci. První obrázek reprezentuje původní verzi na které jsem se podílel i při implementaci a byla nasazena do pilotního provozu. Druhý aktuální implementaci FrontLynxu, kterou již implementoval kolega Bc. Jaroslav Hrách. Podrobnější srovnání naleznete v příloze Ukázka FrontLynxu. Celá tato část sytému staví na datové části BackLynxu, jehož implementace byla součástí této práce.



Obrázek 4.4: Pokladna - Bootstrap



Obrázek 4.5: Pokladna - Material

4.6 Budoucnost systému

V tuto chvíli můžeme systém považovat za úspěšný. Již přes dva měsíce běží v plném provozu na čtyřech pobočkách optik. Pravdou je, že systém takovýchto rozměrů vyžaduje neustálou údržbu a zlepšování. V příštích týdnech se zaměříme na testování běhu systému pro více organizací a zapracujeme na optimalizaci. Současně začneme jednat s dalšími potenciálními klienty a na základě této komunikace vytvoříme cenovou politiku.

Do budoucna plánujeme vytvořit komplexnější administraci systému. Tyto kroky budou vyžadovat zmrazení stávající stabilní verze a distribuci více různých verzí API.

Závěr

Cílem této diplomové práce bylo provést analýzu potřeb očních optik a na základě této analýzy navrhnout a implementovat informační systém, který bude pokrývat nalezené funkční požadavky zjištěné při analýze. Zároveň byl systém nasazen do reálného provozu v několika očních optikách a ještě v rámci této práce vylepšován a dále vyvíjen. Musím konstatovat, že všechny tyto cíle jsem splnil a společně s Bc. Jaroslavem Hráchem a ostatními kolegy se nám povedlo vyvinout velmi komplexní a funkční systém, který plní stanovené cíle.

Samotná implementace se velmi dobře udržuje a rozšiřuje díky moderním a čistým postupům při návrhu a vývoji aplikací, ze kterých je systém složen.

Během implementace a později samotném provozu systému jsem narazil na několik nedostatků návrhu. Většinu těchto problémů se mi povedlo vyřešit a u ostatních mám již navržené řešení, které budeme implementovat později, více v Chybějící části systému.

Prací na vývoji tohoto systému jsem strávil více než 700 hodin, které jsem vykazoval do systému Redmine. Do tohoto výkazu nepočítáme čas strávený na samotné tvorbě této diplomové práce. Pevně věřím, že systém usnadní práci mnoha pracovníkům očních optik a přispěje i ke spokojenosti zákazníků těchto firem.

Tato práce pro mne byla velkou výzvou a reálný provoz v očních optikách potvrzuje, že jsem ji zvládl. V blízké budoucnosti nás čeká rozšiřování systému na další pobočky očních optik a samozřejmě kontinuální vývoj a údržba systému, kterému se u softwaru tohoto typu nelze vyhnout.

Literatura

[1] Rohit Singh: Easy way to understand SDLC | Dignitas Digital [online]. červenec 2016, [vid. 24. 2. 2018]. Dostupné z: http://www.dignitasdigital.com/blog/easy-way-to-understand-sdlc/

[2] Autor neznámý: Overview - Redmine [online]. 2018, [vid. 5. 12. 2018]. Dostupné z: http://www.redmine.org

[3] Autor neznámý: OČNÍ OPTIKA BENEŠOV s.r.o. | Úvodní stránka | brýle, brýlové obruby, brýlové čočky, sluneční brýle, kontaktní čočky [online]. 2017, [vid. 29. 12. 2017]. Dostupné z: http://www.optikabenesov.cz/index.html

[4] Hull, E.: *Requirements Engineering*. London: Springer London, 2002, ISBN 9781447137306.

[5] Autor neznámý: Newton – specializovaný software pro oční optiku – Newton – spec. software [online]. [vid. 11. 4. 2018]. Dostupné z: http://www.maxoft.cz/newton-spec-software-pro-ocni-optiku/

[6] Chung, L.; Nixon, B.; Yu, E.; aj.: *Non-Functional Requirements in Software Engineering*. International Series in Software Engineering, Springer US, 2012, ISBN 9781461552697. Dostupné z: https://books.google.cz/books?id=MNrcBwAAQBAJ

[7] Salam, A.; Gilani, Z.; Haq, S.: *Deploying and Managing a Cloud Infrastructure: Real-World Skills for the CompTIA Cloud+ Certification and Beyond: Exam CV0-001*. Wiley, 2015, ISBN 9781118875582. Dostupné z: https://books.google.cz/books?id=1xQqBgAAQBAJ

[8] Ing. Pavel Žikovský PhD.: 2. přednáška - Návrh UI, prototypy. Přednáška, 2017.

[9] Autor neznámý: Prototypes, Specifications, and Diagrams in One Tool | Axure Software [online]. 2017, [vid. 11. 11. 2017]. Dostupné z: https://www.axure.com

[10] Parcell, S.: *Four Historical Definitions of Architecture*. MQUP, 2012, ISBN 9780773586871. Dostupné z: https://books.google.cz/books?id=liL7arj9a24C

[11] Bosch, J.; Gentleman, M.; Hofmeister, C.; aj.: *Software Architecture: System Design, Development and Maintenance: 17th World Computer Congress – TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture (WICSA3), August 25–30, 2002, Montréal, Québec, Canada*. IFIP Advances in Information and Communication Technology, Springer US, 2013, ISBN 9780387356075. Dostupné z: https://books.google.cz/books?id=ZRbnBwAAQBAJ

[12] Autor neznámý: Electron | Build cross platform desktop apps with JavaScript, HTML, and CSS. [online]. 2018, [vid. 20. 2. 2018]. Dostupné z: https://electronjs.org/

[13] Jablonski, S.; Petrov, I.; Meiler, C.; aj.: *Guide to Web Application and Platform Architectures*. Springer Professional Computing, Springer Berlin Heidelberg, 2004, ISBN 9783540009474. Dostupné z: https://books.google.cz/books?id=Ajn-nP7IdRQC

[14] Ritchie, C.: *Relational Database Principles*. Continuum, 2002, ISBN 9780826457134. Dostupné z: https://books.google.cz/books?id=vYBzeAJOg7IC

[15] David Kovář: Objektové databáze - Linux E X P R E S [online]. listopad 2007, [vid. 1. 11. 2017]. Dostupné z: https://www.linuxexpres.cz/business/objektove-databaze

[16] Sumathi, S.; Esakkirajan, S.: *Fundamentals of Relational Database Management Systems*. Studies in Computational Intelligence, Springer Berlin Heidelberg, 2007, ISBN 9783540483977. Dostupné z: https://books.google.cz/books?id=RjnNA0GW0wsC

[17] Dunglas, K.: *Persistence in PHP with the Doctrine ORM*. Community experience distilled, Packt Publishing, 2013, ISBN 9781782164111. Dostupné z: https://books.google.cz/books?id=0NtdAgAAQBAJ

[18] Snell, J.; Tidwell, D.; Kulchenko, P.: *Programming Web Services with SOAP: Building Distributed Applications*. O'Reilly Media, 2001, ISBN 9780596552015. Dostupné z: https://books.google.cz/books?id=ALo1LxID5q0C

[19] Fielding, R. T.: *Architectural Styles and the Design of Network-based Software Architectures*. Dizertační práce, University of California Irvine, 2000.

[20] Richardson, L.; Ruby, S.: *RESTful Web Services*. O'Reilly Media, 2008, ISBN 9780596554606. Dostupné z: https://books.google.cz/books?id=XUaErakHsoAC

[21] Patni, S.: *Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS*. Apress, 2017, ISBN 9781484226650. Dostupné z: https://books.google.cz/books?id=xBZyDgAAQBAJ

[22] Autor neznámý: OpenAPI-Specification/2.0.md at master · OAI/OpenAPI-Specification [online]. duben 2017, [vid. 1. 12. 2017]. Dostupné z: https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md

[23] Autor neznámý: World's Most Popular API Framework | Swagger [online]. 2018, [vid. 12. 3. 2018]. Dostupné z: https://swagger.io

[24] Autor neznámý: nginx [online]. 2018, [vid. 25. 1. 2018]. Dostupné z: http://nginx.org/en/

[25] Autor neznámý: Web server performance comparison – DreamHost [online]. 2018, [vid. 22. 3. 2018]. Dostupné z: https://help.dreamhost.com/hc/en-us/articles/215945987-Web-server-performance-comparison

[26] Glazar, F.: *Systém pro podporu BI-DBS - semestrální práce*. Bakalářská práce, České vysoké učení technické v Praze, 2016.

[27] Autor neznámý: Co je Laravel? | Laravelblog.cz [online]. 2018, [vid. 14. 2. 2018]. Dostupné z: http://laravelblog.cz/co-je-laravel/

[28] Autor neznámý: laravel/laravel: A PHP framework for web artisans [online]. 2018, [vid. 4. 4. 2018]. Dostupné z: https://github.com/laravel/laravel

[29] Autor neznámý: PostgreSQL: About [online]. 2018, [vid. 20. 4. 2018]. Dostupné z: https://www.postgresql.org/about/

[30] Autor neznámý: TypeScript - JavaScript that scales. [online]. 2018, [vid. 1. 5. 2018]. Dostupné z: https://www.typescriptlang.org

[31] Autor neznámý: About Electron | Electron [online]. 2018, [vid. 1. 5. 2018]. Dostupné z: https://electronjs.org/docs/tutorial/about

[32] Autor neznámý: Chromium - The Chromium Projects [online]. 2018, [vid. 1. 5. 2018]. Dostupné z: https://www.chromium.org/Home

[33] Autor neznámý: About | Node.js | Electron [online]. 2018, [vid. 1. 5. 2018]. Dostupné z: https://nodejs.org/en/about/

[34] Autor neznámý: Compare Repositories - Open Hub [online]. 2018, [vid. 1. 5. 2018]. Dostupné z: https://www.openhub.net/repositories/compare

[35] Autor neznámý: fzaninotto/Faker: Faker is a PHP library that generates fake data for you [online]. 2018, [vid. 30. 4. 2018]. Dostupné z: https://github.com/fzaninotto/Faker

[36] Bassett, L.: *Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON*. O'Reilly Media, 2015, ISBN 9781491929452. Dostupné z: https://books.google.cz/books?id=Qv9PCgAAQBAJ

[37] Autor neznámý: Bootstrap · The most popular HTML, CSS, and JS library in the world. [online]. 2018, [vid. 1. 5. 2018]. Dostupné z: https://getbootstrap.com

[38] Autor neznámý: Material Design [online]. 2018, [vid. 1. 5. 2018]. Dostupné z: https://material.io

[39] Autor neznámý: Zákon o evidenci tržeb (úplné znění) - Podnikatel.cz [online]. 2018, [vid. 1. 5. 2018]. Dostupné z: https://www.podnikatel.cz/zakony/zakon-o-evidenci-trzeb/uplne/

[40] Overton, M.: *Numerical Computing with IEEE Floating Point Arithmetic:*. Other Titles in Applied Mathematics, Society for Industrial and Applied Mathematics, 2001, ISBN 9780898715712. Dostupné z: https://books.google.cz/books?id=EV_Ojz00A9IC

[41] Autor neznámý: brick/math: Arbitrary-precision arithmetic library for PHP [online]. 2018, [vid. 1. 5. 2018]. Dostupné z: https://github.com/brick/math

[42] Phillips, T.: *Dive into Unit Testing*. BookRix, 2016, ISBN 9783739653440. Dostupné z: https://books.google.cz/books?id=fYkjDAAAQBAJ

[43] Autor neznámý: HTTP Tests - Laravel - The PHP Framework For Web Artisans [online]. 2018, [vid. 1. 5. 2018]. Dostupné z: https://laravel.com/docs/5.6/http-tests

[44] Autor neznámý: Selenium IDE plugin [online]. 2016, [vid. 10. 10. 2017]. Dostupné z: http://www.seleniumhq.org/projects/ide/

[45] Autor neznámý: Trello [online]. 2018, [vid. 9. 2. 2018]. Dostupné z: https://trello.com

[46] Haojian Wu: electron-sample-apps/desktop-capture at master · hokein/electron-sample-apps [online]. 2018, [vid. 1. 5. 2018]. Dostupné z: https://github.com/hokein/electron-sample-apps/tree/master/desktop-capture

[47] Arin Sime: 3 Things WebRTC Cannot Do [online]. 2016, [vid. 1. 5. 2018]. Dostupné z: https://webrtc.ventures/2016/03/3-things-webrtc-cannot-do/

Příloha A

Seznam použitých zkratek

IS Informační systém

UI User Interface

SDLC Software Development Life Cycle

UX User Experince

RAM Random Access Memory

HW Hardware

HTML HyperText Markup Language

OOP Objektově orientované programování

ORM Objektově relační mapování

SOAP Simple Object Access Protocol

REST Representational State Transfer

WSDL Web Services Description Language

HTTP Hypertext Transfer Protocol

API Application Programming Interface

URI Uniform Resource Identifier

PHP Hypertext Preprocessor

SQL Structured Query Language

XML Extensible Markup Language

PL/PGSQL SQL Procedural Language

MVC Model-View-Controller architektura

JSON JavaScript Object Notation

CPU Central Processing Unit

QEMU Quick Emulator

HDD Hard Disk Drive

SSD Solid State Disk

OS Operační systém

DPH Daň z přidané hodnoty

EET Elektronická evidence tržeb

NPM Node Package Manager

GUI Graphical User Interface

CSS Cascading Style Sheets

JPEG Joint Photographic Experts Group

YAML Ain't Markup Language

TCP Transmission Control Protocol

UDP User Datagram Protocol

BSD Berkeley Software Distribution

IČZ Identifikační číslo poskytovatele zdravotních služeb

VZP Všeobecná zdravotní pojišťovna

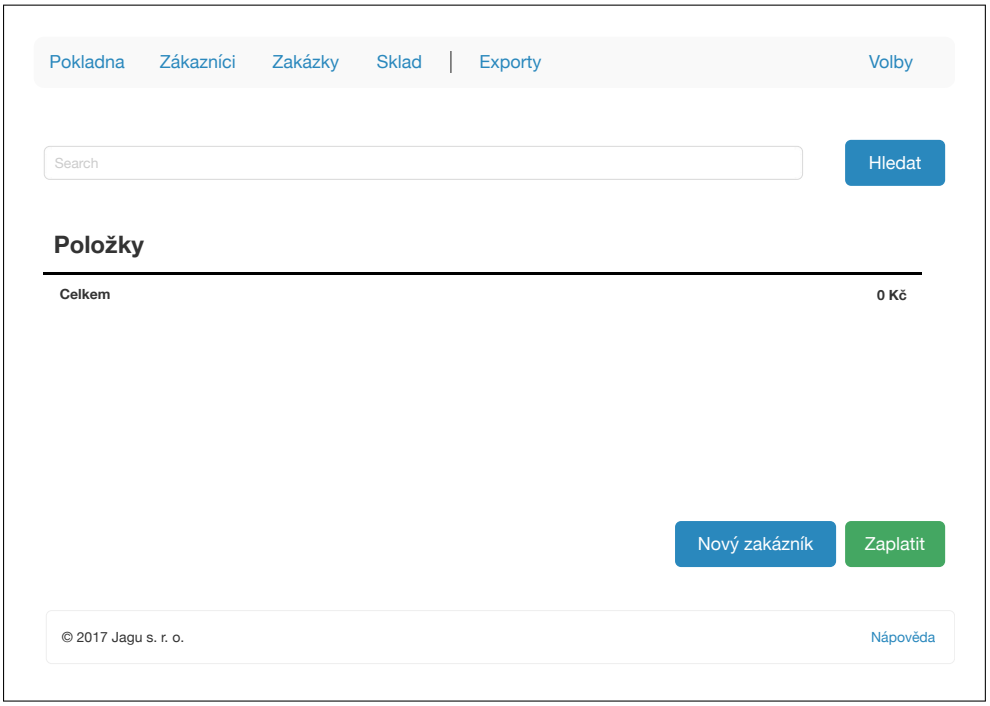
GIT Typ distribuovaného verzovacího systému

IDE Integrated Development Environment

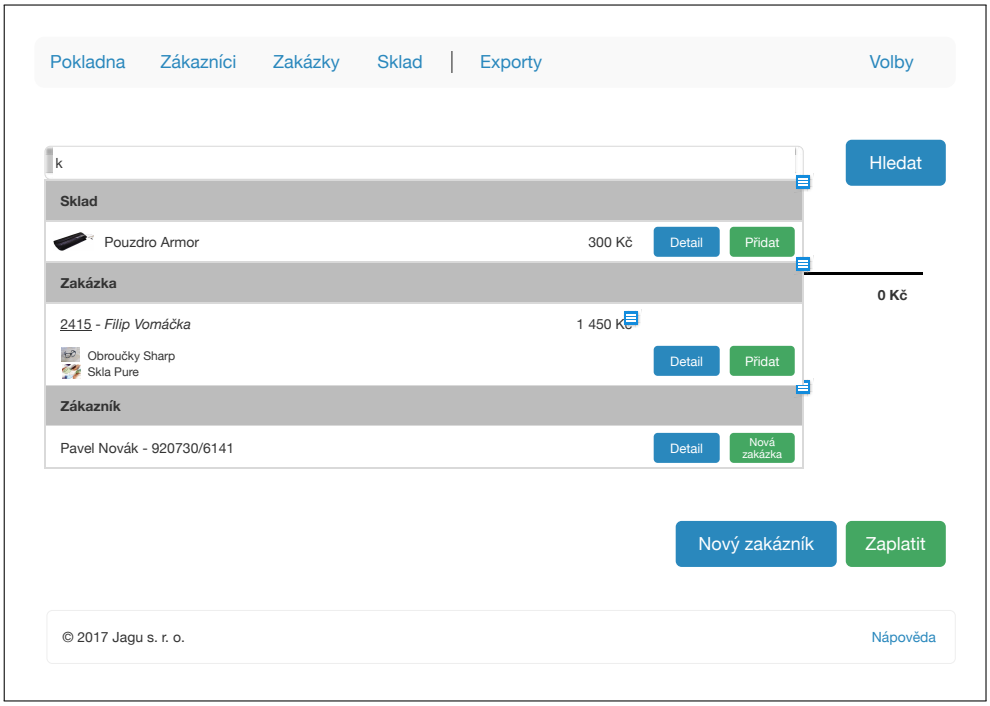
WebRTC Web Real-Time Communication

Příloha B

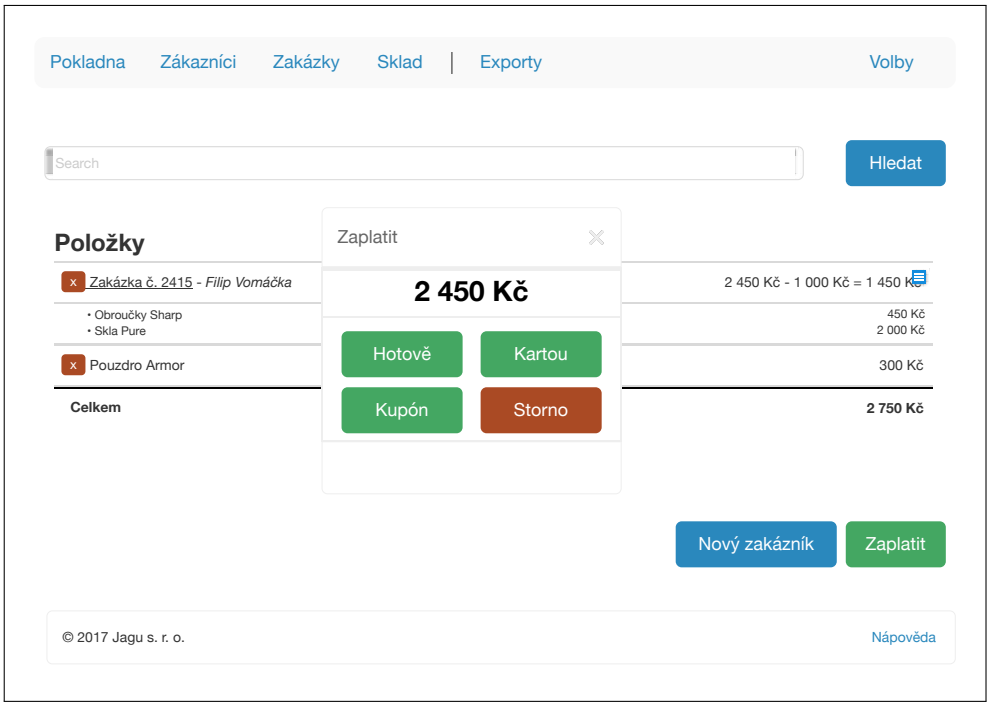
Prototyp aplikace



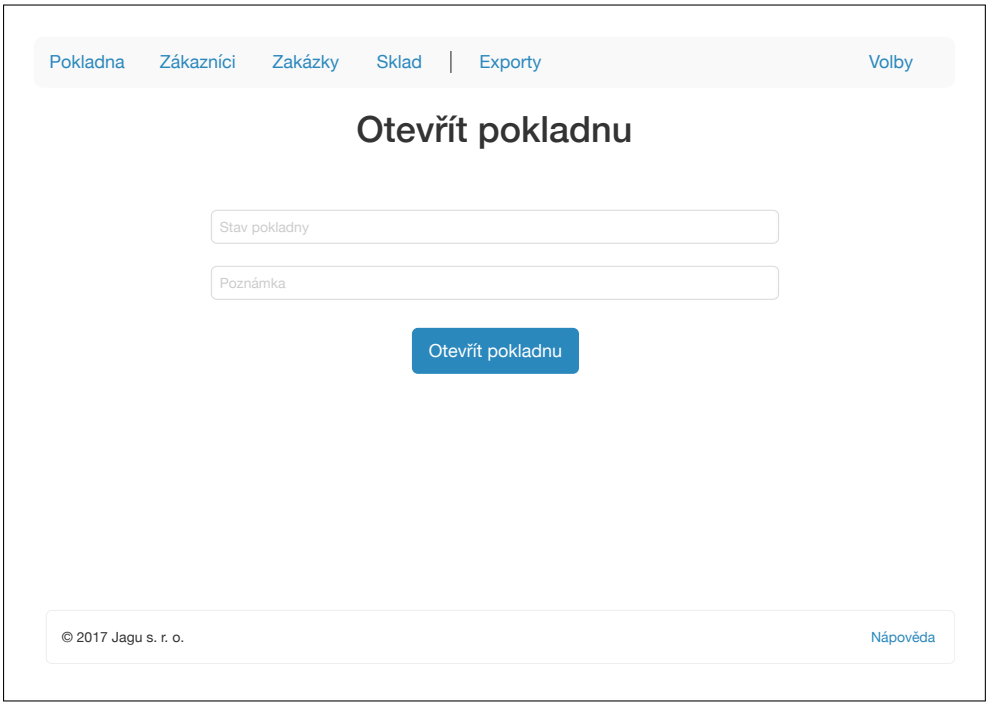
Obrázek B.1: Úvodní obrazovka pokladny



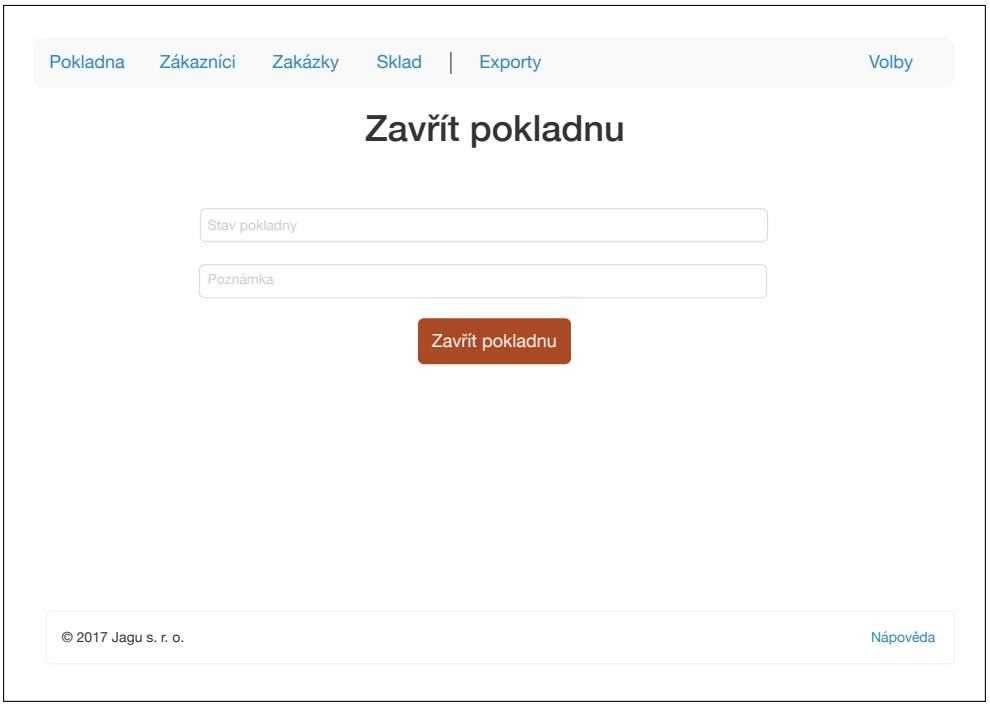
Obrázek B.2: Vyhledání položek pro vložení do pokladny



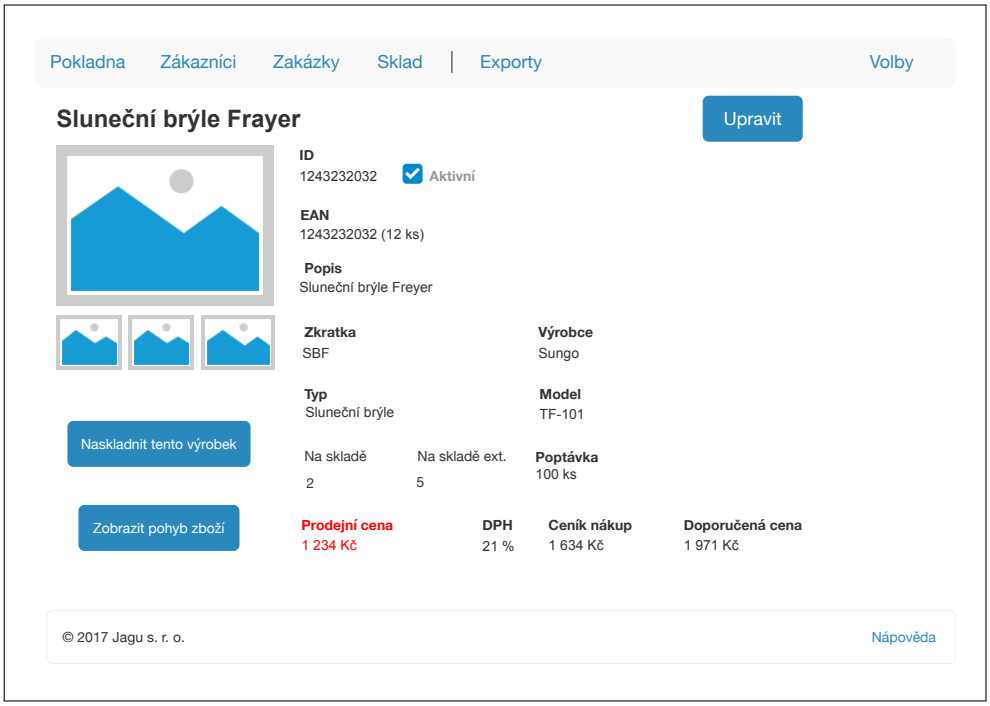
Obrázek B.3: Výběr typu platby v pokladně



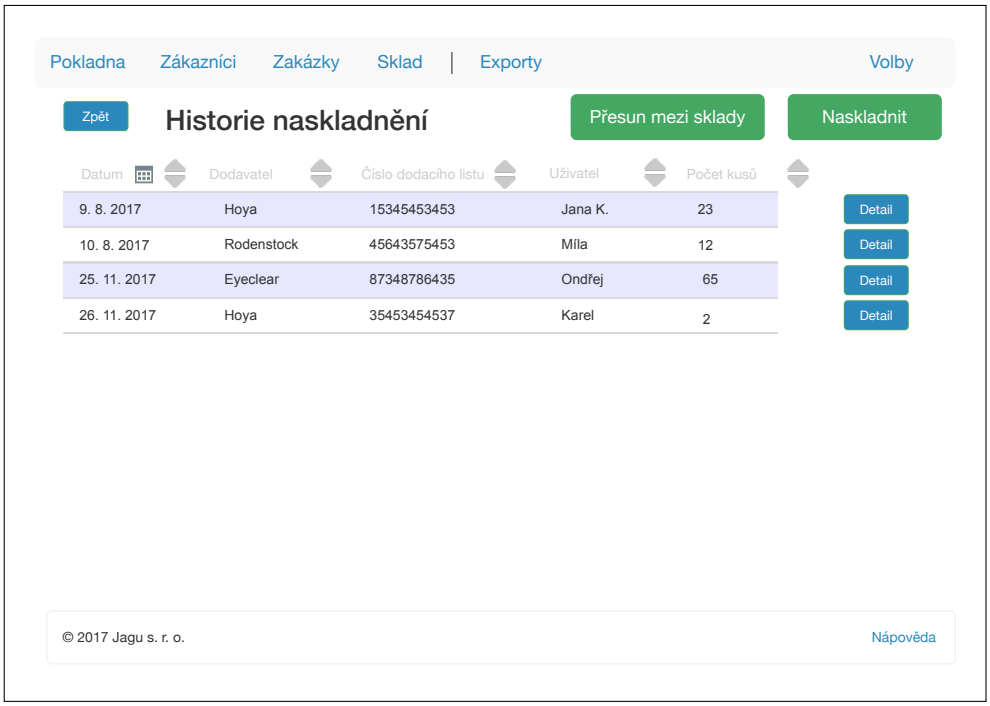
Obrázek B.4: Formulář pro otevření pokladny



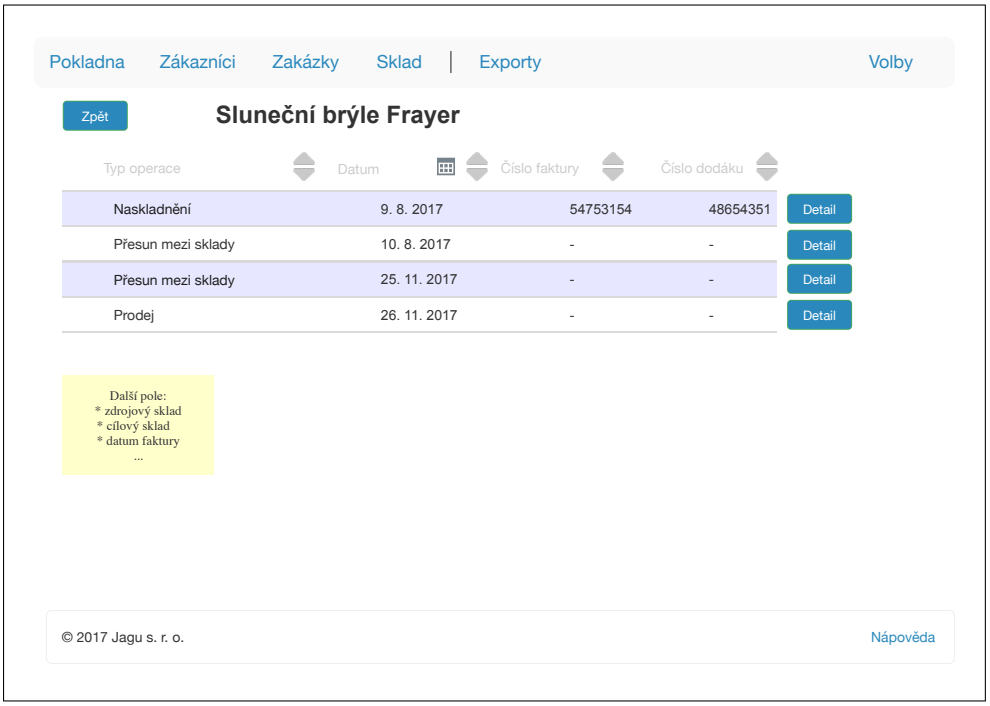
Obrázek B.5: Formulář pro zavření pokladny



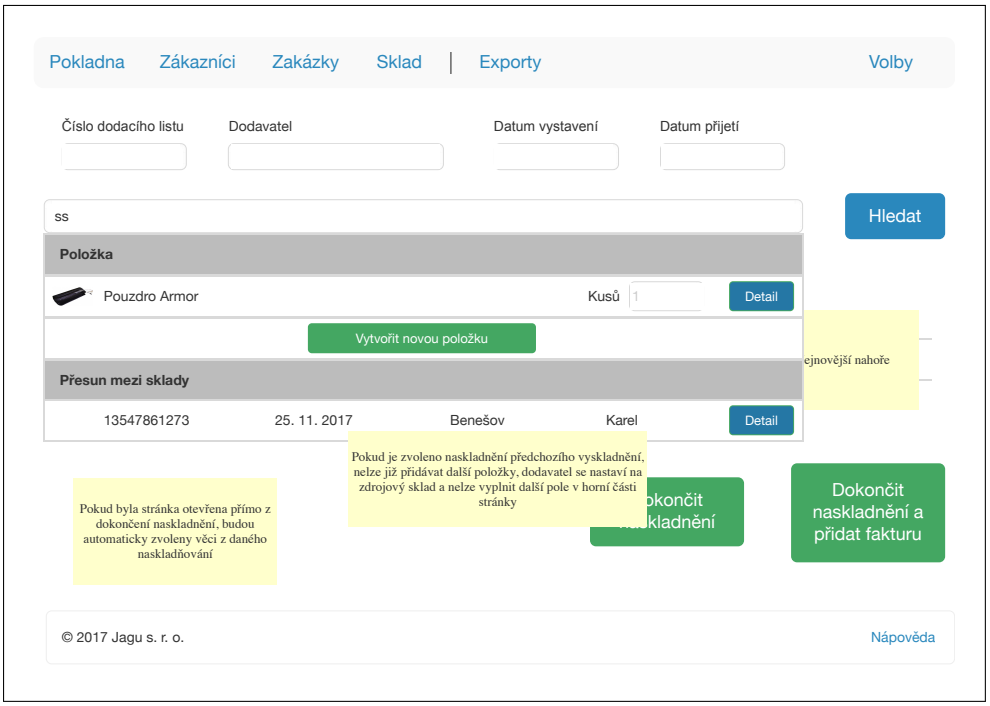
Obrázek B.6: Detail produktu ve skladu



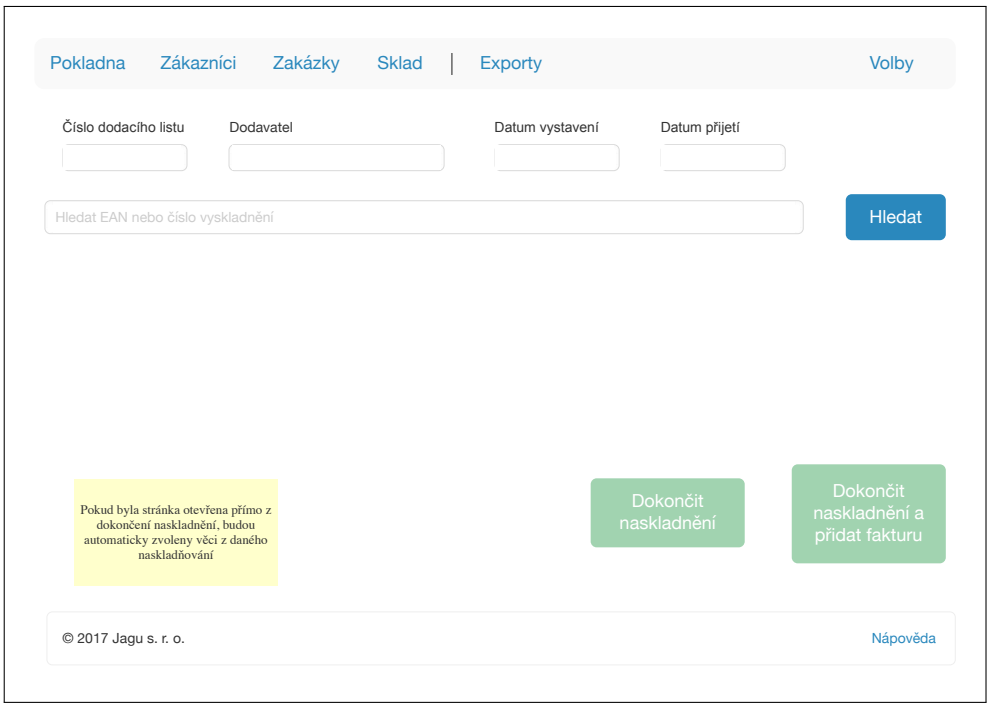
Obrázek B.7: Historie naskladnění pro pobočku



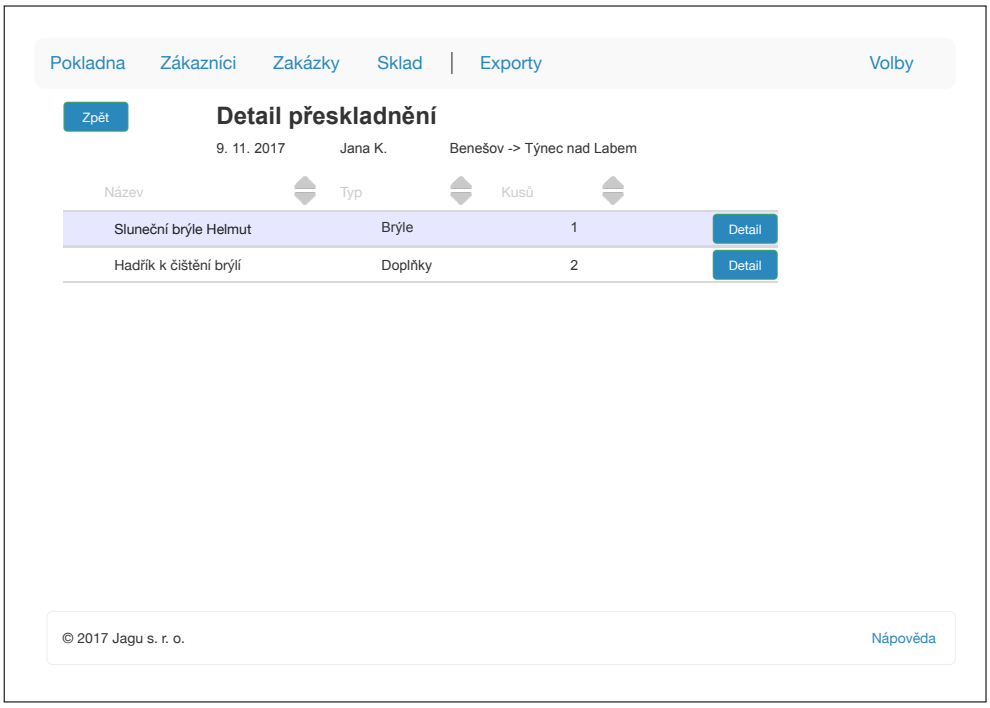
Obrázek B.8: Historie naskladnění produktu



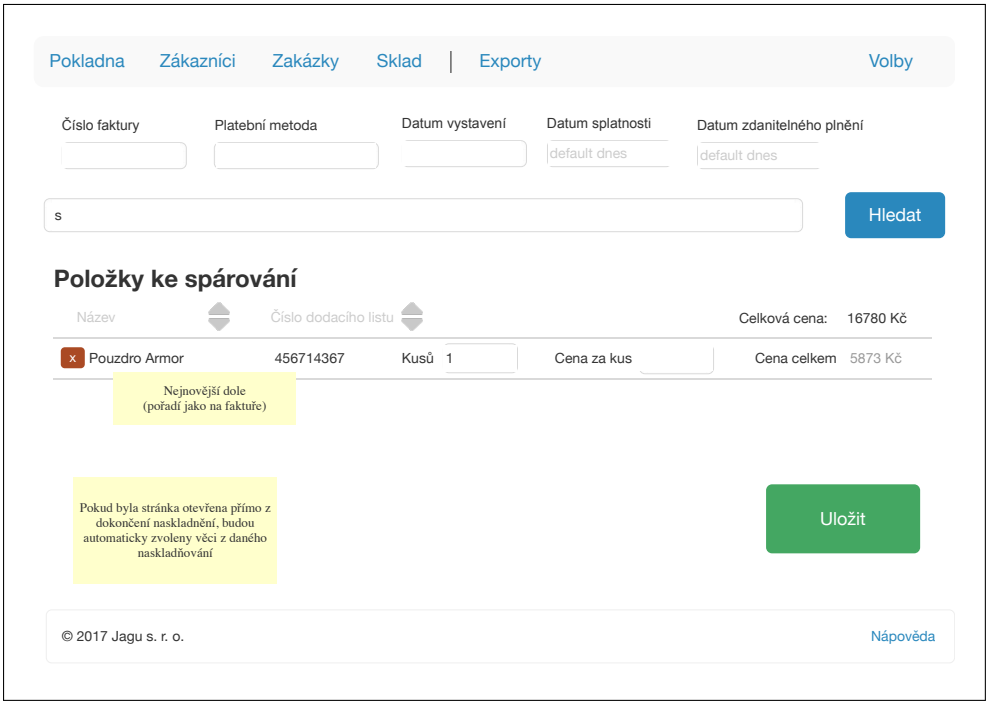
Obrázek B.9: Našeptáváč pro přidání produktu k naskladnění



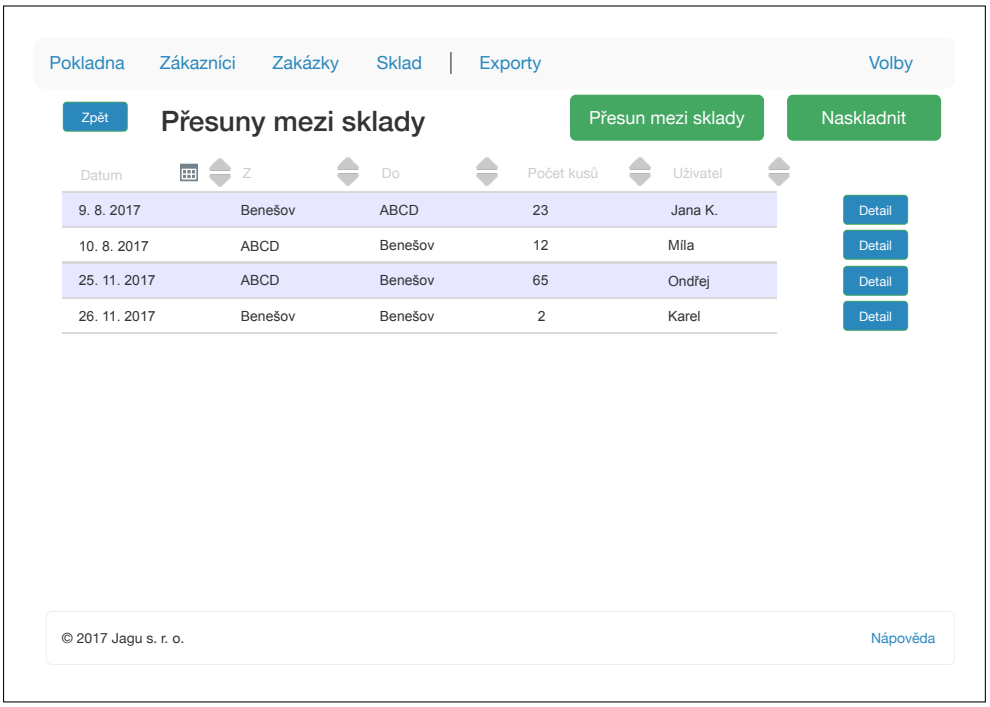
Obrázek B.10: Hlavní obrazovka naskladnění



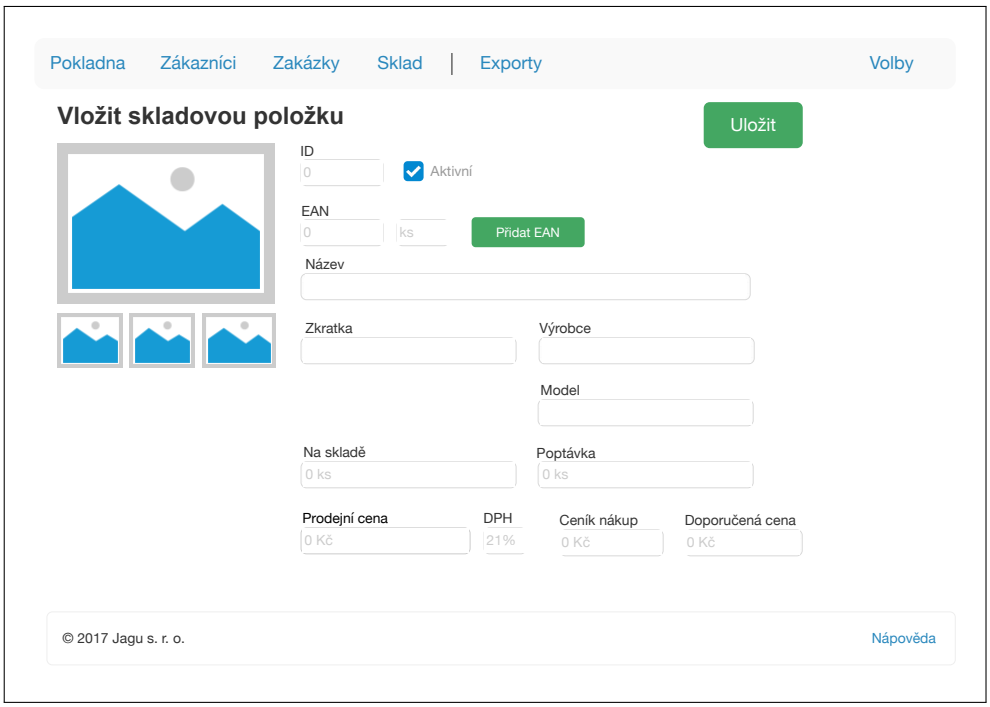
Obrázek B.11: Přesun zboží na jinou pobočku



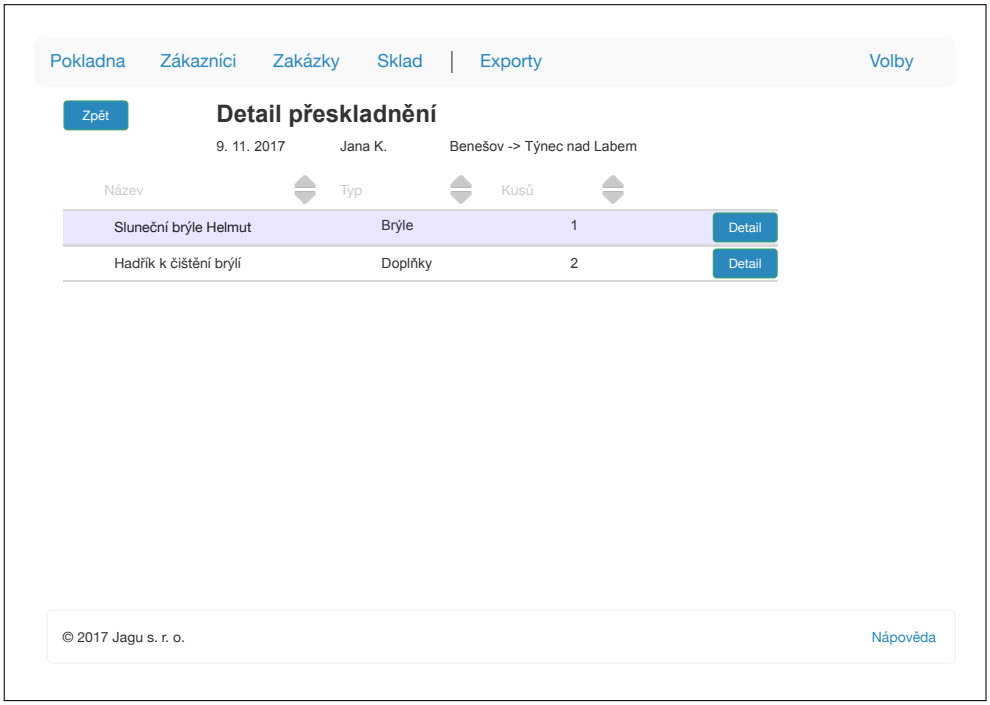
Obrázek B.12: Párování faktur ve skladu



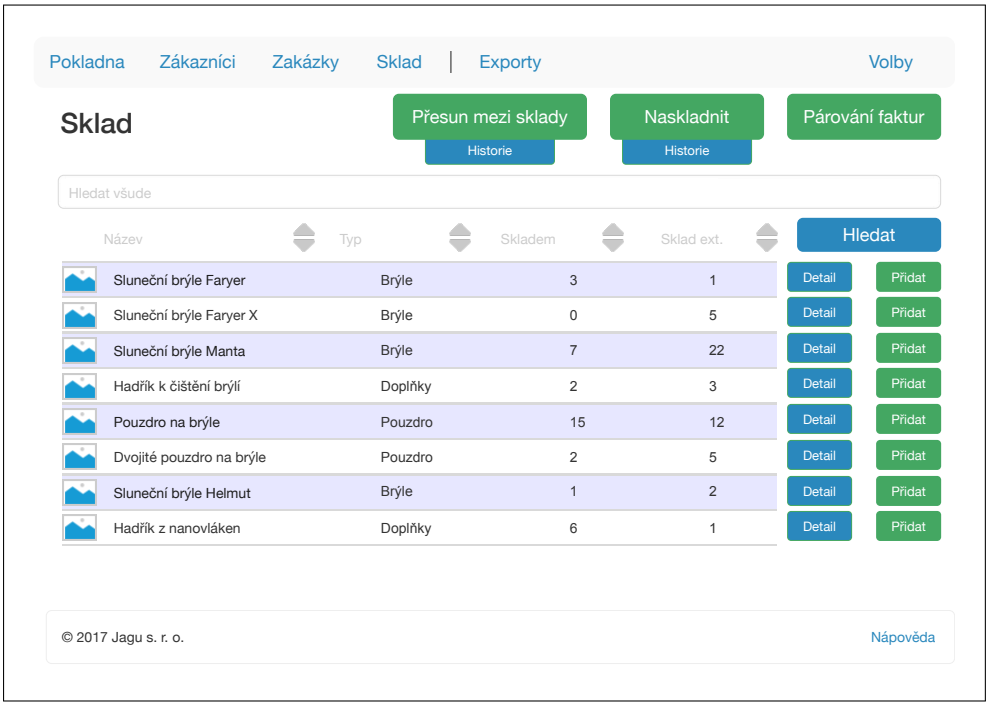
Obrázek B.13: Historie přeskladnění zboží



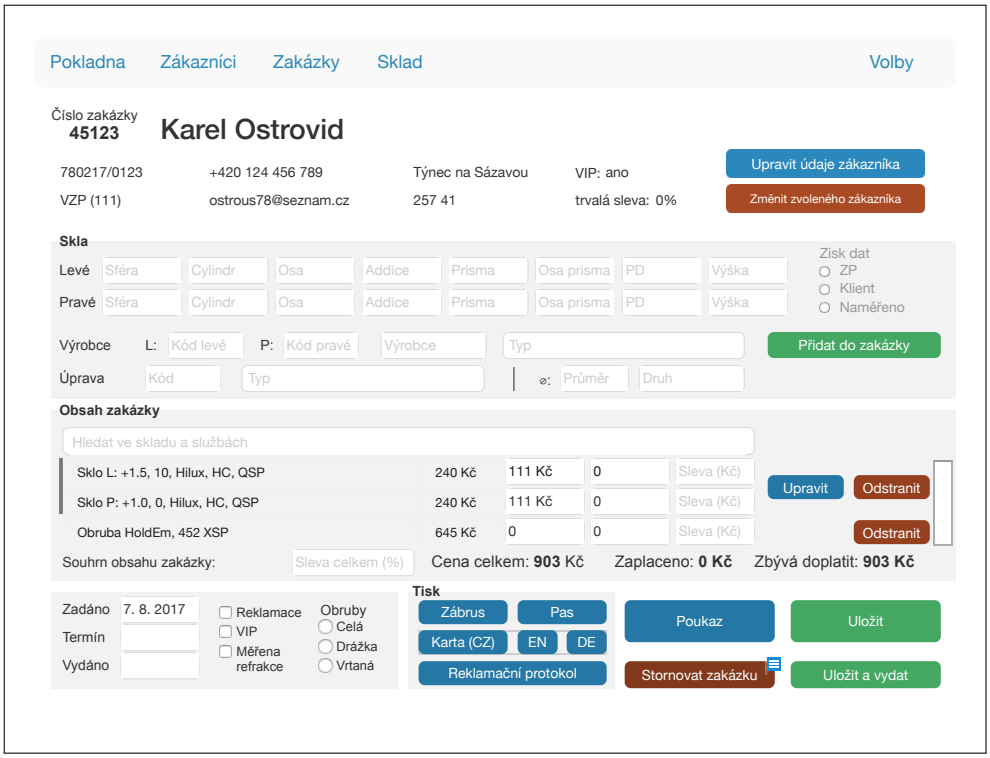
Obrázek B.14: Vytvoření skladové karty ve skladu



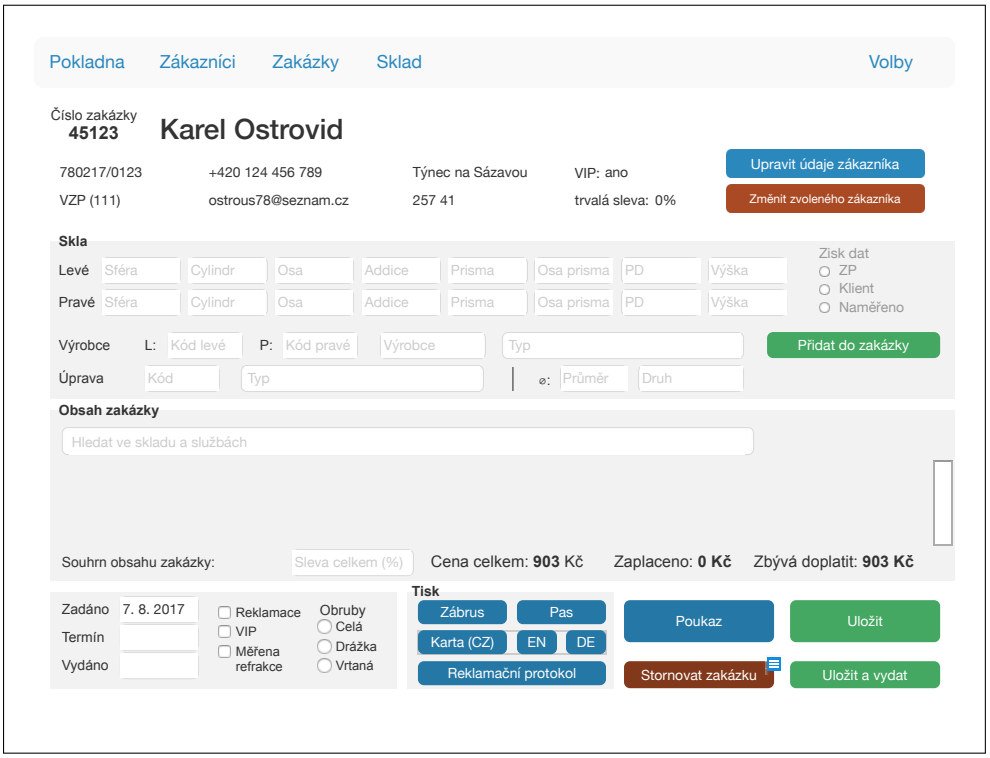
Obrázek B.15: Detail konkrétního naskladnění



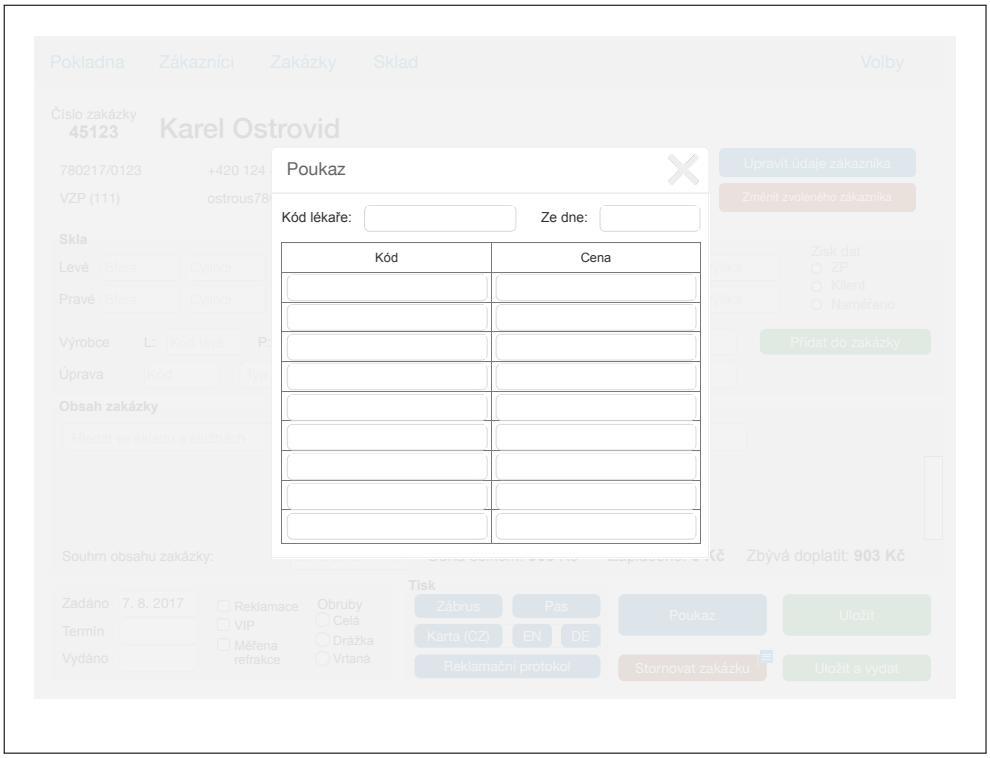
Obrázek B.16: Hlavní obrazovka skladu



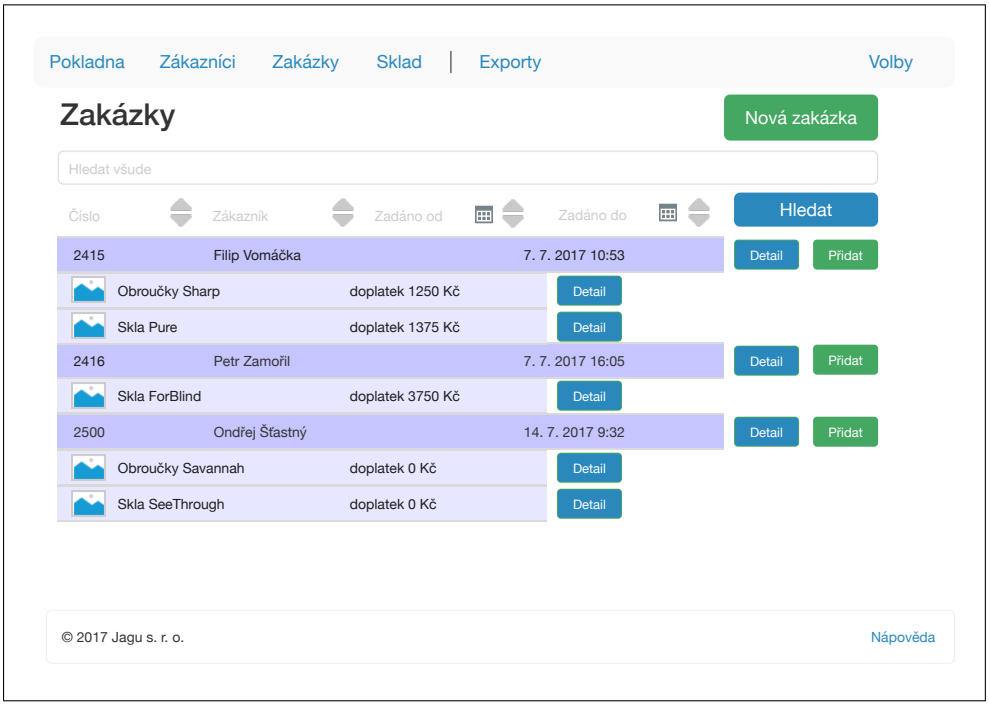
Obrázek B.17: Vytvořená zakázka



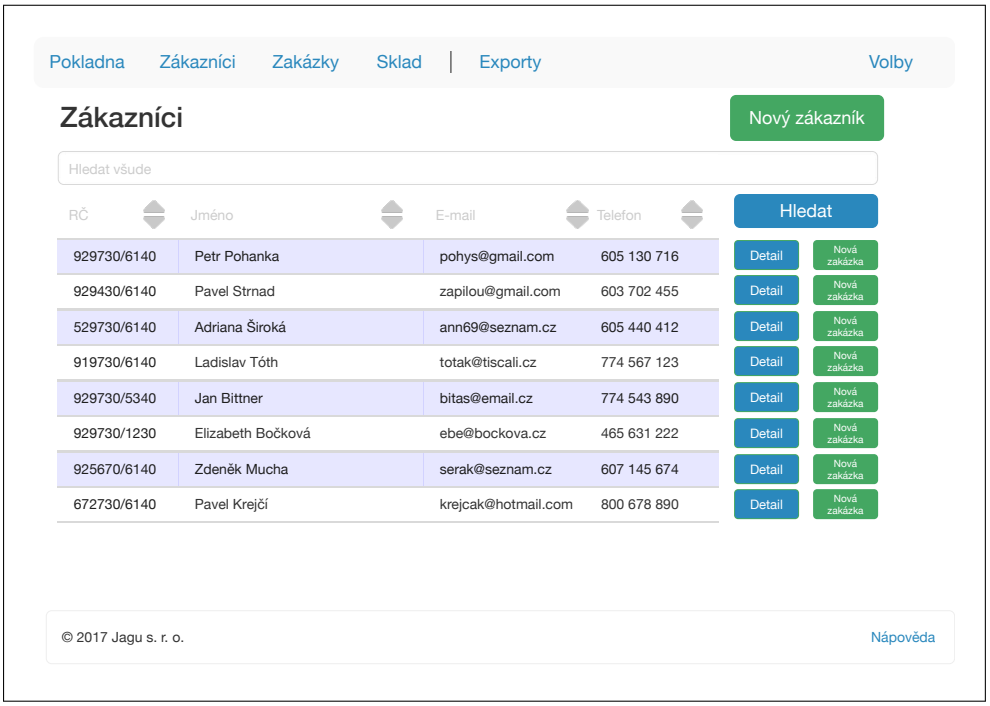
Obrázek B.18: Formulář pro vložení nové zakázky



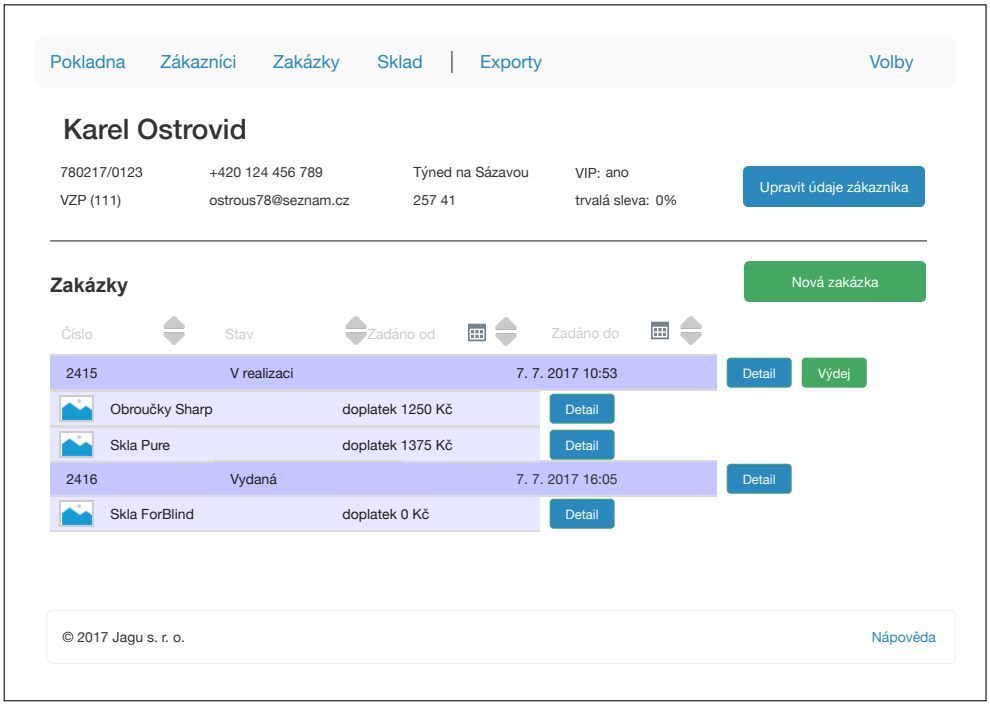
Obrázek B.19: Vyplnění poukazu pro pojišťovnu



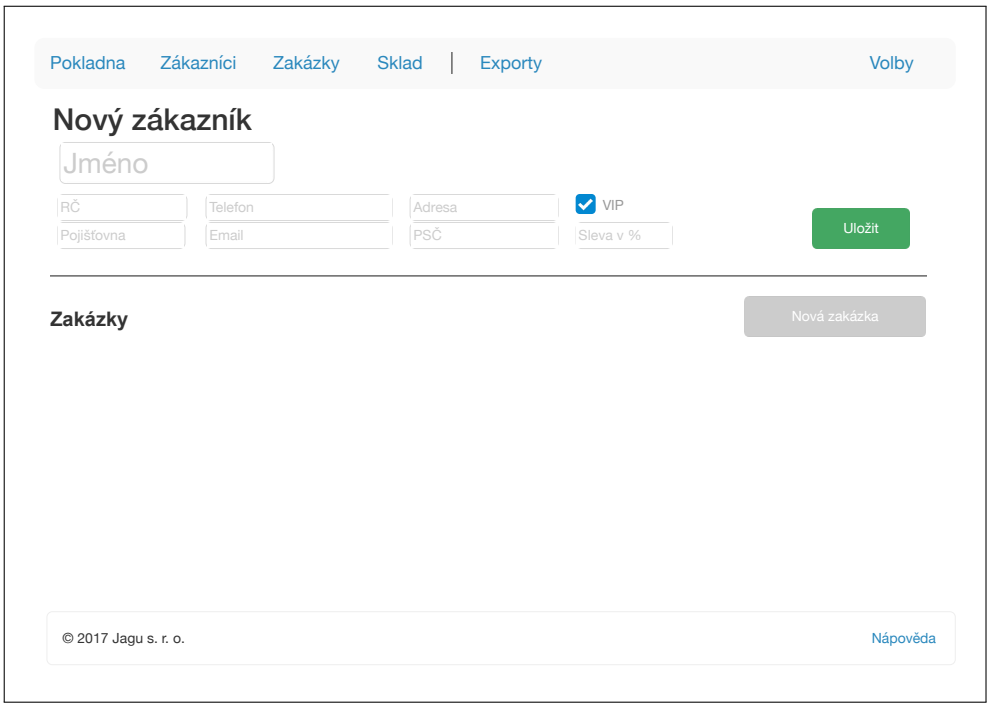
Obrázek B.20: Přehled zakázek



Obrázek B.21: Přehled zákazníků



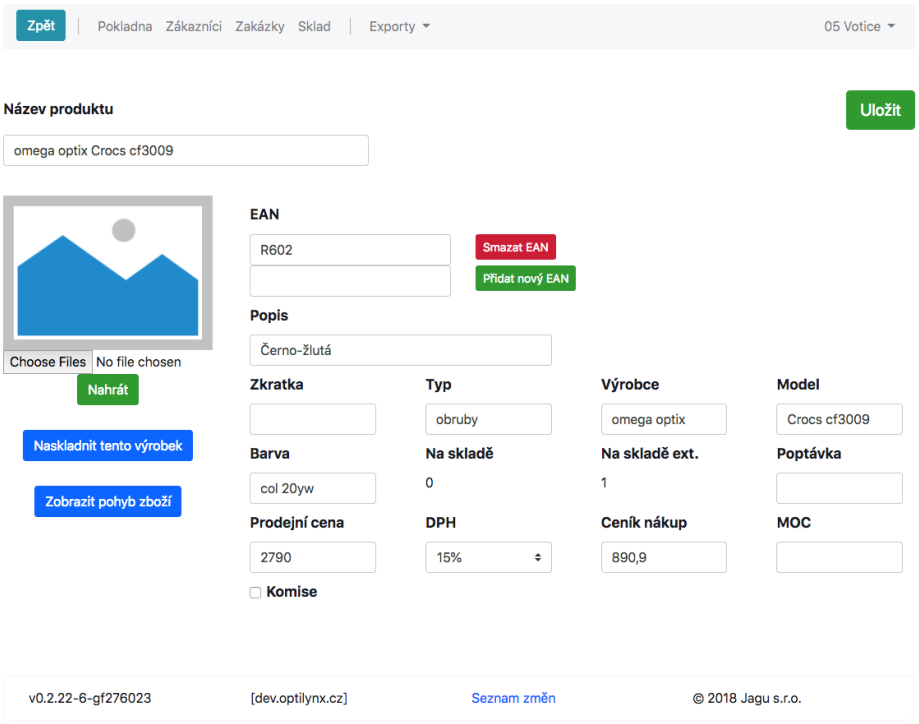
Obrázek B.22: Detail zákazníka



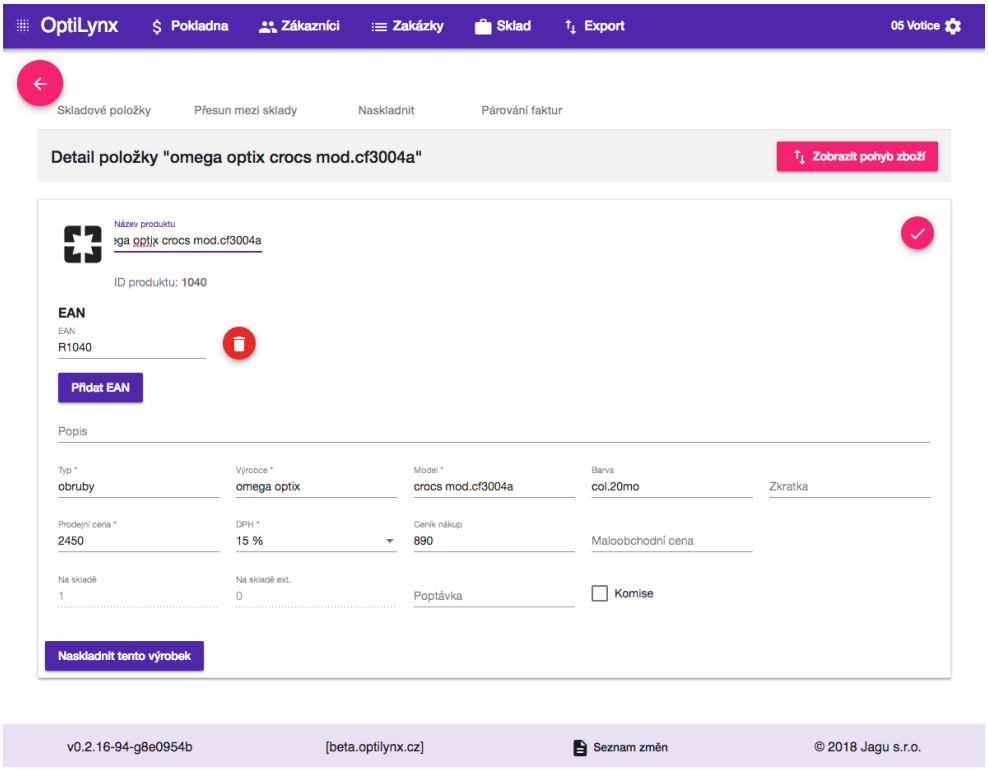
Obrázek B.23: Vložení nového zákazníka

Příloha C

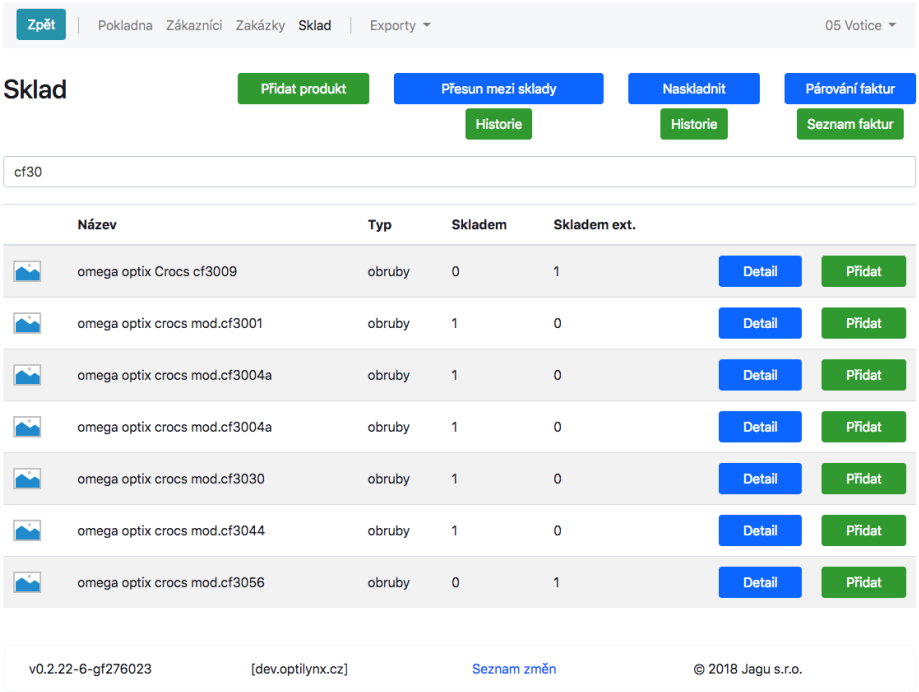
Ukázka FrontLynxu



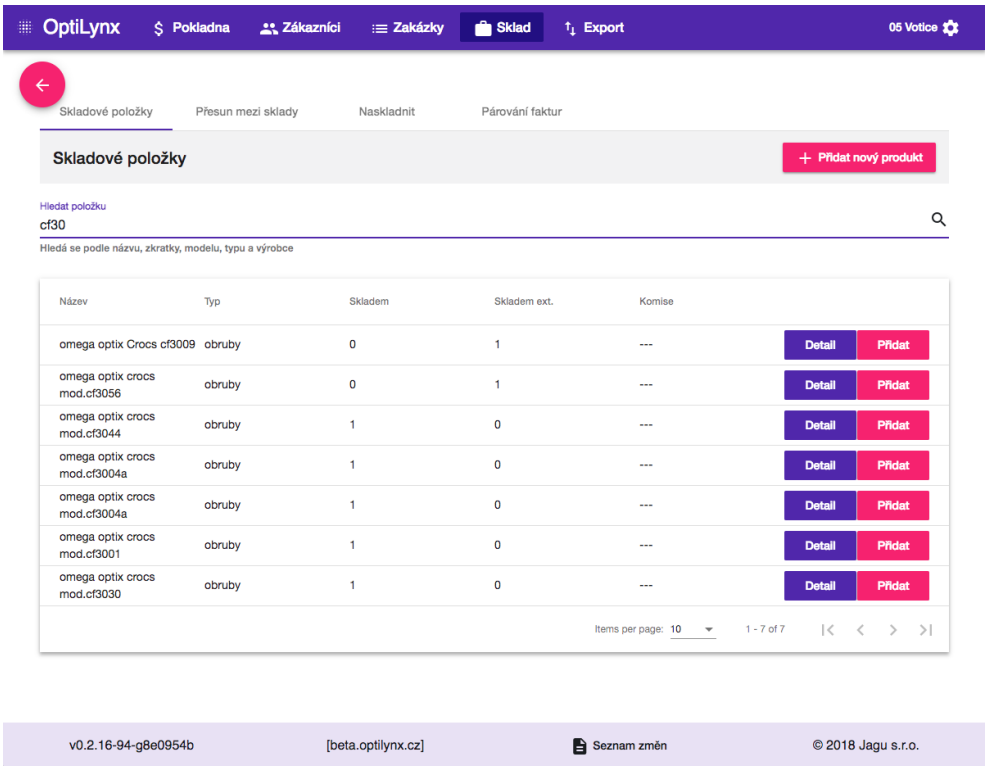
Obrázek C.1: Produkt - Bootstrap



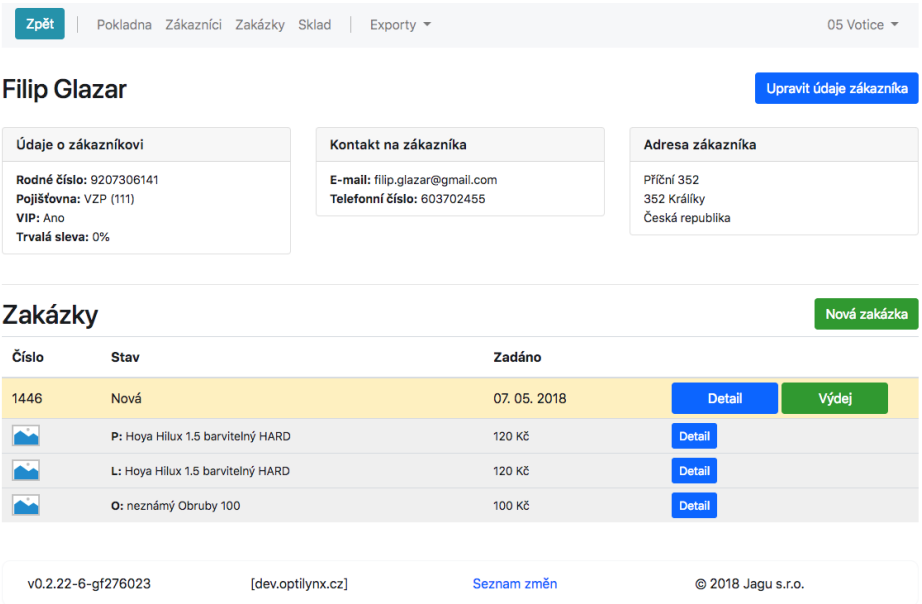
Obrázek C.2: Produkt - Material



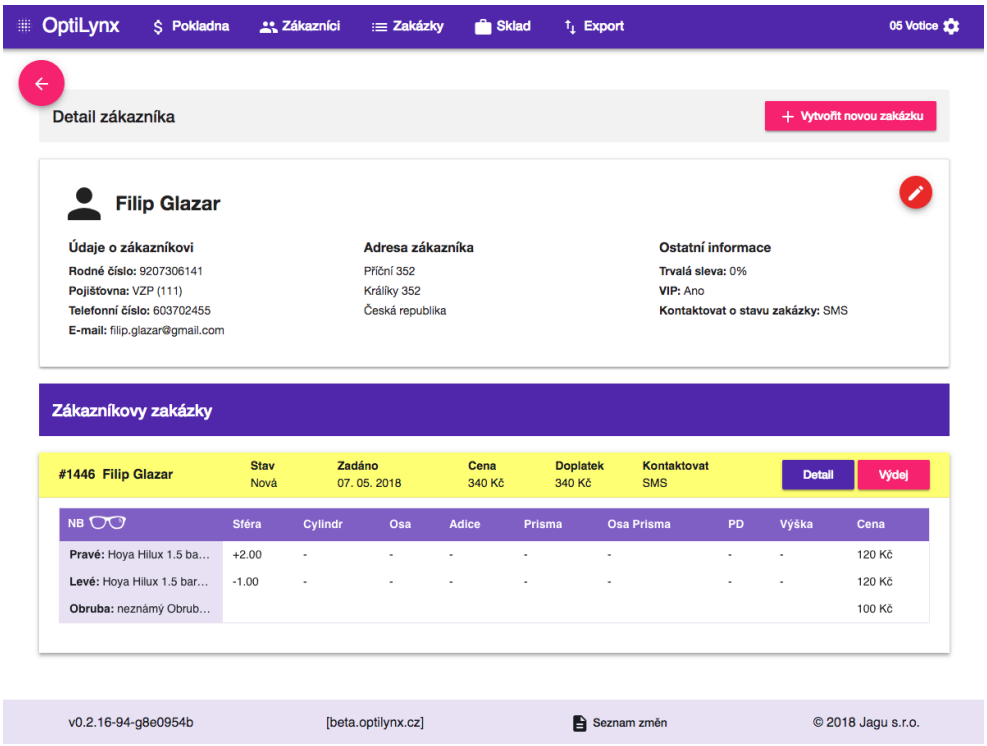
Obrázek C.3: Sklad - Bootstrap



Obrázek C.4: Sklad - Material



Obrázek C.5: Zákazník - Bootstrap



Obrázek C.6: Zákazník - Material

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Glazar_Filip_2018.pdf` ..... text práce ve formátu PDF
- `screenshots` ..... přílohy
  - `wireframes` ..... návrh uživatelského rozhraní
  - `comparsion` ..... srovnání verzí FrontLynxu
- `docs`
  - `swagger.txt` ..... odkaz na dokumentaci