FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Webová aplikace - Licence manager II
Student:	Tomáš Sládek
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Cílem této práce je vytvoření finální webové aplikace, která zajišťuje v rámci ekosystému Apps Manager komunikaci mezi vývojovou firmou a zákazníkem ohledně správy licencí na různé druhy softwaru. Aplikace bude jednou z komponent Apps Manageru, který se kromě této aplikace skládá také z Deployment Manageru a Builderu.

Postupujte v těchto krocích:
1. Důkladně analyzujte prototyp první verze Licence manageru, který předcházel vývoji Apps Manageru. Dále se zaměřte na konkrétní potřeby společnosti Jagu v souvislosti s touto prací a popište je.
2. Na základě analýzy vytvořte vhodný návrh aplikace, která bude komunikovat s ostatními částmi Apps Manageru, na kterých pracují v rámci bakalářských prací Alena Ježková a Adam Staš.
3. Na základě návrhu vytvořte prototyp aplikace.
4. Prototyp důkladně otestujte vhodnou sadou testů.
5. Na základě testů proveďte vhodné úpravy prototypu a jeho dokončení.
6. Zhodnoťte vámi dosažené výsledky a případně navrhněte možná budoucí vylepšení.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 18. ledna 2023 v Praze.

Bakalárska práca

WEBOVÁ APLIKACE - LICENCE MANAGER II

Tomáš Sládek

Fakulta informačních technologií
Katedra softwarového inženýřství
Vedúci: Ing. Jiří Huňnka
14. mája 2024

Obsah

Zoznam obrázkov

Zoznam výpisov kódu

Rád by som na prvom mieste poďakoval môjmu vedúcemu práce Ing. Jiřímu Hunkovi, za túto úžasnú príležitosť pracovať na tomto projekte. Ďalej by som rád poďakoval celému tímu, s ktorým sme v rámci predmetov BI-SP1 a BI-SP2 začali pracovať na tomto projekte.

Vyhlásenie

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, 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.

Abstrakt

Táto bakalárska práca sa zaoberá návrhom a implementáciou webovej aplikácie na nákup a správu softvérových produktov. Táto aplikácia je súčasťou širšieho ekosystému aplikácií, ktorý sa nazýva Apps Manager. Aplikácia sa skladá z frontendovej a backendovej časti. Frontendovú časť tvorí jednostránková moderná aplikácia imlementovaná vo frameworku Vue.js v jazyku JavaScript. Backendová časť je tvorená RESTful API, s ktorým následne komunikuje frontend. Backendová časť je implementovaná vo frameworku Laravel v jazyku php. Na záver je prevedené používateľské testovanie aplikácie.

Klíčová slova webová aplikácia, Vue.js, Laravel, PHP, JavaScript

Abstract

This bachelor thesis deals with the design and implementation of a web application for purchasing and managing software products. This application is part of a broader ecosystem of applications called Apps Manager. The application consists of a frontend and a backend. The frontend is a single-page modern application implemented in the Vue.js framework using JavaScript. The backend consists of a RESTful API, which communicates with the frontend. The backend is implemented in the Laravel framework using PHP. Finally, user testing of the application is conducted.

Keywords web application, Vue.js, Laravel, PHP, JavaScript

Zoznam skratiek

ACID	Atomicity, Consistency, Isolation and Durability
API	Application programming interface
ARES	Administrativní registr ekonomických subjektů
AWS	Amazon Web Services
BFF	Backend for Frontend
BI-SP1	Bakalársky predmet - Softvérový projekt 1
BI-SP2	Bakalársky predmet - Softvérový projekt 2
CRUD	Create Read Update Delete
CSRF	Cross-site request forgery
CSS	Cascading Style Sheets
ČVUT	České vysoké učení technické
FIT	Fakulta informačních technológií
FURPS	Functionality, Usability, Reliability, Performance and Supportability
GNU	GNU's Not Unix!
GPU	General Public License
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
HTML	HyperText Markup Language
I18n	Internationalization
IČO	Identifikačné číslo
IDE	Integrated Development Environment
IoT	Internet of Things
JSON	JavaScript Object Notation
JSX	JavaScript XML
MIT	Massachusetts Institute of Technology
MoSCoW	Must have, Should have, Could have and Won't have
MPA	Multi Page Application
MVC	Model View Controller
OAS	OpenAPI Specification
PHP	Personal Home Page
REST	Representational State Transfer
RPC	Remote Procedure Call
SaaS	Software as a Service
SDLC	Software development life cycle
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SPA	Single Page Application
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
UX	User Experience
W3C	World Wide Web Consortium
WS	Web Service
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSS	Cross Site Scripting

Úvod

S rozmachom výpočtovej techniky prišla prirodzená potreba zautomatizovať rôzne procesy vo firmách, vytvárať vlastné informačné systémy, ktoré by túto automatizáciu vykonali. Prostriedky na tvorbu vlastných informačných systémov dokázali alokovať len veľké spoločnosti, pre menšie spoločnosti by takéto riešenie, na výrobu vlastného systému, bolo nerentabilné. Tak vznikol na trhu priestor pre vývojárov na tvorbu informačných systémov použiteľných širokým spektrom menších firiem. Toto riešenie narazilo na problém, ako distribuovať takýto software medzi firmy čo najjednoduchšie, či už z pohľadu pohodlia s akým si zákazník takýto software zakúpi, tak aj z pohľadu ako vývojárov zbaviť procesu komplexného nasadzovania daného systému.

Cieľ práce

Cieľom tejto bakalárskej práce bolo vytvorenie finálnej verzie webovej aplikácie na predaj a následnú správu licencií, primárne pre firmu Jagu s. r. o., ktorá je súčasťou väčšieho celku Apps Manager. Pred samotnou tvorbou aplikácie bola prevedená analýza aktuálnych potrieb spoločnosti Jagu s. r. o. a už existujúcich riešení, ktoré sa venujú podobným problémom. Na základe analýzy bol vytvorený vhodný návrh aplikácie License Manager, v rámci systému Apps Manager. Po vytvorení návrhu, bol podľa neho vytvorený funkčný prototyp aplikácie. Funkčný prototyp aplikácie bol podrobený patričnému otestovaniu, aby bolo zachytených čo najviac nedostatkov, prehliadnutých počas tvorby a boli opravené do výslednej verzie aplikácie. Na koniec bolo predstavenie ďalších možností vývoja aplikácie s možnosťami budúceho rozvoja vylepšení.

Kapitola 1
Analýza problému

V tejto kapitole je prevedený výskum požiadaviek, procesov predávania a správou licencií na softvérové produkty s automatizovaným nasadzovaním. Jednotlivé požiadavky sú opísané v časti požiadavky 1.6.3. Následne sú analyzované aktuálne riešenia pre danú problematiku a to konkrétne License Manager od Viktora Holého a Builder, ktorý vznikol v rámci predmetov BI-SP1 a BI-SP2. Okrem riešení License Manager a Builder, ktoré vznikli v rámci prác na FIT ČVUT, budú zanalyzované aj systémy podobné, ktoré sú komerčne dostupné.

Tvorbe softvéru sa doposiaľ venovalo už veľmi mnoho ľudí. Ich vývoj sa častokrát stretol s problémami v rôznych častiach vývoja. Niektorí vývojári to nezvládli až do takej miery, že ich vývoj skončil neúspechom.

Zároveň ale množstvo projektov bolo úspešne zakončené. Spoločným úspešne ukončených projektov je kvalitne odvedená analýza problému na začiatku projektu. Kvalitná analýza poskytuje predpoklad na základe ktorého, môžeme robiť zásadné dizajnové rozhodnutia počas celého priebehu návrhu a implementácie.

Po nedôslednej analýze na začiatku projektu, pri samotnej práci, častokrát môžme len odhadovať kroky postupu v rámci vývoja a dúfať, že si zvolíme práve tú najvhodnejšiu postupnosť krokov v smerovaní vývoja projektu. Pri zlej voľbe našich krokov, sa v práci buď musíme vracať, alebo pri programovaní hľadať neštandardný spôsob ako obísť problém, čím často zanášame do programu rôzne zraniteľnosti, možnosti náhodných chýb. Oba spôsoby môžu viesť k strate času.

1.1 Súčasný stav

Prvým riešením predaju softvéru a správy licencií na softvér bola webová aplikácia License Manager, ktorá vznikla ako bakalárska práca Viktora Holého v roku 2021. Vstupom do tejto aplikácie je zo strany predajcu softvéru konfigurovateľná aplikácia, ktorej možnosti konfigurácie si v aplikácií nastaví. Následne si zákazník podľa možností konfigurácie, nakonfigurovanú aplikáciu objedná a automatizované zostavovacie prostredie ju pre neho zostaví, pripravenú na použitie. Táto webová aplikácia bude hlbšie analyzovaná v sekcii 1.1.1.

Na toto riešenie nadviazal vývoj zostavovacieho prostredia Builder v rámci výučby na FIT ČVUT v predmetoch BI-SP1 a BI-SP2. Builder sa primárne zameriaval na nasadzovanie už existujúcich softvérov od zadávateľa. V tomto smere využil skutočnosť, že tieto softvéry sú umiestnené na platforme GitLabe, ktorá umožňuje proces kontinuálneho vývoja a kontinuálneho nasadzovania pomocou svojich pipelines. Výsledná podoba prostredia Builder bola aplikácia napísaná v jazyku JavaScript za použitia prostredia node.js, ktorá spracovávala požiadavky od License Manageru a ukladala si ich do vlastnej databázy a volala GitLab pipelines na zostavenie

softvérov.

Počas vývoja zostavovacieho prostredia Builder sa začali črtať nové požiadavky riešiť problém nasadenia, konkrétne, keď sa softvér skladá z viacerých častí, treba postrážiť kompatibilitu verzií rôznych komponentov. Okrem toho si predávajúci vie určiť akú verziu chce nasadzovať pri novo vytvorených objednávkach, a vie určiť aj miesto kam sa majú nasadiť. Za týmto účelom vznikla bakalárska práca Adama Staše Deployment Manager. [1]

Na vývoji zostavovacieho prostredia Builder pokračovala Alena Ježková v rámci svojej bakalárskej práce. Počas ktorej spravila refactoring node.js aplikácie a implementovala komunikáciu s novo vznikajúcim komponentom Deployment Manager. [2]

Paralelne s vývojom zostavovacieho prostredia Builder vznikala nová verzia License Manageru v diplomovej práci Viktora Holého. Táto verzia sa zameriavala na prerobenie pôvodnej monolitickej verzie License Manageru, pomocou architektúry mikro služieb. V rámci tejto práce bola implementovaná backendová časť License Manageru pomocou mikro-služieb a rozhranie typu GraphQL na komunikáciu s frontendom.[3]

Na začiatku vývoja svojej bakalárskej práce som sa rozhodol vychádzať z monolitickej verzie aplikácie License Manager, ktorá bola vytvorená v rámci bakalárskej práce Viktora Holého, z dôvodu dobrého oboznámenia sa s monolitickou verziou počas vývoja v predmetoch BI-SP1 a BI-SP2 a z môjho rozhodnutia odlišnej koncepcie riešenia architektúry aplikácie oproti diplomovej práci Viktora Holého.

1.1.1 License Manager

Webovú aplikáciu License Manager môžme rozdeliť na dve časti, zákaznícku a administrátorskú. V rámci zákazníckej časti si zákazník môže prezrieť produkty, ktoré sú ponúkané. Ku každému produktu si vie zobraziť detaily o produkte a cenové plány, v ktorých je produkt ponúkaný. Zákazník má prehľad nad svojimi objednávkami a licenciami.

Pokiaľ chce zákazník pokračovať k objednaniu produktu, je už nutná registrácia, alebo prihlásenie už registrovaného zákazníka do systému.

Súčasťou procesu objednávky je vyplnenie konfigurácie, ktorá predstavuje sériu otázok na nastavenie produktu a následné vyplnenie fakturačných údajov, prípadne vybraním už existujúcich fakturačných údajov.

V rámci administrátorskej časti, môže administrátor vytvárať nové produkty, upravovať už existujúce produkty. Administrátorovi sú zobrazené všetky objednávky a už vytvorené licencie. Úlohou administrátora je aj spracovávať požiadavky, ohľadom zákazníkom definovaných cenových plánov. Jedna z ostatných vecí, ktoré sú ešte obsiahnuté v administrátorskej časti, je generátor a prehľad tokenov na prácu s License Manager API pre iné aplikácie, ktoré by ho chceli využívať.

Takmer celá aplikácia je písaná v jazyku PHP a využíva framework Laravel. Čo sa týka architektúry, tak ide o trojvrstvovú architektúru, ktorá využíva návrhový vzor MVC. Jediná časť, ktorá nie je písaná v jazyku PHP, je používateľské rozhranie na konfiguráciu produktu, ktoré je písané v jazyku JavaScript s využitím frameworku Vue.js.[4]

Analýzou kódu aplikácie, som zistil, že veľká časť kódu je znovu použiteľná na rozšírenie API, alebo prevzatie do frontendu. Z tejto mojej analýzy som vychádzal v ďalšej mojej práci

1.1.2 Pripomienky z používania v rámci BI-SP1/2

Počas vývoja v rámci predmetov BI-SP1/2 došlo k značnému využívaniu License Manageru, pri integračnom testovaní a pri prvotnej identifikácií potrebnej komunikácie medzi Builderom a License Managerom. Počas tohto používania bolo identifikovaných niekoľko neprívetivých používateľských skúseností, najmä s tvorbou produktu v rámci administrátorskej sekcie.

Medzi hlavné problémy patrí takmer neexistujúca spätná väzba, čo spôsobuje potiaže, najmä pokiaľ má používateľ minimálne znalosti s používaním aplikácie. Napríklad tvorba produktu sa stáva zdĺhavou činnosťou, keď používateľ chce doladiť výzor stránky produktu. Na overenie

si výzoru stránky produktu je nutné produkt zverejniť, čím sa stáva nie len viditeľný pre všetkých zákazníkov, ale zároveň aj nemenným. Nemennosť produktu vytvára problémy pre ďalšie úpravy. V prípade ďalších úprav je potrebné vytvoriť kópiu produktu, ktorá je ešte nezverejnená a doladiť detaily v nej.

Ďalším nedostatkom je absencia možnosti upravovať, prípadne zmazať niektoré položky, či už pri tvorbe produktu, alebo v rámci používania aplikácie zákazníkom. Pri tvorbe produktu toto vedie k zmazaniu a nanovo vytváraniu niektorých častí, kvôli drobným úpravám.

Ostatným identifikovaným nedostatkom je jednosmerný postup pri tvorbe objednávky, bez možnosti návratu, opravy, prípadne celkového zrušenia objednávky.

1.2 Analýza konkurencie

V tejto časti som zisťoval konkurenciu k License Manageru, s cieľom identifikovať funkcie a procesy, ktorými by stálo za úvahu sa inšpirovať. Hľadanie som realizoval vyhľadávaním kľúčových slov týkajúcich sa App Manageru v anglickom jazyku, na internetových vyhľadávačoch. Podarilo sa mi takto vyhľadať dve kategórie produktov s názvom Licesne Manager.

Prvá kategória produktov obsahuje aplikácie, ktoré považujú používateľa ako koncového zákazníka licencií a zobrazujú mu všetky licencie ktoré on využíva. Takto definovaný prípad použitia sa líši od zamýšlaného prípadu použitia pre moju bakalársku prácu.

Druhá kategória produktov obsahuje aplikácie, ktoré považujú používateľa ako predajcu softvéru, ktorý vydáva licencie na svoj softvér. Produkty v tejto kategórií sa zaoberajú podobnou tématikou ako je téma mojej bakalárskej práce. Medzi také patria AWS License Manager s funkciou Seller Issued Licenses a 10Duke's software license management. Tento výber obsahuje len niektoré z nájdených, ktoré boli silno pozitívne hodnotené používateľmi.[5]

1.2.1 AWS License Manager

AWS License Manager je jeden z mnohých produktov, ktoré sú obsiahnuté v rámci AWS. Jeho primárnou úlohou je správa licencií pre produkty tretích strán, ktoré bežia v rámci AWS a je vyžadovaná na ne licencia. License Manager stráži správne využívanie licencie, aby nedošlo k prekročeniu licencie. Funkčnosť stráženia licencií na produkty tretích strán, je však iná ako je požadovaná funkčnosť systému od zadávateľa. Okrem tejto funkčnosti však AWS License Manager obsahuje aj vydávanie licencií pre koncových zákazníkov, čo je požadovaná funkčnosť od zadávateľa. Medzi podporované druhy takto vydávaných licencií patrí:

- Večné (Perpetual) - Licencie, ktoré sú vydané bez konca platnosti a autorizujú používateľa na využívanie softvéru na dobu neurčitú.
- Flexibilné (Floating) - Zdieľané licencie s viacerými inštanciami aplikácií. V takomto prípade môžu byť licencie predplatené s fixným súborom obmedzení.
- Predplatné (Subscription) - Licencie s koncom platnosti, ktoré môžu byť automaticky predĺžené, pokiaľ nebudú implicitne zrušené.
- Na základe spotreby (Usage-based) - Licencie so špecifickými požiadavkami na spotrebu, požiadavky môžu byť napríklad počet požiadaviek na API, transakcií, alebo možnosti úložiska.

Vydávanie licencií je primárne prispôsobené pre používateľov, ktorí tiež majú účet na AWS, a licencia sa napojí na ID ich účtu. Pokiaľ používateľ nemá účet na AWS, je možné mu vydať licenciu pomocou dočasného mandátu, kde si vygeneruje autentizačný token. Po vygenerovaní licencie sa môže licencia nachádzať v jednom z týchto stavov, vytvorená (created), aktualizovaná (updated), deaktivovaná (deactivated), alebo zmazaná (deleted).[6]

Hlavnými nedostatkami tohto riešenia sú chýbajúca možnosť konfiguračného dotazníka pre zákazníka, v ktorom si svoj produkt nakonfiguruje podľa svojej potreby a chýbajúce napojenie na zostavovacie prostredie.

1.2.2 10Duke's software license management

10Duke ponúka celú škálu produktov venujúcich sa správe používateľov. 10Duke's software license management je súčasťou tohto balíčka a obsahuje skôr backendovú podporu na kontrolu licencií. Funguje princípom delegovania overenia používateľa z aplikácie, na ktorú je vystavená licencia na 10Duke. Počas overenia na 10Duke sa skontroluje, či sa používateľ prihlasuje správnymi prihlasovacími údajmi a či má patričnú licenciu.

Okrem backendovej časti obsahuje aj používateľské rozhranie pre administrátorov a zákazníkov. Pre zákazníkov obsahuje prihlasovací komponent, ktorý si môže správca implementovať do svojej aplikácie. Tento komponent obsahuje prihlasovaciu stránku a stránku profilu. Na stránke profilu vie následne zákazník spravovať svoje osobné údaje, zmeniť alebo obnoviť heslo a zapnúť dvoj-faktorové overenie.

Administrátorská časť slúži na zadávanie informácií o produktoch, na ktoré sú vydávané licencie. Pri produkte sú uvedené informácie o produkte, rezdelenie na zložky, to znamená, či je rozdelený na menšie komponenty, ktorých kombinácie by tvorili licenciu. Keď už je definovaný produkt, tak sa zvolia licenčné druhy, ktorými je produkt licencovaný, či je to licencia navždy, alebo na mesačnej báze, prípadne takzvaná floating licencia. V rámci administrátorskej časti ešte existuje detailný prehľad využívania licencií zákazníkmi.

Celé riešenie je nasadené v cloude a potrebuje pripojenie k internetu na overenie licencie.[7]

Tento nástroj neobsahuje verziu zadarmo, ktorú by bolo možné si vyskúšať a dalo by sa s ňou lepšie oboznámiť. Hlavným rozdielom so zadaním práce je chýbajúca komerčná časť, kde sa produkt predáva zákazníkovi a s tým spojená počiatočná konfigurácia produktu zákazníkom.

1.3 Licencia na softvér

Softvérová licencia je dokument, ktorý poskytuje právne záväzné pokyny na používanie a distribúciu softvéru.

Softvérové licencie zvyčajne poskytujú koncovým používateľom právo na jednu alebo viac kópií softvéru bez porušenia autorských práv. Licencia tiež vymedzuje povinnosti strán uzatvárajúcich licenčnú zmluvu a môže obmedzovať spôsob použitia softvéru.

Licenčné podmienky softvéru obvykle zahŕňajú spravodlivé používanie softvéru, obmedzenia zodpovednosti a záruky. Taktiež špecifikujú ochranu v prípade, že softvér alebo jeho použitie poruší duševné vlastníctvo ostatných.

Softvérové licencie sú zvyčajne proprietárne, bezplatné alebo open source. Charakteristickým znakom sú podmienky, podľa ktorých môžu používatelia redistribuovať alebo kopírovať softvér pre budúci vývoj alebo použitie.[8]

Táto bakalárska práca sa zaoberá len licenciami proprietárnymi, takže licencie zakazujú akékoľvek ďalšie šírenie softvéru.

1.4 Softvér ako služba - SaaS

Softvér ako služba je spôsob doručovania aplikácií ako služba cez internet. Namiesto inštalácie a údržby softvéru k nemu jednoducho pristupujete cez internet, čím sa oslobodíte od zložitej správy softvéru a hardvéru.

Aplikácie SaaS sa niekedy nazývajú: webový softvér, softvér na požiadanie alebo hosťovaný softvér. Bez ohľadu na názov sa aplikácie SaaS spúšťajú na serveroch poskytovateľa SaaS. Poskytovateľ spravuje prístup k aplikácii, vrátane zabezpečenia, dostupnosti a výkonu.[9]

1.4.1 Modely softvérových licencií

Modely softvérových licencií predstavujú stratégie prijaté softvérovými spoločnosťami na distribúciu a speňaženie ich softvéru. Tieto modely definujú, ako môžu používatelia pristupovať k softvéru, trvanie používania a práva, ktoré im boli udelené.

Výber licenčného modelu výrazne ovplyvňuje príjmy spoločnosti a celkový úspech na trhu.

Existuje mnoho rôznych modelov softvérových licencii, z ktorých každý má svoje výhody a obmedzenia. Najlepší model závisí od konkrétnych potrieb a požiadaviek.

Model trvalej licencie je tradičný prístup, pri ktorom si používatelia zakúpia jednorazovú licenciu na používanie softvéru na dobu neurčitú. To znamená, že používatelia môžu pokračovať v používaní softvéru aj po skončení obdobia podpory a aktualizácií.

Zákazníkovi poskytuje flexibilitu používať softvér tak dlho, ako ho potrebuje. Model s trvalou licenciou má však aj svoje nevýhody. Najväčšou nevýhodou je obtiaž sledovania počtu používateľov, ktorí používajú softvér, čo môže sťažiť riadenie súladu s licenciami.

Podnikové licencovanie je model licencovania softvéru, ktorý je prispôsobený veľkým organizáciám, ktoré vyžadujú, aby k softvéru pristupovalo súčasne viacero používateľov. Namiesto individuálneho licencovania každého používateľa ponúka tento model cenovo efektívne riešenie, kde jedna licencia pokrýva viacerých používateľov v rámci organizácie, zvyčajne na základe počtu zakúpených miest.

Podnikové licencovanie má niekoľko výhod. Môže to byť nákladovo efektívnejšie ako licencovanie na používateľa, najmä pre organizácie s veľkým počtom používateľov. Správa môže byť jednoduchšia, keďže nie je potrebné sledovať jednotlivé užívateľské licencie. Vyjednávanie a implementácia však môže byť zložitejšia ako pri iných licenčných modeloch.

Licencovanie na základe predplatného si v posledných rokoch získal popularitu. Je to pravdepodobne najpopulárnejší licenčný model. Za prístup k softvéru je potrebné platiť pravidelný poplatok, zvyčajne na mesačnej, alebo ročnej báze.

Tento model zabezpečuje nepretržité príjmy pre softvérovú spoločnosť a často zahŕňa podporu a aktualizácie počas obdobia predplatného. Jednou z hlavných výhod predplatených licencií je to, že ich možno vykonať s malým rozpočtom bez obetovania kvality a bezpečnosti.

Model pohyblivej licencie , tiež známy ako súbežné licencovanie, umožňuje obmedzenému počtu používateľov pristupovať k softvéru súčasne. Celkový počet zakúpených licencií určuje maximálny počet používateľov, ktorí môžu softvér používať súčasne. Tento model je ideálny pre organizácie s veľkou používateľskou základňou, ktorá vyžaduje občasný prístup k softvéru.

Licencovanie podľa funkcií je založené na funkciách softvéru rozdelených do samostatných modulov, alebo funkcií. Používatelia si môžu vybrať funkcie, ktoré potrebujú a podľa toho za ne zaplatiť.

Tento model ponúka majiteľom firiem flexibilitu platiť za funkcie, ktoré potrebujú, a zároveň umožňuje softvérovej spoločnosti uspokojiť rôzne potreby zákazníkov.

Licenčné modely proprietárneho softvéru je typ licenčného modelu softvéru, v ktorom si dodávateľ softvéru ponecháva vlastníctvo zdrojového kódu softvéru. To znamená, že používatelia softvéru nemajú povolené upravovať, distribuovať, alebo spätne analyzovať softvér.

Je to bežný licenčný model v komerčných softvérových produktoch. Umožňuje to predajcovi softvéru chrániť svoje práva duševného vlastníctva obmedzením prístupu k zdrojovému kódu.

Licencie proprietárneho softvéru môžu mať určité výhody. Napríklad, môžu pomôcť zabezpečiť kvalitu a bezpečnosť softvéru.

Model licencovania s otvoreným zdrojom (open-source) je typ softvérovej licencie, ktorý používateľom umožňuje prístup k zdrojovému kódu softvéru. To znamená, že softvér je možné používať a upravovať, pokiaľ sú dodržané licenčné podmienky.

Medzi najpopulárnejšie open-source licencie patrí GNU General Public License (GPL), Apache License a MIT License.

Licencovanie uzlom je model softvérového licencovania, v ktorom je licencia na softvérovú aplikáciu priradená jednému, alebo viacerým hardvérovým zariadeniam, ako je počítač, mobilné zariadenie, alebo zariadenie IoT. Pre takéto licencie je zvyčajne povolený akýkoľvek počet inštancií.

Túto formu licencovania používajú vydavatelia softvéru, aby zabezpečili, že licencia bude spustená iba na konkrétnych hardvérových zariadeniach. Každý uzol je identifikovaný jedinečným odtlačkom zariadenia, ktorý je potrebné získať alebo zadať počas nastavenia produktu, alebo počas prvého overovania licencie.

Licencovanie s uzlom je nákladovo efektívne riešenie pre scenáre pre jedného používateľa. Je to tiež dobrá voľba pre softvér, ktorý sa používa na špecializovanom hardvéri, ako sú lekárske prístroje, alebo vedecké zariadenia. Licencovanie s blokovaním uzlov však môže byť obmedzujúce pri potrebe presunúť softvér na iné zariadenie.

Licenčný model založený na meraní alebo použití je model licencovania softvéru, v ktorom používatelia platia na základe skutočného používania softvéru. Tento model sa často používa pre cloudové služby, alebo produkty typu softvér ako služba (SaaS), kde poplatky určujú metriky ako využitie dát, počet transakcií, alebo aktívni používatelia.

Existuje niekoľko výhod licencovania na základe merania, alebo používania, ktoré zahŕňajú nižšie náklady, pretože používatelia platia iba za funkcie, ktoré skutočne používajú. Môže pomôcť sledovať vzory používania a identifikovať oblasti, kde je možné softvér používať efektívnejšie. Odhadnúť náklady vopred však môže byť náročné, čo sťažuje rozpočet.

Licenčný model založený na lokalite je model licencovania softvéru, v ktorom sa licencia udeľuje pre celé miesto, alebo lokalitu. To znamená, že k softvéru môžu pristupovať viacerí používatelia v rámci tejto lokality bez ohľadu na počet zariadení, ktoré používajú. Je to nákladovo efektívny prístup pre organizácie s viacerými používateľmi na centralizovanom mieste.

Niektoré z jeho výhod sú flexibilita a jednoduchosť. Licencovanie na základe lokality umožňuje organizáciám flexibilne upravovať počet potrebných licencií podľa toho, ako sa menia ich potreby. Ich správa je jednoduchšia ako licencovanie podľa jednotlivých používateľov, pretože nie je potrebné sledovať jednotlivých používateľov, alebo zariadenia. Nie je však škálovateľný.

Model skúšobnej licencie ponúka časovo obmedzenú bezplatnú skúšobnú verziu softvéru pred zakúpením plnej licencie. Ide o marketingovú stratégiu na predvedenie schopností softvéru a prilákanie potenciálnych zákazníkov. Na druhej strane umožňuje potenciálnym zákazníkom vyskúšať si softvér pred jeho kúpou, čo im môže pomôcť urobiť informované rozhodnutie o tom, či je softvér pre nich vhodný alebo nie.

Licenčný model založený na zariadení je model softvérovej licencie, ktorý umožňuje inštaláciu a používanie softvéru na konkrétnom zariadení bez ohľadu na počet používateľov, ktorí k nemu pristupujú. Bežne sa používa pre softvér, ktorý funguje na špecializovanom hardvéri, alebo zariadení.

Je ľahké ho spravovať, pretože nie je potrebné sledovať jednotlivých používateľov, alebo licencie a môže byť nákladovo efektívnejšie ako licencovanie na jedného používateľa, najmä pre organizácie s veľkým počtom zariadení. Môže tiež pomôcť zabrániť neoprávnenému prístupu k softvéru.

[10]

1.5 Popis domény

Popis domény slúži na presnejšiu definíciu pojmov, ktoré sa vyskytujú v riešenej problematike. V nasledujúcich podkapitolách sa venujem definíciám jednotlivých pojmov domény.

1.5.1 Zákazník

Osoba, ktorá príde na web stránku so záujmom o prezretie ponúkaných produktov, prípadne aj za účelom následnej kúpi produktov, alebo osoba ktorá už má zakúpený produkt. Na kúpu musí byť zákazník zaregistrovaný. Zákazník má na danej web stránke prehľad o svojich objednávkach, faktúrach a licenciách.

1.5.2 Produkt

Softvér, na ktorý si zákazník môže zakúpiť licenciu na používanie. Produkt môže byť ponúkaný v rôznych cenových plánoch, z ktorých si zákazník môže vybrať. Produkt sa skladá z jednej a viac aplikácií (webových, inštalovateľných).

1.5.3 Cenový plán produktu

Cenový plán predstavuje nastavenie produktu, napríklad požadovaným rozsahom, ktorý má produkt pokryť. Ďalšie možnosti nastavenia sú z hľadiska spôsobu financovania (vyšší iniciálny záväzok s nižšími priebežnými platbami, alebo nulová počiatočná platba s primerane nacenenými priebežnými platbami, alebo iné riešenie). Pokiaľ zákazníkovi nevyhovuje ani jeden z ponúkaných cenových plánov, z hľadiska rozsahu, môže si zažiadať o nacenenie vlastného cenového plánu.

1.5.4 Konfigurácia produktu

Každý produkt má vlastnú konfiguráciu, ktorá sa zakladá na súbore otázok nastavených v rámci administratívnej časti License Manageru. Konfiguráciu musí zákazník vyplniť v procese vytvárania objednávky. Na základe konfigurácie sa mu nastaví produkt v procese zostavovania.

1.5.5 Objednávka

Objednávku vytvára zákazník (alebo administrátor), na vybraný produkt z ponuky. V objednávke sa vyplní konfigurácia produktu a fakturačné údaje. Po vytvorení objednávky sa vytvorí faktúra s fakturačnými údajmi zadanými počas vytvárania objednávky. Po zaplatení počiatočnej faktúry sa produkt, ku ktorému bola vytvorené objednávka, začne zostavovať.

1.5.6 Licencia

Licencia sa vždy viaže ku konkrétnej objednávke a vzniká po úspešnom zostavení produktu pre danú objednávku. Obsahuje dobu platnosti licencie, prehľad faktúr vystavených na danú licenciu a prístup k aplikáciam produktu. Prístup k aplikáciam môže mať rôznu podobu. Môže to byť odkaz na webovú stránku, kde beží daná aplikácia, alebo odkaz na stiahnutie inštalovateľného balíka aplikácie, alebo ako zobrazenie prihlasovacích údajov do danej aplikácie.

1.5.7 Administrátor

Osoba, ktorá spravuje produkty a spracováva požiadavky na nacenenie zákazníkom definovaného cenového plánu produktu. S produktami môže robiť nasledovné operácie: vytváranie nových produktov, úpravu už existujúcich produktov a rušenie starých produktov.

1.6 Zber požiadaviek

Zber požiadaviek predstavuje základný kameň pri tvorbe akéhokoľvek softvéru. Požiadavky kladú na systém obmedzenia a vytyčujú jeho výslednú podobu. Presnejšie tieto požiadavky čo najpresnejšie definujú ako by systém mal a ako by nemal vyzerať.

Zber požiadaviek sa zakladá na komunikácií medzi zákazníkom, budúcim používateľom a analytikom, ktorý tieto požiadavky zbiera a snaží sa s druhou stranou dohodnúť, ktoré požiadavky budú a ktoré nebudú obsiahnuté vo vyvíjanom softvéri.

Zber požiadaviek môžeme realizovať buď kvantitatívne, získavaním odpovedí na otázky od veľkého množstva používateľov, napríklad pomocou dotazníkov, ktoré sú následne štatisticky spracovávané, alebo kvalitatívne, získavaním požiadaviek pomocou hlbších rozhovorov s používateľom. [11] V rámci svojej práce som realizoval kvantitatívny zber požiadaviek analýzou požiadaviek na pôvodnú verziu License Manageru, ktoré som buď prebral úplne, alebo rozdelil na viacero požiadaviek. Kvalitatívny zber požiadaviek som realizoval v rámci práci s License Managerom počas predmetov BI-SP1 a BI-SP2. Ďalšie požiadavky som získal pomocou rozhovorov s Ing. Jiřím Hunkou a Ing. Oldou Malcem.

Niektoré požiadavky sú pri vývoji neobsiahnuté z rôznych dôvodov. Medzi ktoré najčastejšie patria: protichodnosť niektorých požiadaviek, v tom prípade sa dá vyhovieť iba jednej z možností. Alebo zahrnutie neobsiahnutej požiadavky by značne predĺžilo čas na vývoj softvéru, alebo predražilo vývoj nad rámec finančných možností zákazníka.

Požiadavky sú rôznych druhov. Jedným z druhov sú funkčné požiadavky, ktoré obmedzujú funkcionalitu softvéru. To v jednoduchosti znamená, že definujú, čo systém vie a čo nevie spraviť.

Ďalším druhom sú nefunkčné požiadavky, tieto obmedzujú všetky obmedzenia, ktoré sa netýkajú funkcionality systému. Medzi príklady na nefunkčné požiadavky patrí to, ako má systém vyzerať, aké zariadenia má podporovať, alebo záťaž, ktorú má vydržať.

Aby sme si udržali prehľad o všetkých požiadavkách, ktoré sme dostali na vývoj nášho systému, je dobré nejako kategorizovať tieto požiadavky do logických kategórií. Na kategorizáciu požiadaviek existujú primárne dva hlavné spôsoby kategorizácie požiadaviek. Prvým spôsobom je FURPS+ a druhým je MoSCoW, ktoré si v nasledujúcich pasážach priblížime.

1.6.1 FURPS+

Kategorizácia požiadaviek FURPS sa venuje primárne rozdeleniu na požiadavky funkčné a nefunkčné. Nefunkčné ďalej delí na použiteľnosť, spoľahlivosť, výkon a udržateľnosť. FURPS je následne rozširiteľný do FURPS+ špecifikovaním obmedzení na systém. Kategórie obmedzení, ktoré takto môžme na systém stanoviť sú návrhové, implementačné, fyzické a komunikačné. Kategórie FURPS+ znamenajú:

Funkčnosť (Functionality) sú požiadavky, ktoré vymedzujú jednotlivé funkcionality softvéru.

Použiteľnosť (Usability) sú požiadavky, ktoré vymedzujú kto a ako bude vyvíjaný softvér používať, zhodnotenie celkového dojmu, popis dokumentácie a používateľských príručiek.

Spoľahlivosť (Reliability) sú požiadavky, ktoré vymedzujú chybovosť softvéru, tieto požiadavky vymedzujú koľko chýb a ich frekvencia, je ešte akceptovateľná v softvéri.

Výkon (Performance) sú požiadavky, ktoré vymedzujú výkon softvéru, čo znamená koľko bude trvať spracovanie vstupu o danej veľkosti, alebo požiadavky na hardvér na ktorom bude softvér fungovať.

Udržateľnosť (Supportabilty) sú požiadavky, ktoré vymedzujú údržbu a podporovateľnosť softvéru, ako aj možnosti na ďalšie rozšírenia softvéru.

Návrhové obmedzenia nám určujú aké technológie môžme použiť počas návrhu systému. Napríklad obmedzenie na využitie relačnej databázy nám určuje následný postup pri ďalšom návrhu systému.

Implementačné obmedzenia nám určujú priebeh implementácie. Určujú nám aký programovací jazyk sa má použiť, a aké štandardy sa majú dodržiavať.

Fyzické obmedzenia sú určované hardvérom na ktorom bude systém fungovať.

Komunikačné obmedzenia nám určujú externé systémy s ktorými je potrebné komunikovať v našom systéme.

[12]

1.6.2 MoSCoW

Kategorizácia požiadaviek MoSCoW rozdeľuje požiadavky podľa priorít, s ktorými by mali byť dané požiadavky obsiahnuté vo výslednom softvéri. Toto rozdelenie je nasledovné:

Musí obsahovať (Must have) sú požiadavky, ktoré softvér musí obsahovať a bez nich nemôže byť kompletný.

Mal by obsahovať (Should have) sú požiadavky, ktoré by softvér mal obsahovať, ale zároveň ich prípadná absencia nenaruší celý softvér a dokáže existovať aj bez nich.

Mohol by obsahovať (Could have) sú požiadavky, ktoré by softvér mohol obsahovať. Sú to často veci s veľmi nízkou prioritou, ktoré boli vymyslené ako príjemné zlepšenie, ale takmer nijako nezasahujú do základného fungovania softvéru. Často sa realizujú dodatočným vývojom.

Nebude obsahovať (Won't have) sú požiadavky, ktoré naopak nebudú v softvéri implementované a obmedzujú rozsah projektu.

[13]

1.6.3 Požiadavky

Čo sa týka požiadaviek na License Manager, tak takmer všetky pôvodne z práce Viktora Holého zostávajú zachované[4], až na komunikáciu s Buildrom, ktorú nahradí komunikácia s Deloyment Mangerom. Okrem komunikácie s Deployment Managerom pribudli aj ďalšie nové požiadavky. Všetky požiadavky sú vypísané nižšie.

Pri požiadavkách som sa rozhodol skombinovať FURPS a MoSCoW spôsoby, ku ktorým som ešte pridal číselnú prioritu a náročnosť implementácie požiadaviek. Podľa číselnej priority budem postupne implementovať požiadavky vo vzostupnom poradí, t.j. 1 je najvyššia priorita. Požiadavky s nižšími prioritami budú tvoriť možný budúci vývoj projektu.

Náročnosť implementácie bude slúžiť ako doplnok ku číselnej priorite a požiadavky, ktoré budú mať nižšiu náročnosť, budú uprednostňované v rámci rovnakej priority.

Všetky požiadavky popisujem v následnej štruktúre,

- Stručný názov požiadavku
- Podrobnejší popis požiadavku
- Kategória podľa FURPS
- Kategória podľa MoSCoW
- Číselná priorita
- Náročnosť

V nasledujúcej sekcií je zoznam všetkých požiadaviek na aplikáciu.

P1: Objednávka produktov

- Systém umožní zákazníkovi vytvoriť objednávku na cenový plán produktu. Následne v rámci tejto objednávky vyplniť konfiguráciu produktu a vybrať si fakturačné údaje. Po potvrdení objednávky pri rekapitulácií, systém vytvorí objednávku a pošle zákazníkovi počiatočnú faktúru.
- Funkčnosť
- Musí obsahovať (Must have)
- Priorita 1
- Stredne náročné

P2: Zákazníkom definovaný cenový plán

- Systém umožní produktu povoliť vlastné zákazníkom definované cenové plány. Pokiaľ bude táto možnosť produktu povolená, zákazník, ktorému nevyhovuje ani jeden z ponúkaných cenových plánov, si môže špecifikovať vlastné požiadavky v rámci zákazníkom definovaného cenového plánu. Takáto požiadavka je následne predaná administrátorovi na nacenenie daného cenového plánu. Na koniec ma zákazník na výber, či danú ponuku prijme, alebo zamietne.
- Funkčnosť
- Mal by obsahovať (Should have)
- Priorita 3
- Stredne náročná

P3: Správa objednávok a licencií

- Zákazník bude môcť v rámci systému spravovať svoje zakúpene objednávky a k nim prislúchajúce licencie. Ku každej objednávke, alebo licencii budú dostupné nasledovné operácie: zrušenie objednávky/licencie, zobrazenie aktuálnych informácii, ohľadom objednávky, alebo licencie, zobrazenie faktúr vystavených pre danú objednávku, alebolicenciu, zobrazenie zostavených produktov k licencii, navýšenie licencie a zníženie licencie.
- Funkčnosť
- Mal by obsahovať (Should have)
- Priorita 1
- Náročná

P4: Stráženie platnosti licencií

- Systém bude na základe zaplatených faktúr predlžovať platnosť licencií. Systém licencie s nezaplatenými pohľadávkami zablokuje. Blokácia bude mať dve úrovne: odstavenie (zrušenie prístupu) aplikácie a zmazanie aplikácie vrátane zákazníckych dát.
- Funkčnosť
- Musí obsahovať (Must have)
- Priorita 2
- Stredne náročná

P5: Fakturácia

- Systém zaistí vystavovanie faktúr pomocou softvéru Fakturoid. Systém si bude overovať stav faktúr pomocou tohto softvéru.
- Funkčnosť
- Musí obsahovať (Must have)
- Priorita 2
- Menej náročná

P6: Používateľské rozhranie

- Systém bude možné ovládať z grafického používateľského prostredia.
- Funkčnosť
- Musí obsahovať (Must have)
- Priorita 1
- Náročná

P7: Zostavovanie produktov

- Systém bude komunikovať zostavenie objednávok s ďalšími komponentmi systému Apps Manager.
- Funkčnosť
- Mal by obsahovať (Should have)
- Priorita 2
- Stredne náročná

P8: Používateľské konto

- Aplikácia bude podporovať používateľské účty a spolu s nimi príslušné operácie: registrácia, prihlásenie, odhlásenie, obnovenie zabudnutého hesla, zmenu hesla. Konto môže byť typu zákazník, alebo administrátor.
- Funkčnosť
- Musí obsahovať (Must have)
- Priorita 1
- Menej náročná

P9: Prehľad produktov

- Aplikácia umožní zákazníkovi prehľad o všetkých zverejnených produktoch, každý produkt bude mať tlačidlo na kúpu toho produktu a tlačidlo na zobrazenie podrobných informácií o danom produkte.

- Funkčnosť
- Musí obsahovať (Must have)
- Priorita 1
- Menej náročná

P10: Manuálne zostavenie objednávky

- Systém umožní označiť objednávku ako zostavenú aj manuálne z administrátorského prostredia.
- Funkčnosť
- Mohol by obsahovať (Could have)
- Priorita 4
- Menej náročná

P11: Prihlásenie pomocou sociálnych sietí

- Systém umožní registráciu a prihlásenie pomocou sociálnych sieti.
- Funkčnosť
- Mohol by obsahovať (Could have)
- Priorita 7
- Náročná

P12: Doplnenie zostavenej licencie

- Systém umožní manuálne doplniť zostavené produkty k už existujúcej licencii. Okrem zostavených produktov, pôjde pridať aj prístupové údaje k nim.
- Funkčnosť
- Mohol by obsahovať (Could have)
- Priorita 4
- Stredne náročná

P13: Responzívny dizajn grafického rozhrania

- Dizajn grafického rozhrania sa bude responzívne prispôsobovať veľkosti obrazovky, na ktorej je zobrazený.
- Použiteľnosť
- Mohol by obsahovať (Could have)
- Priorita 3
- Náročná

P14: Monitoring systému

- Systém bude napojený na monitoring pomocou softvéru Sentry.
- Udržateľnosť
- Mal by obsahovať (Should have)
- Priorita 2
- Menej náročná

P15: API rozhranie

- Systém bude podporovať komunikáciu s ďalšími komponentmi pomocou API.
- Použiteľnosť
- Musí obsahovať (Must have)
- Priorita 1
- Stredne náročná

P16: Automatické generovanie dokumentácie API

- Systém umožní automatické generovanie API dokumentácie vo formáte OpenAPI.
- Použiteľnosť
- Musí obsahovať (Must have)
- Priorita 2
- Menej náročná

P17: Integrácia API rozhrania služby ARES

- Systém bude integrovať API rozhranie služby ARES. Pomocou tohto API bude systém schopný na základe poskytnutého IČO zákazníkom, získať ďalšie informácie o zákazníkovi a predvyplniť ich pre neho.
- Funkčnosť
- Mohol by obsahovať (Could have)
- Priorita 4
- Menej náročná

1.7 Metodiky vývoja

Vývoj aplikácií je náročný proces, ktorý sa rozsahom výsledného produktu výrazne komplikuje. V rámci štandardných metodík vývoja rozlišujeme tri hlavné metodiky, ktoré sa používajú. Patrí medzi ne:

Vodopád je klasický lineárny model vývoja softvéru, kde každá fáza začína až po skončení tej predchádzajúcej. Tradičný vodopádový vývoj má nasledujúce fázy: 1. Definícia požiadaviek, v tejto fáze sú zhromaždené a definované všetky požiadavky na systém, 2. Návrh, v tejto fáze je na základe definovaných požiadaviek navrhnutá celková architektúra systému, 3. Implementácia, táto fáza je venovaná samotnej implementácií systému podľa špecifikácie z predchádzajúcej fázy, 4. Testovanie, systém je podrobený testovaniu, na overenie funkčnosti podľa požiadaviek, 5. Údržba, vo fáze údržby sú opravované chyby, ktoré sa našli pri prevádzke a podľa potreby sú vytvárané ďalšie aktualizácie.

Medzi hlavné výhody vodopádového modelu patrí jasne definovaný plán už na začiatku projektu, ktorý zostáva nemenný počas celého projektu. Takto definovaný plán nám zároveň umožňuje veľmi dobrú predikovateľnosť časovej cenovej, ale aj rozsahovej náročnosti projektu. Vďaka tomu sa zjednodušuje komplexita koordinácie práce na projekte medzi množstvom vývojárov.

Okrem výhod však obsahuje aj nevýhody. Medzi tie patrí potreba mať dobre definované požiadavky už na začiatku projektu. Pokiaľ vznikne potreba na zmenu nejakých požiadaviek, alebo nové požiadavky, tak sa reaguje podstatne horšie.

Iteratívny vývoj je prístup k vývoju softvéru, ktorý obsahuje opakujúce sa cykly, takzvané iterácie. V rámci každej iterácie prebieha vodopádový model vývoja, na ktorého konci sa spraví vyhodnotenie, ktoré vstupuje ako podklad do ďalšej iterácie.

Agilný vývoj je prístup založený na iteratívnom vývoji, ktorý je zameraný na rýchle dodávanie častí. Rozdiel s iteratívnym vývojom sú kratšie iterácie, ktoré nie vždy končia použiteľný produkt.

[14]

1.8 API

API je skratka pre aplikačné programovacie rozhranie – softvérový sprostredkovateľ, ktorý umožňuje dvom aplikáciám komunikovať medzi sebou. Rozhrania API predstavujú spôsob extrahovania a zdieľania údajov v rámci organizácií a medzi nimi.

Termín „API“ sa všeobecne používa na opis rozhraní pripojenia k aplikácii. V priebehu rokov však moderné API nadobudlo niektoré jedinečné vlastnosti, ktoré skutočne zmenili technologický priestor. Po prvé, moderné rozhrania API dodržiavajú špecifické štandardy (zvyčajne HTTP a REST), ktoré umožňujú, aby boli rozhrania API priateľské pre vývojárov, boli zdokumentované, ľahko dostupné a široko pochopiteľné.

Okrem toho sa dnes s API zaobchádza viac ako s produktami než s kódom. Sú určené na spotrebu pre špecifické publikum (napr. vývojári mobilných zariadení) a sú zdokumentované a verzované spôsobom, ktorý umožňuje používateľom jasne očakávať ich údržbu a životný cyklus.

Keďže API sú štandardizovanejšie, možno ich monitorovať a spravovať z hľadiska výkonu aj rozsahu. A čo je najdôležitejšie, majú oveľa silnejšiu disciplínu v oblasti bezpečnosti a riadenia.

A napokon, ako každý iný softvér, ktorý sa vyrába, aj moderné API, má svoj vlastný životný cyklus vývoja softvéru (SDLC) – od analyzovania, navrhovania a testovania, až po vytváranie, správu a ukončenie používania. Tieto API sú dobre zdokumentované pre spotrebu, aj pre vytváranie verzií v procese.[15]

1.9 Druhy API

Existujú rôzne druhy API, podľa technológií a paradigiem, na ktorých sú postavené. Pri výbere typu, ktorý by šiel použiť na komunikáciu medzi frontendovou časťou a backendovou časťou License Manageru som sa rozhodoval medzi nasledujúcimi najrozšírenejšími druhmi.

1.9.1 SOAP API

SOAP je štandardný protokol, ktorý bol navrhnutý tak, aby aplikácie vytvorené v rôznych jazykoch a na rôznych platformách mohli medzi sebou komunikovať. Keďže ide o protokol, zavádza vstavané pravidlá, ktoré zvyšujú jeho zložitosť a réžiu, čo môže viesť k dlhšiemu času spracovávania. Tieto štandardy však ponúkajú aj vstavané normy, vďaka ktorým sú vhodnejšie pre podnikové scenáre. Medzi vstavané normy patrí bezpečnosť, atomicita, konzistencia, izolácia a odolnosť (ACID), čo je súbor vlastností na zabezpečenie spoľahlivých databázových transakcií.

Bežné špecifikácie webových služieb zahŕňajú:

- Zabezpečenie webových služieb (WS-security): Štandardizuje spôsob zabezpečenia a prenosu správ prostredníctvom jedinečných identifikátorov, nazývaných tokeny.
- WS-ReliableMessaging: Štandardizuje spracovanie chýb medzi správami prenášanými cez nespoľahlivú infraštruktúru.
- Adresovanie webových služieb (WS-addressing): Zabalí informácie o smerovaní ako metadáta v hlavičkách SOAP namiesto toho, aby sa tieto informácie uchovávali hlbšie v sieti.
- Jazyk opisu webových služieb (WSDL): opisuje, čo webová služba robí a kde táto služba začína a končí.

Keď sa požiadavka o dáta odošle do SOAP API, môže sa spracovať prostredníctvom ktoréhokoľvek z protokolov aplikačnej vrstvy: HTTP (pre webové prehliadače), SMTP (pre e-mail), TCP a ďalších. Po prijatí požiadavky však musia byť odpovede SOAP vrátené ako dokumenty XML. Dokončenú požiadavku na SOAP API prehliadač nemôže uložiť do vyrovnávacej pamäte, takže k nej nie je možné pristupovať neskôr bez opätovného odoslania do API.[16]

1.9.2 REST API

REST je súbor architektonických obmedzení. REST nie je protokol, ani štandard, na základe čoho si ho vývojári API môžu implementovať rôznymi spôsobmi. API, ktoré spĺňajú tieto obmedzenia sa často nazývajú aj RESTful API.

Požiadavka klienta, zadaná cez RESTful API, prenáša reprezentáciu stavu zdroja na koncový bod. Tieto informácie, alebo reprezentácie sa doručujú v jednom z niekoľkých formátov cez HTTP: JSON (Javascript Object Notation), HTML, XML, Python, PHP alebo obyčajný text. JSON je najpopulárnejší formát súborov, ktorý sa používa, pretože napriek svojmu názvu je jazykovo nezávislý a je čitateľný pre ľudí aj stroje.

Hlavičky a parametre sú podstatnou súčasťou metód HTTP pri RESTful API HTTP požiadavkách, pretože obsahujú dôležité informácie. Medzi informácie, ktoré môžu byť obsiahnuté v hlavičke HTTP požiadavky, patria informácie o identifikátoroch, autorizácií, jednotnóm identifikátori zdroja (URI), ukladanie do vyrovnávacej pamäte, súbory cookie a viac. HTTP hlavičky sú rozdielne pre požiadavky a odpovede na požiadavky, kde každá má svoje vlastné informácie o pripojení HTTP a stavových kódoch.

Aby bolo API považované za RESTful, musí spĺňať tieto kritériá:

- Architektúra klient-server tvorená klientmi, servermi a prostriedkami, pričom požiadavky sú riadené prostredníctvom HTTP.
- Bezstavová komunikácia klient-server, čo znamená, že medzi požiadavkami na získanie nie sú uložené žiadne informácie o klientovi a všetky požiadavky sú samostatné a neprepojené.
- Dáta ukladateľné do vyrovnávacej pamäte, ktoré zefektívňujú interakciu klient-server.
- Jednotné rozhranie medzi komponentmi, takže informácie sa prenášajú v štandardnej forme. To vyžaduje, aby:
  - požadované prostriedky boli identifikovateľné a oddelené od reprezentácie zaslanej klientovi.
  - klient môže manipulovať s prostriedkami prostredníctvom reprezentácie, ktorú dostane zo serveru, pretože reprezentácia obsahuje dostatok informácií.
  - samopopisné správy vrátené klientovi majú dostatok informácií na to, aby opísali, ako by ich mal klient spracovať.
  - hypertext/hypermedia je k dispozícii, čo znamená, že po prístupe k prostriedku by mal byť klient schopný pomocou hypertextových odkazov nájsť všetky ostatné aktuálne dostupné akcie, ktoré môže vykonať.
- Vrstvený systém, ktorý organizuje každý typ servera (zodpovedného za bezpečnosť, vyrovnávanie záťaže atď.) zapojeného do získavania požadovaných informácií do hierarchií, neviditeľných pre klienta.
- Code-on-demand (voliteľné): možnosť na požiadanie poslať spustiteľný kód zo servera klientovi, čím sa rozširuje funkčnosť klienta.

Aj keď REST API musí spĺňať tieto kritériá, stále sa považuje za jednoduchšie na použitie ako predpísaný protokol, akým je napríklad SOAP.[17]

1.9.3 gRPC API

gRPC je open-source API architektúra a systém riadený Cloud Native Computing Foundation. Je založený na modeli Remote Procedure Call. Zatiaľ čo model RPC je široký, gRPC je špecifická implementácia.

V RPC komunikácia medzi klientom a serverom funguje tak, ako keby požiadavky klientskeho rozhrania API boli lokálnou operáciou, alebo keby požiadavka bola interným kódom servera.

V RPC klient odošle požiadavku procesu na server, ktorý vždy počúva vzdialené volania. V požiadavke obsahuje funkciu servera, ktorá sa má volať, spolu s akýmikoľvek parametrami, ktoré sa majú odovzdať. RPC API používa ako základný mechanizmus výmeny údajov protokol ako HTTP, TCP alebo UDP.

gRPC je systém, ktorý implementuje tradičné RPC s niekoľkými optimalizáciami. Napríklad gRPC používa protokolové vyrovnávacie pamäte a HTTP 2 na prenos údajov.

Tiež abstrahuje mechanizmus výmeny údajov od vývojára. Napríklad iná široko používaná implementácia RPC API, OpenAPI, vyžaduje, aby vývojári mapovali koncepty RPC na protokol HTTP. Ale gRPC abstrahuje základnú HTTP komunikáciu. Tieto optimalizácie robia gRPC rýchlejším, jednoduchším na implementáciu a priaznivejším pre web ako iné implementácie RPC.[18]

1.9.4 GraphQL

GraphQL je dotazovací jazyk pre API a spúšťač pre vyhľadávanie týchto dotazov s existujúcimi dátami. V širšom zmysle je GraphQL syntax, ktorú vývojári môžu použiť na požiadanie o konkrétne dáta a vrátenie týchto dát z viacerých zdrojov. Keď klient definuje štruktúru požadovaných dát, server vráti dáta pomocou rovnakej štruktúry.

Vývojári môžu vytvárať aktualizácie dát v reálnom čase prostredníctvom schopností GraphQL pre procesy čítania, úprav dát a monitorovania. Serverové implementácie pre GraphQL boli vyvinuté pre použitie s populárnymi programovacími jazykmi ako sú JavaScript, Python, Ruby, C#, Go a PHP. Cieľom GraphQL je poskytnúť vývojárom komplexný pohľad na dáta uložené v rámci API. To zahŕňa schopnosť prijímať iba dáta priamo relevantné pre určité dotazy a vytváranie architektúry, ktorá zjednodušuje škálovanie a prispôsobovanie sa API v čase.

Na vytvorenie API pomocou GraphQL musí server obsluhovať API a klienta, ktorý sa pripája ku koncovému bodu aplikácie. GraphQL API sa skladá z troch základných komponentov:

- Schéma. Typový systém používaný na definovanie API pre implementáciu servera, vrátane všetkých jeho schopností a funkcií.
- Dotaz. Požiadavka alebo inštrukcia na výstup. Nové dotazy sa deklarujú pomocou kľúčového slova a môžu podporovať vnorené položky, polia a argumenty.
- Resolver. Funkcia, ktorá určuje, ako a kde môže API pristupovať k dátam v danom poli. Bez toho by GraphQL server nevedel, ako spracovať dotazy.

[19, 20]

1.9.5 Webhook

Webhook je prostriedok na automatizáciu odpovedí pre softvér. Niekdy sa nazývajú aj ako „rozhrania API založené na udalostiach“. To znamená, že nový používateľ, alebo aplikácia urobí konkrétnu akciu, alebo sa vyskytne udalosť a táto konkrétna akcia spustí webhook, ktorý zareaguje podľa predchádzajúceho naprogramovania.

Existujú webhooky, ktoré pomáhajú dvom zariadeniam komunikovať, ale táto komunikácia je skutočne jednostranná.

Výsledkom je, že webhooky môžu byť pomerne komplikované nástroje, ale vždy sa držia tohto zjednocujúceho konceptu.[21]

1.10 Architektúra webovej aplikácie

Jednoducho povedané, architektúra webovej aplikácie je rámec, ktorý definuje, ako jednotlivé prvky webovej aplikácie spolupracujú. Máme tu na mysli prvky ako komponenty, databázy, servery, používateľské rozhrania alebo middleware systémy.

Dobre navrhnutá architektúra webovej aplikácie zaručuje, že server na strane klienta, nazývaný aj frontend, bude logicky spojený s backend serverom a že obe prostredia budú kompatibilné.

Moderná architektúra webových aplikácií musí byť navrhnutá tak, aby bola škálovateľná, bezpečná a efektívna.

V rámci architektúry webovej aplikácie rozlišujeme dve hlavné časti, ktoré plnia svoje špecifické úlohy, frontend a backend.[22]

1.10.1 Frontend

Časť webovej stránky, s ktorou používateľ priamo interaguje, sa nazýva frontend. Označuje sa tiež ako „klientská strana aplikácie". Zahŕňa všetko, s čím používatelia priamo interagujú: farby a štýly textu, obrázky, grafy a tabuľky, tlačidlá, farby a navigačné menu. HTML, CSS a JavaScript sú jazyky používané na vývoj frontendu.

Reakcia a výkon sú dva hlavné ciele frontendu. Vývojár musí zabezpečiť, aby stránka bola responzívna, t. j. aby sa správne zobrazovala na zariadeniach všetkých veľkostí, žiadna časť webovej stránky by sa nemala správať abnormálne, bez ohľadu na veľkosť obrazovky.[23]

1.10.2 Backend

Backend, nazývaný aj serverová strana, je infraštruktúra, ktorá podporuje frontend a pozostáva z častí softvéru, ktoré bežní používatelia nevidia. Backend je v podstate mozog webovej stránky.

Backend zahŕňa server, ktorý poskytuje údaje vždy, keď sú požadované, databázu, v ktorej sú tieto údaje usporiadané, a aplikáciu, ktorá tieto informácie dodáva.

Na zmenu webovej stránky na dynamickú webovú aplikáciu, je nutné pridať ďalšie komponenty backendu. To sa líši od statickej webovej stránky, kde obsah často zostáva rovnaký a nepotrebuje databázu.

Vývojári backendu riešia všetko, čo nezahŕňa poskytovanie používateľského rozhrania. To môže zahŕňať písanie API, vytváranie knižníc a pomocných programov. Uľahčujú komunikáciu medzi prezentačnou a obchodnou vrstvou. V porovnaní s frontend webovými dizajnérmi zohrávajú kľúčovú a vysoko spolupracujúcu úlohu pri vývoji webu.

Zjednodušene povedané, backendoví vývojári vytvárajú kód, aby zabezpečili, že všetko na frontende funguje správne. Aby sa zabezpečilo efektívne fungovanie webovej stránky, zvyčajne trávia viac času, ako weboví dizajnéri zisťovaním logistiky a zavádzaním algoritmov.[24]

1.10.3 Previazanosť Backendu a Frontendu

Tradičný prístup spočíva v aplikáciach, ktoré robia veľa rôznych vecí, pričom všetky funkcie sú v rámci rovnakej kódovej základne.

Jedna softvérová aplikácia je zodpovedná za backendovú logiku aj za vykresľovanie frontendu webovej stránky. Inými slovami, celá aplikácia je jedna robustná kódová základňa, ktorá robí všetko, od pripojenia k databázam, až po výstup jednoduchého textu.

S týmto nastavením frontend existuje len ako koncept a zdieľa rovnaký životný priestor ako backend.

Tieto webové stránky sa často označujú ako monolity.

Hoci monolity sú ľahké na vytvorenie, môžu sa rýchlo stať problémovými pri škálovaní a udržiavaní. To preto, že ak sa snažíte škálovať jednotlivú časť, alebo funkciu, musíte zväčšiť celú aplikáciu pridaním kapacity servera.

Ako rastie infraštruktúra, rastie aj potreba lepšej architektúry. Snaha o rozšírenie frontendu môže viesť k ústupkom a k vytvoreniu kódu, na ktorý sa každý bojí siahnuť. Čo vedie k náročnému udržiavaniu

Pre vývojárov frontendu jedným z najväčších problémov sú dáta, ktoré im backend neposkytuje a na ktorých získanie je nutné sa ponoriť do backendovej obchodnej logiky, na zistenie, toho čo vyžaduje frontend. Následne pri vykonaní zmien v kóde, môžu nastať neočakávané dôsledky v niektorej inej časti aplikácie.

To znamená väčšie a dlhšie testovacie cykly a vyžaduje to viac skúseností s platformou.

Má to aj negatávny vedľajší účinok pre tímovú prácu, pretože hranica medzi vývojom backendu a frontendu sa stáva nejasnou, s šedými oblasťami v kóde, za ktoré sa nikto necíti byť zodpovedný. [25]

1.10.4 Jedno stránkové aplikácie

Jedno stránková aplikácia z angl. Single Page Aplication (SPA), je jedna webová stránka, webová lokalita alebo webová aplikácia, ktorá funguje v rámci webového prehliadača a načítava iba jeden dokument. Počas používania nepotrebuje opätovné načítavanie a väčšina jej obsahu zostáva rovnaká, zatiaľ čo iba časť obsahu potrebuje aktualizáciu. Pri potrebe aktualizácie obsahu využíva volanie na API pomocou JavaScriptu.

Týmto spôsobom sa používatelia pri používaní webstránky vyhnú načítaniu celej novej stránky zo servera počas navigácie na danej webovej aplikácií. Výsledkom je zvýšenie výkonu a nadobudnutie pocitu používania natívnej aplikácie. Ponúka svojím používateľom dynamickejší webový zážitok. SPA pomáhajú používateľom byť v jednom, nekomplikovanom webovom priestore jednoduchými a funkčnými spôsobmi.

Medzi príklady najznámejších SPA patria Gmail, Facebook, Trello, Google Maps atď.. Všetky z menovaných ponúkajú vynikajúcu používateľskú skúsenosť v prehliadači, bez opätovného načítania stránky.[26]

V rámci architektúry SPA server odošle HTML dokument iba pri prvej požiadavke a pri ďalších požiadavkách posiela už iba dáta vo formáte JSON, pokiaľ je nutné. To znamená, že SPA prepíše obsah aktuálnej stránky a nenačíta celú novú webovú stránku. Toto vedie na vyšší výkon počas používania, bez potrebného času na opätovné načítanie webstránky. Táto schopnosť umožňuje SPA sa správať ako natívna aplikácia.

1.10.5 Viac stránkové aplikácie

Viac stránkové aplikácie fungujú „tradičným“ spôsobom. Každá zmena, napríklad zobrazenie dát, alebo odoslanie dát späť na server, vyžaduje vykreslenie novej stránky zo servera v prehliadači. Tieto aplikácie sú veľké, väčšie ako jednostránkové aplikácie, pretože to potrebujú. Kvôli množstvu obsahu majú tieto aplikácie mnoho úrovní používateľského rozhrania.

V porovnaní s jedno stránkovými aplikáciami môžu viac stránkové aplikácie mať rýchlejšie načítanie pri prvom spustení. Taktiež je oveľa jednoduchšie dosiahnuť vysoké umiestnenie viac stránkových aplikácií vo vyhľadávačoch, pretože podkladové HTML odzrkadľuje ich obsah veľmi dobre. Ostatnou výhodou je, že viac stránkové aplikácie majú lepšiu kompatibilitu so starými prehliadačmi, než jednostránkové aplikácie a nepotrebujú nutne JavaScript.[27, 28]

1.10.6 Backend pre Frontend

Backend pre Frontend je vzor softvérovej architektúry, ktorý pomáha oddeliť backend a frontend aplikácie, pričom im stále umožňuje bezproblémovú spoluprácu. Tento vzor je obzvlášť užitočný pri vytváraní zložitých, rozsiahlych aplikácií, ktoré potrebujú spracovať viacero typov klientov, ako sú webové, mobilné a IoT zariadenia.

Vrstva BFF funguje ako sprostredkovateľ medzi backendom a frontendom, čo umožňuje komunikáciu medzi nimi, bez potreby priamej integrácie. To umožňuje flexibilnejší frontend, pretože nemusí byť pevne spojený s backendom. Vrstva BFF navyše dokáže spracovať aj úlohy, ako je bezpečnosť, ukladanie do vyrovnávacej pamäte a transformácia údajov, ktoré je možné prispôsobiť potrebám frontendu.[29]

1.11 Dizajn používateľského rozhrania

Používateľské rozhranie spraví na potencionálneho zákazníka prvý dojem. Je jednou z vecí, ktorú si zákazník všimne ako prvú a ktoré zaváži, či bude softvér používať, alebo nie.

1.11.1 Material Design

Material Design je dizajnový systém vyvinutý spoločnosťou Google, ktorý slúži k vytváraniu konzistentého a esteticky príjemného používateľského rozhrania pre mobilné aplikácie, webové stránky a ďalšie digitálne platformy. Tento dizajnový systém bol prvý krát predstavený v roku 2014.

Material Design vychádza z myšlienky, že digitálne rozhranie by malo simulovať vlastnosti fyzických materiálov a reagovať na používateľské interakcie rovnakým spôsobom, akým by reagovali fyzické objekty v skutočnom svete. Kľúčovými prvkami Material Design sú vrstvy, tiene, farebné palety a plynulé animácie, ktoré vytvárajú príjemný a intuitívny používateľský zážitok.

Material Design obsahuje aj sadu pravidiel a doporučení pre vývojárov, aby výsledné UI bolo konzistentné naprieč rôznymi platformami a zariadeniami. Tím sa zaručuje, aby používatelia na rôznych platformách mali podobný zážitok.

Aby uľahčil prácu vývojárom a dizajnérom, tak okrem pravidiel a doporučení, obsahuje Material Design aj sadu funkčných a interaktívnych komponentov. Tieto komponenty uľahčujú tvorbu dizajnu a aplikácií, a zároveň zachovávajú konzistentnosť. [30, 31, 32]

1.11.2 Flat Design

Flat Design by sa dal považovať za opak skeuomorfizmu. Flat Design namiesto použitia skutočných snímok, ako pri skeuomorfnom dizajne, používa 2-rozmerné vizuálne detaily.

Flat Design nepoužíva tiene ako Material Design. Flat Design stelesňuje digitálny priestor, z ktorého vzišiel.

Flat Design je zredukovaný na minimum. Nie, aby bol zámerne minimalistický, ale aby sa zameriaval na funkciu dizajnu. Estetika zohráva významnú úlohu vo Flat Design, kde sa veľkoryso používa výrazná farba. Nedostatok tieňa a hĺbky však môže spôsobiť problémy s použiteľnosťou.[33]

Kapitola 2
Návrh aplikácie

Táto kapitola sa venuje návrhu aplikácie. Na základe tohto návrhu sa bude postupovať v implementácií.

2.1 Prípady použitia

Pokiaľ chceme zachytiť funkčné požiadavky, ktoré sú kladené na náš systém, máme na výber z viacero nástrojov v rámci UML. Na zachytenie funkčných požiadaviek som zvolil prípady použitia. Prípady použitia sú zachytené v textovej podobe v nasledujúcej časti. Následne sú namapované na funkčné požiadavky. [34]

UC1 - Zaregistrovanie sa Zákazník, ktorý ešte nemá účet a chce si niečo kúpiť, tak sa zaregistruje. Počas registrácie je potrebné zadať meno, email a heslo.

UC2 - Prehľad produktov Zákazník si môže prezerať všetky zverejnené produkty, ich cenové plány a detaily.

UC3 - Objednanie produktu Zákazník si zvolí konkrétny cenový plán produktu, vyplní konfiguráciu produktu, vyplní fakturačné údaje, alebo si vyberie z údajov z predchádzajúcich objednávok daného zákazníka a na konci je mu vystavená faktúra, ktorá je zobrazená pri objednávke.

UC4 - Navýšenie licencie Pre všetky licencie, ktoré zákazník má aktívne a ak produkt, na ktorý bola vydaná táto licencia obsahuje vyššie cenové plány, ako ten, na ktorý je aktuálne vystavená licencia, môže zákazník túto licenciu navýšiť na takýto vyšší cenový plán.

UC5 - Ukončenie licencie Pokiaľ má zákazník aktívnu licenciu, tak v rámci detailného prehľadu licencie sa nachádza tlačidlo, pomocou ktorého zákazník danú licenciu zruší.

UC6 - Obnovenie hesla Keď zákazník zabudne heslo, môže si ho obnoviť pomocou svojho mailu, ktorým sa zaregistroval.

UC7 - Zrušenie objednávky Zákazník má možnosť zrušiť objednávku v ktoromkoľvek bode procesu jej vytvárania, až do bodu keď sa začne objednávka nasadzovať.

UC8 - Zmena hesla Zákazník má možnosť zmeniť si svoje heslo do účtu.

UC9 - Požiadanie o zákazníkom definovaný cenový plán Pokiaľ zákazníkovi nevyhovuje žiaden z ponúkaných cenových plánov a k produktu sú povolené zákazníkom definované cenové plány, tak si môže zákazník požiadať o zákazníkom definovaný cenový plán. Takýto cenový plán musí následne spracovať administrátor.

UC10 - Vytvorenie produktu Administrátor vie po prihlásení vytvoriť nový produkt. Súčasťou vytvorenia produktu sú aj vytvorenia cenových plánov a dotazníku na konfiguráciu produktu

UC11 - Prehľad zákazníkov Administrátor vie po prihlásení zobraziť všetkých zákazníkov.

UC12 - Asistované objednanie produktu Administrátorovi je umožnené vytvoriť objednávku a používateľský účet v mene iných osôb.

Následná tabuľka obsahuje namapovanie požiadavkov na prípady použitia.2.1

Tabuľka 2.1 Pokrytie požiadavkov prípadmi použitia

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P17
UC1						X		X			X		
UC2		X				X			X				
UC3	X	X			X	X	X	X					X
UC4	X		X		X	X		X					
UC5			X			X							
UC6		X	X			X		X					
UC7	X		X			X							
UC8						X		X					
UC9	X	X	X			X	X						
UC10						X			X				
UC11						X							
UC12	X					X			X				

2.2 Komunikácia s ostatnými komponentami

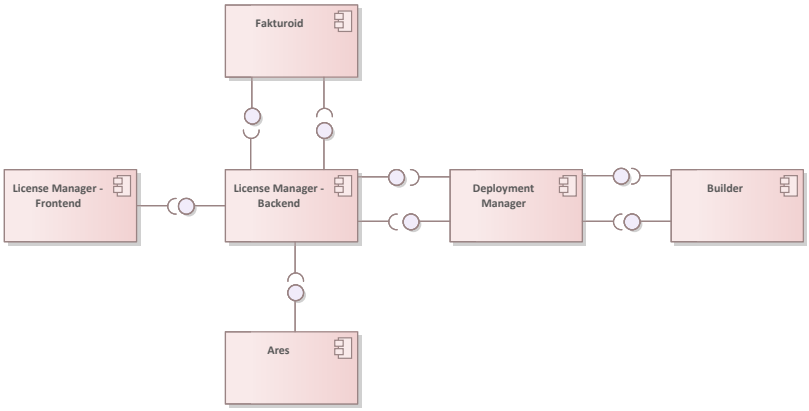
Pôvodný návrh License Manageru komunikoval priamo s Builderom pri počiatočnom zostavovaní objednávky, navýšení, zablokovaní a zrušení licencie. Pri vzniku Deployment Manageru vzišla otázka ako bude komunikovať s License Managerom a Builderom.

Výsledný návrh, aby sa čo najmenej duplikovala komunikácia medzi týmito komponentami, spočíval v Deployment Manageru ako prostredníku v komunikácií medzi Builderom a License Managerom. Pôvodné požiadavky z License Manageru do Builderu, si Deployment Manager rozšíri o svoje ďalšie parametre pre Builder.

Deployment Manager ešte navyše umožňuje License Manageru pridať produkt z License Manageru aj do Deployment Manageru, kde si ho následne administrátor doplní o parametre, ktoré sú potrebné pre Deployment Manager, ale nie sú súčasťou License Manageru.

Po mimo App Manageru komunikuje License Manager ešte s externým systémom Fakturoid pri vystavovaní a kontrole faktúr.2.6

Kompletná komunikácia s ostatnými komponentmi je zachytená na obrázku 2.1



Obr. 2.1 Diagram komponentov

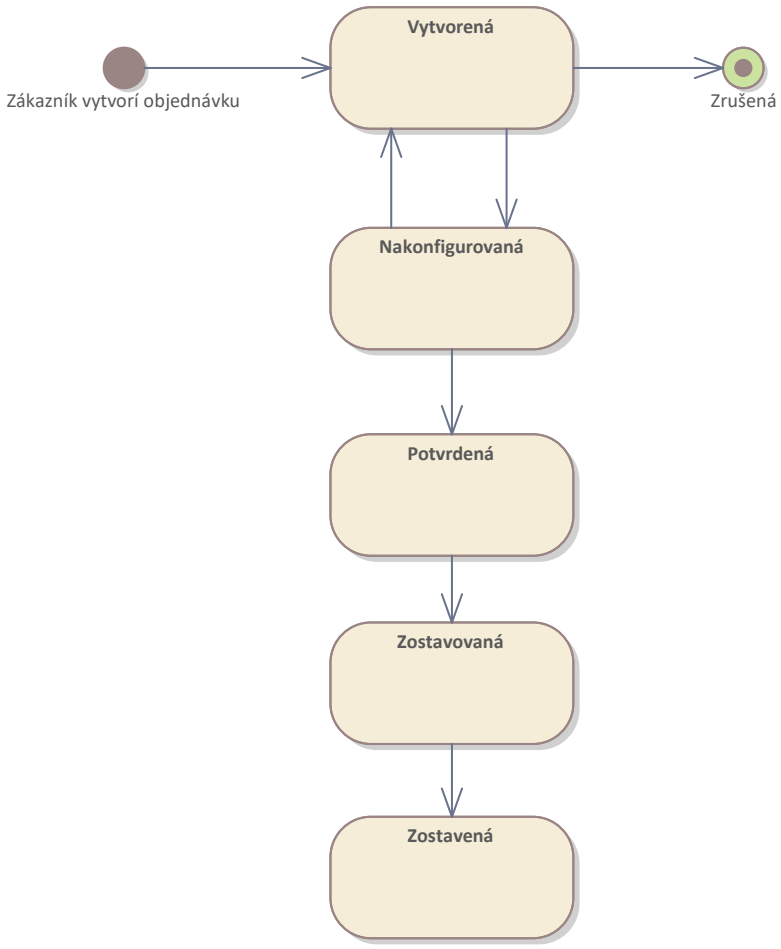
2.3 Objednávka

V rámci procesu objednávania mení postupne objednávka svoj stav. Po vytvorení objednávky je objednávka v stave vytvorená. V stave vytvorená zákazník vyplní požadovaný konfiguračný formulár a vyberie si fakturačné údaje. Následne prechádza objednávka do stavu nakonfigurovaná. V stave nakonfigurovaná je zobrazená rekapitulácia objednávky a zákazník objednávku buď potvrdí, alebo sa môže vrátiť naspäť ku konfigurácií a výberu fakturačných údajov. Pri potvrdení objednávky je odoslaná požiadavka na zostavenie žiadanej konfigurácie produktu do Deployment Manageru. Následným potvrdením doručenia požiadavky na zostavenie prejde požiadavka do stavu zostavovaná. V tomto stave je až dokým nepríde z Deployment Manageru informácia o kompletnom zostavení objednávky a potom prechádza do stavu zostavená a je k nej vytvorená licencia.

V prípade že sa zákazník rozhodne, že nechce pokračovať v objednávke a chce ju zrušiť, môže tak spraviť pred potvrdením objednávky a to buď priamo zrušením objednávky zo stavu vytvorená, alebo prechodom naspäť do stavu vytvorená zo stavu nakonfigurovaná a následným zrušením. Zrušenie objednávky prebieha zmenením stavu do stavu zrušená. Objednávka v takomto stave sa už nezobrazuje zákazníkovi medzi jeho objednávkami, ale zostáva zachovaná v databáze aj so všetkými svojimi vyplnenými odpoveďami v rámci konfiguračného formuláru. Toto riešenie poskytuje možnosť prípadného obnovenia zadaných vstupov pre zákazníka, pokiaľ by sa rozhodol zmeniť svoje rozhodnutie. Mimo toho ešte umožňuje pre administrátora prípadnú hlbšiu analýzu, záujmu a nezáujmu o produkt.

2.4 Produkt

Pri vytvorení produktu sa produkt nachádza v stave nový, v tomto stave je možné ho akokoľvek upravovať a nezobrazuje sa zákazníkom. Keď je produkt pripravený na zverejnenie prejde do stavu verejný. Od tohto bodu už nie je možné zmazať cenový plán, ale stále je možné ho skryť a všetky ostatné operácie nad produktom sú stále dostupné. V prípade ukončenia predávania produktu prejde produkt do stavu skrytý. V tomto stave už nie je zobrazovaný zákazníkom, tento stav nie je trvalý a produkt môže prejsť naspäť do stavu verejný.



Obr. 2.2 Stavový diagram objednávky

2.4.1 Tvorba produktu

Tvorba produktu mala počas práce s pôvodnou verziou License Manageru v rámci predmetov SP1/2, najhorší UX zo všetkých častí. Hlavnými nedostatkami bola prakticky neexistujúca spätná väzba. Tvorbu produktu som preto rozdelil do niekoľkých častí na karta produktu, detail produktu, cenové plány, vstupný formulár a všeobecné veci o produkte.

Karta produktu je komponent ktorý sa používateľovi zobrazí na hlavnom rozcestníku produktov. Následným stlačením tlačidla na zistenia ďalších informácií, sa zobrazia používateľovi detailnejšie informácie o produkte.

Pri tvorbe produktu je možnosť pridať vizualizáciu konečnej podoby, pomocou recyklácie komponentov používaných pri zobrazení produktu naostro.

2.4.2 Konfiguračný formulár

Najkomplexnejšou časťou tvorby produktu je vytvorenie konfiguračného formuláru, ktorý následne zákazníci vypĺňajú pri zadávaní objednávky.

Konfiguračný formulár predstavuje súbor otázok a vstupov, ktoré sú v tvare stromu, teda sú koreňové otázky a tie následne môžu obsahovať podotázky, ktoré môžu rekurzívne obsahovať svoje vlastné podotázky. Otázka následne môže obsahovať vstupy. Otázka je ďalej daná horným a spodným limitom. Limity určujú minimálne koľkokrát musí byť daná otázka obsiahnutá v nadradenej otázke alebo produkte, a maximálny počet takéhoto obsiahnutia danej otázky. Tieto limity môžu byť viazané k premennej.

V prípade že ku produktu už existuje objednávka, tak máme odpovede od zákazníka na otázky a vstupy. Zmena, alebo zmazanie týchto vstupov, by rozhodilo databázu. Z tohto dôvodu som pridal možnosť skryť otázku alebo vstup. V prípade že je skrytá otázka, tak dedičným spôsobom sú skryté aj všekty jej vstupy a všetky podotázky.

Pri návrhu UI som sa snažil tento proces zjednodušiť a spraviť ho omnoho viac responzívny, podobajúci sa výslednému formuláru. Otázka sa vytvorí priamo na mieste kam patrí, priamo pod produkt, alebo pod inú otázku. Podobným postupom sa vytvorí vstup priamo v otázke, ku ktorej patrí. Vytvorenie premennej produktu prebieha pri určovaní medzí. Tlačidlo na vytvorenie premennej sa nachádza priamo vo výbere premenných.

2.4.3 Cenový plán produktu

Pri tvorbe cenového plánu produktu sa určí inštalačný poplatok, ktorý je jednorázový a bude obsiahnutý v prvej faktúre. Ďalej sa určí pravidelný poplatok a perióda, s ktorou je tento poplatok fakturovaný. Táto perióda sa udáva v mesiacoch. Kombináciou týchto parametrov sme schopný nastaviť tak ako licencie na dobu neurčitú, tak aj licencie na princípe predplatného. Ďalej určíme hodnotu všetkých premenných toho produktu, ktoré nám určia ako sa má zobraziť konfiguračný formulár v procese objednávania.

Cenovému plánu som pridal stav daného cenového plánu. Tento stav zahrňuje, či sa má daný cenový plán zobrazovať medzi cenovými plánmi produktu potencionálnym zákazníkom. V prípade, že by cenový plán už nemal byť ponúkaný, tak ho prepneme do stavu skrytý a neovplyvní to už aktívne licencie zakúpené v tomto cenovom pláne.

2.5 Návrh License Manager Backendu

Na základe analýzy prvej verzie License Manageru, som sa rozhodol ponechať dátovú a biznisovú vrstvu tak ako sú, a prerobiť prezentačnú vrstvu na formu API, s ktorou budem následne komunikovať pomocou frontendovej aplikácie. Týmto spôsobom chcem odstrániť hlavný nedostatok aktuálneho stavu License Manageru a to je neprívetivé používateľské rozhranie.

2.5.1 Návrh API

Komunikácia medzi Frontendom a Backendom sa bude skladať zväčša z menších požiadavkov na dobre definované objekty, ktoré je potreba v celom obsahu objektu. Na základe týchto potrieb som zvolil REST API na pokrytie týchto požiadaviek, nakoľko sa na to najlepšie hodí. Ostatné druhy API mi prídu až príliš komplikované na tieto potreby.

Pri návrhu koncových bodov som dodržiaval princípy využitia metód HTTP GET pre získanie údajov z API, POST pre vytvorenie objektu cez API, PUT pre aktualizáciu dát a DELETE pre zmazanie objektu. Výnimku z tohto robia koncové body, ktoré pochádzajú z pôvodného License Managera, z ktorého som vychádzal. Koncové body dodržiavajú zásady CRUD pri každom objekte.

Zoznam koncových bodov sa nachádza v prílohe A.

2.5.2 Databáza

Pri návrhu databázovy som vychádzal z databázového modelu ktorý vytvoril Viktor Holý vo svojej bakalárskej práci.[4] Tento databázový model som rozšíril o ďalšie informácie potrebné pre Frontend. Ďalej som zjednotil názvy a typy pri väzbách otázky na produkt a otázky na podotázky.

Rozšírený databázový model sa nachádza v prílohe B.

2.6 Fakturácia

V predchádzajúcej verzii License Manageru bolo implementované napojenie na fakturačné prostredie Fakturoid[35]. Implementácia napojenia bola formou volania API Fakturoidu. Táto implementácia je vyhovujúca v prípade, keď sa vo Fakturoide vytvára subjekt, prípadne faktúra, no prestáva byť vyhovujúca, keď sa snažíme zistiť stav faktúr. Touto implementáciou získavame informáciu o zmene stavu faktúry pomocou opakovaného volania zistenia stavu a porovnávania s predchádzajúcim stavom. Na základe potrebnej rýchlosti identifikácie zmeny nastavíme opakované zisťovanie tohto stavu. Toto riešenie môže viesť na prílišné využívanie poskytovaného API a môže viesť k zbytočnému zahlcovaniu poskytovateľa tohto API. Čo sa podarilo nedopatrením dosiahnuť v rámci vývoja v predmetoch SP1/SP2, pri zisťovaní každú minútu.

Fakturoid poskytuje v rámci svojho API možnosť na nastavenie webhookov pri zmene stavu faktúry. Nastavením tohto webhooku by sme sa zbavili problému neustáleho dopytovania sa na stav faktúry.

2.7 Návrh License Manager Frontendu

Pri návrhu Frontendu som sa zameral na používateľské rozhranie a najmä jeho použiteľnosť. Na dosiahnutie čo najlepšieho používateľského zážitku som si zvolil SPA architektúru, ktorá pôsobí natívnejšie na používateľa, a má rýchlejšie odozvy pri práci v rámci aplikácie. Frontend bude tvoriť samostatnú webovú aplikáciu, ktorá bude čerpať dáta z Backendovej časti.

2.7.1 Výber dizajnu

Pri výbere dizajnu som zhodnocoval ostatné systémy zadávateľa, aby bol dizajn čo najkonzistentnejší naprieč aplikáciami. Zadávateľ vo svojich aplikáciach využíva vo veľkom Material Design. Na tomto základe som sa rozhodol využívať Material Design aj pri tvorbe GUI Frontendu License Manageru.

2.7.2 Vzhľad aplikácie

Návrh vzhľadu aplikácie som robil v online nástroji na grafické návrhy Figma[36]. Počas celej doby grafického navrhovania som sa riadil pravidlami Material Designu. Pri prvotnom grafickom návrhu som sa držal idei hlavnej stránky s produktami, kde má každý produkt svoju vlastnú kartu so základnými informáciami, pár obrázkami a tlačidlom na ktoré umožňuje navigáciu na detailné informácie o danom produkte. V rámci detailných informácii bol priestor na textový popis a prípadné obrázky produktu.

Ďalšou časťou bola objednávka v rámci svojho životného cyklu. Na začiatku vo fáze vyplnenia. V tomto stave sa na ľavej strane zobrazí konfiguračný formulár, a na pravej, voľba fakturačných údajov. Po tomto stave nasledoval stav rekapitulácie, kde bol zhrnutý cenový plán produktu a fakturačné údaje. Po potvrdení objednávky sa zobrazí navyše tabuľka s faktúrami a stav objednávky. Zostavením produktu zo strany Builderu sa zobrazia aj zostavené produkty, pripravené na použitie.

Tvorba produktu pozostávala z formuláru na vyplnenie informácii pre hlavnú a detailnú stránku, tabuľkou cenových plánov a tvorbou konfiguračného formuláru. V rámci konfiguračného formuláru boli zanorené úrovne otázok podla vzťahu medzi otázkou a jej podotázkou. V každej úrovni bolo možné pridať novú podotázku, alebo vstup.

Od zvolenia produktu farebná schéma objednávka je zhodná s farebnou schémou produktu.

Druhý návrh som realizoval po zoznámení sa s knižnicou Vuetify, o ktorej je viac napísané v časti implementácie 3.3.5. Na základe vedomostí nadobudnutých z oboznámenia sa s knižnicou Vuetify, som zjednodušil návrh hlavnej stránky a stránky produktu. Z kariet produktov na hlavnej stránke som odstránil všetky obrázky, až na logo produktu, s tým že toto logo nemusí byť povinné. Obsah stránky produktu som zjednodušil na názov produktu, ktorý môže byť nahradený štylizovaným obrázkom obsahujúcim názov produktu, dlhším textovým opisom produktu, a tabuľkou obsahujúcou cenové plány produktu.

Grafické návrhy aplikácie sa nachádzajú v prílohách C a D.

Kapitola 3

Implementácia návrhu

Táto kapitola je venovaná technológiám a postupom ktoré boli použité počas implementácie tejto práce. Použité technológie sú rozdelené na backendovú a frontendovú časť.

3.1 Vývojový proces

Aby sme vniesli poriadok do vývojového procesu, používame pri ňom nástroje, ktoré nám to umožňujú. Medzi hlavné problémy v rámci vývojového procesu patria otázky čo urobiť, kde to urobiť a ako sledovať čas venovaný vývoju.

Na problém čo urobiť, existujú nástroje na správu úloh. Medzi hlavné patrí Redmine, Jira, GitLab Tasks. Pri vývoji som sa rozhodol využiť Redmine, nakoľko slúži na správu úloh zadávateľa.

V rámci procesu vývoja má každá úloha svoj životný cyklus, ktorým si prechádza od vytvorenia úlohy, až po jej dokončenie.

Pri spracovávaní úloh je dobré udržovať si prehľad o čase, ktorý sme na tej úlohe strávili. Pri určovaní tohto času, sa ho môžme pokúsiť odhadnúť, ale toto riešenie vedie k nepresným výsledkom a preto je lepšie ho merať exaktne.

Keď máme čas zmeraný, tak ho potrebujeme priradiť k danej úlohe. Toto priradenie môžeme robiť manuálne, alebo si prácu zjednodušíme a zvolíme automatický nástroj, ktorý nám automaticky zmeraný čas priradí k danej úlohe. Ja som si pri vývoji zvolil meranie času pomocou nástroju Toggl Track.[37] Tento nástroj slúži na meranie času a obsahuje značky, ktoré ide aplikovať na nameraný čas. Sám o sebe tento nástroj neobsahuje automatické priradenie, ale obsahuje API na získanie času a značiek ktoré sú k nemu priradené.

Na automatické priradenie som zvolil ďalší nástroj Timer2Ticket [38], ktorý slúži ako synchronizačný nástroj medzi Toggl API a API nástroja na správu úloh. Tento nástroj synchronizuje úlohy z nástroja na správu úloh do Togglu v podobe označení a z Togglu vykázaný čas do nástroja na správu úloh.

Na správu súborov som zvolil verzovací systém git [39]. V rámci práci s gitom som dodržiaval zásady práce s gitom. V rámci repozitára som si založil master vetvu a dev vetvu. Následne z dev vetvy som vytváral vetvy s funkciami ktoré som zrovna implementoval. Keď bola implementácia hotová, založil som merge request na zlúčenie naspäť do dev vetvy. Keď bola v dev vetve kompletná celá časť, tak som založil merge request na zlúčenie do master vetvy.

3.2 Backend

Implementácia backendovej časti pokračovala v pôvodnom License Manageri, do ktorého bolo doplnené REST API na komunikáciu s novo vznikajúcim Frontendom. Zvolený bol programovací jazyk pôvodného License Managera, teda PHP s využitím frameworku Laravel. Aj keď jazyk zostal rovnaký od doby, kedy bol napísaný License Manager, vyšli nové verzie tak ako jazyku PHP, tak aj frameworku Laravel. Rozhodol som sa preto aktualizovať kód, aby bežala najnovšia stabilná verzia. Pôvodná verzia PHP bola verzia 8.0 a Laravelu verzia 8.

Aktuálizácia verzie prebiehala vo viacerých krokoch. Prvým krokom bola aktualizácia frameworku Laravel na verziu 9 podľa návodu na aktualizáciu.[40]

Následne bola aktualizovaná verzia PHP na verziu 8.1 ako minimálna verzia jazyku PHP pre verziu Laravelu 10. Hlavnou vecou, ktorú bolo nutné kvôli tejto verzii zmeniť, bol jazykový konštrukt enumerácie, ktorý bol do tohto bodu podporovaný externou knižnicou, no s pridaním enumerácií do PHP verzie 8.1, prestala byť táto knižnica ďalej udržiavaná a bolo nutné prepísať enumerácie definovené knižnicou do podoby akou ich definuje samotné PHP.

Posledným krokom bola aktualizácia frameworku Laravel na verziu 10, podľa návodu na aktualizáciu.[41]

3.2.1 API ovládače

Pri implementácií API endpointov som využil ovládače, ktoré poskytuje Laravel. Ovládače koncových bodov API obsahujú funkcie, ktoré sa spustia po doručení požiadavky. Zlepšujú prehľadnosť súboru obsahujúceho definície koncových bodov API. V rámci definície koncových bodov sa ku konkrétnej URL pridá funkcia, ktorá sa zavolá pri prichádzajúcej požiadavke na koncový bod API. Táto funkcia môže byť buď definovaná anonymne priamo v mieste defínicie koncového bodu, alebo môže byť definovaná na inom mieste a do koncového bodu sa v tomto prípade dá iba ukazateľ na túto funkciu.

V rámci definície ovládačov API som si ich sémanticky rozdelil, aby som udržal zdrojový kód čo najprehľadnejší.

3.2.2 Databázové migrácie

Pri implementácií som vychádzal z už existujúcej databázy ktorú vytvoril Viktor Holý počas svojej bakalárskej práce. Na úpravu databázy, aby odpovedala mojím požiadavkám som zvolil databázové migrácie, ktoré poskytuje Laravel [42]. Namiesto komplikovanej úpravy tabuliek v databázy, takto migrácie umožňujú jej úpravu pomocou jedného príkazu na spustenie migrácií.

Jednotlivé migrácie na seba nadväzujú a tvoria sériu zobrazení, ktoré sa na tabuľkách databázy aplikujú. Pri databázových migráciách definujeme dopredný smer migrácie, čo sa má spraviť s databázou a môžeme definovať aj spätný smer pre prípad, že by sme chceli migráciu vrátiť. Pri migráciách môže dôjsť ku strate dát v rámci databázy.

Výpis kódu 3.1 Príklad databázovej migrácie na pridanie stĺpcu slug to tabuľky produkty

```
class AddSlugToProducts extends Migration
{
    /**
     * Run the migrations.
     */
    public function up()
    {
        Schema::table('products', function (Blueprint $table) {
            $table->string('slug')->default("")->unique();
        });
    }
```

```
    /**
     * Reverse the migrations.
     */
    public function down()
    {
        Schema::table('products', function (Blueprint $table) {
            $table->dropColumn(['slug']);
        });
    }
}
```

3.2.3 Zabezpečenie

Zabezpečenie API je proces ochrany API pred útokmi. Keďže API sú veľmi bežne používané a keďže umožňujú prístup k citlivým softvérovým funkciám a údajom, stávajú sa primárnym cieľom útočníkov.

Zabezpečenie API je kľúčovou súčasťou zabezpečenia moderných webových aplikácií. Rozhrania API môžu mať slabé miesta, ako je nefunkčná autentifikácia a autorizácia, chýbajúce obmedzenie rýchlosti a vloženie kódu. Organizácie musia pravidelne testovať rozhrania API, aby identifikovali slabé miesta a riešili tieto chyby pomocou osvedčených postupov zabezpečenia.[43]

Pri zabezpečení API som využil knižnicu Sanctum [44], ktorá je ponúkaná v rámci frameworku Laravel. Na autorizáciu používateľa slúži v tejto knižnici autorizácia pomocou API tokenov, ktoré sú pri požiadavke na prihlásenie vygenerované. Na následnu autorizáciu pri požiadavkách sú tieto tokeny obsiahnuté v hlavičke HTTP požiadavky.

Na autentizáciu využíva knižnica Sanctum vstavanú autentizáciu Laravelu pomocou cookies. Toto riešenie poskytuje CSRF ochranu, autentizáciu relácie, ako aj ochranu pred únikom autentifikačných údajov cez XSS.[44]

3.2.4 Automatické generovanie dokumentácie API

Vývoj API so sebou prináša svoje úskalia. Z podstaty poskytovania API, ako služby pre ostatných, vychádza potreba mať toto API dobre zdokumentované, aby sme používateľom nášho API, čo najviac uľahčili jeho používanie. Tradične bola tvorba API dokumentácie manuálna činnosť, ktorú si každý robil podľa seba. Tomuto prístupu chýbal spoločný formát, vďaka ktorému by používatelia jednoducho zakaždým pochopili ako API využívať.

Jedným zo štandardov, ktorý by predpisoval ako by mala vyzerať dokumentácia API je OpenAPI Specification. OAS definuje štandardný opis rozhrania HTTP API bez ohľadu na programovací jazyk, ktorý umožňuje ľuďom, aj počítačom, objaviť a pochopiť schopnosti služby bez toho, aby vyžadovali prístup k zdrojovému kódu, dodatočnú dokumentáciu, alebo kontrolu sieťovej prevádzky. Pri správnom definovaní cez OpenAPI môže spotrebiteľ porozumieť vzdialenej službe a interagovať s ňou s minimálnym množstvom implementačnej logiky.[45]

Vďaka tomu, že formát OpenAPI má pevne danú štruktúru ako by mal vyzerať, tak môžme generáciu takejto dokumentácie automatizovať. Automatické generovanie môže prebiehať dvomi spôsobmi. Prvý spôsob je vo formáte PHPDoc anotácií nad každou metódou API radičov a nad modelmi, ktoré posielame cez API. Tento spôsob vedie k častej duplicite informácií, ktoré sú zároveň v kóde, aj v anotáciách nad ním.

Ďalším automatickým spôsobom ako vygenerovať dokumentáciu k API je pomocou statickej analýzy kódu. Vďaka tomu, že Laravel má pevne definovanú štruktúru na definovanie koncových bodov API, tak sa táto statická analýza dá jednoducho spraviť. Pri generácii API dokumentácie som využil nástroj Scramble[46], ktorý presne týmto spôsobom vytvára API dokumentáciu.

3.3 Frontend

V tejto časti sa budem ďalej venovať technológiám použitých pri vývoji frontendu podľa návrhu.

3.3.1 Framework

Na základe návrhu spraviť Frontend ako jednostránkovú aplikáciu, mi zostali na výber frameworky alebo jazyky, ktoré sa interpretujú, alebo kompilujú do JavaScriptu. Nakoľko na dosiahnutie správania jednostránkovej aplikácie je potreba, aby sa obsah menil dynamicky v prehliadači používateľa, na čo slúži práve jazyk JavaScript.

Keďže napísanie celej aplikácie v čistom JavaScripte by bolo prílišne náročné rozhodol som sa zvoliť framework na uľahčenie tejto práce. Pri výbere fremworku som zohľadnil hodnotenia frontendových frameworkov, z prieskumu State of JavaScript 2022.[47] Tento prieskum prebieha každoročne od roku 2016 s cieľom zistiť používanie jazyku JavaScript a prípadných frameworkov, alebo nástrojov na prácu s týmto jazykom. V dobe písania tejto bakalárskej práce sú najnovšie zverejnené výsledky z roku 2022.

V kategórií frontendových frameworkov a knižníc [48] som sa zameral predovšetkým na použitie a spokojnosť používateľov pri používaní daných frameworkov. Tieto metriky som si vybral s ohľadom na to, že čím je jazyk frameworku používanejší, tak existuje viac knižníc na riešenie rôznych problémov a spokojnosť používateľov pri používaní som si vybral aby sa mi v ňom následne príjemne pracovalo.

Z hľadiska použiteľnosti vzišli traja kandidáti React, Angular a Vue.js, ktoré dosiahli aspoň 25% v tejto kategórií. Na základe spokojnosti používateľov pri používaní sa React a Vue.js pohybovali tesne nad 75% a Angular v tejto metrike oproti ním zaostával iba na úrovni 42,7%. Ďalej sa teda budem zaoberať primárne porovnaním Reactu a Vue.js.

React je JavaScript knižnica na prácu s UI prvkami, zatiaľ čo Vue.js je samostatný framework obsahujúci JavaScript a HTML šablóny. Tento rozdiel robí Vue.js tradičnejší pri používaní. Keďže som nemal žiadne skúsenosti s ani jednou z týchto dvoch technológií, tak som sa ďalej pozrel na ich krivku učenia sa, aby som zhodnotil, ktorú technológiu sa rýchlejšie naučím. Z porovnaní technológií mi vzišlo, že React má strmejšiu krivku učenia sa, tým pádom je ťažšie sa ho naučiť. Hlavným dôvodom na túto strmejšiu krivku je rozdiel v šablónovej syntaxi. React využíva JSX syntax v JavaScripte a HTML, čo ho robí náročnejší na pochopenie, najmä pre začiatočníkov. Na druhej strane, Vue využíva syntax, ktorá je viac podobná tradičnému HTML, čo z neho robí jednoduchší pre začiatočníkov na pochopenie [49]. Toto ma viedlo k výberu Vue.js technológie na implementáciu Frontendu.

3.3.2 Lokalizácia

Pri počiatku implementácie som vychádzal z možností, že by do budúcna pribudli preklady aplikácie aj do iných jazykov. Rozhodol som sa preto využiť knižnicu Vue I18n [50], ktorá umožňuje jednoduchým spôsobom definovať preklady pre rôzne miestne nastavenia a jednoduchou zmenou sa prepnú všetky názvy z jedného jazyka do ďalšieho jazyka.

Preklady som sémanticky rozdelil do rôznych súborov, ktoré som následne importoval v hlavnom súbore. Dosiahol som týmto väčšiu prehľadnosť v rámci definícií prekladov a zároveň mi to umožnilo lepšie rozlišovať názvy, ktoré sú v rôznych jazykoch zastúpene inými slovami, napríklad meno osoby a názov produktu.

Okrem vlastných prekladov som knižnicu Vue I18n použil na preloženie chybových hlášok z výstupu validácie a na nastavenie prekladov obsiahnutých v rámci knižnice Vuetify.

3.3.3 Validácia

Aplikácia potrebuje na správne fungovanie vstup od používateľa ako pri operáciach na vytváranie, tak aj pri upravovacích operáciach. Na zaistenie kontroly tohto vstupu, aby spĺňal definované pravidlá, slúži validácia vstupu. Na validáciu vstupu na úrovni frontendu som použil knižnicu Vuelidate.[51] Knižnica Vuelidate umožňuje definovať pravidlá na validáciu vstupu priamo pri modeloch, ktorých sa týkajú.

Na podporu lokalizácie aplikácie obsahuje knižnica Vuelidate obal na pravidlá, ktorý pomocou knižnice Vue I18n dodá preklady k potrebným pravidlám. Preložené chybové hlášky v prípade, že validácia nebola úspešná, sú následne dostupné na zobrazenie v komponente vstupov.

3.3.4 Router

Smerovanie na strane klienta sa používa v jednostránkových aplikáciách (SPA) na prepojenie URL adresy v prehliadači s obsahom, ktorý vidí používateľ. Pri navigácii používateľov po aplikácii sa URL adresa aktualizuje podľa potreby, ale stránka sa nemusí znovu načítať zo servera. [52]

Na smerovanie som použil Unplugin Vue Router [53], ktorý je rozšírením pôvodného Vue Router. Toto rozšírenie umožňuje smerovanie založené na súboroch, ktoré funguje na princípe adresáru komponentov nazvaný pages, v ktorom sú uložené komponenty stránok, ktoré sa majú načítať. Týmto spôsobom si uľahčíme definovanie každej cesty v rámci URL. Príklad ciest definovaných v aplikácií:

```
pages ..... adresár obsahujúci definíciu všetkých ciest
├── customers ..... adresár pre cesty týkajúce sa zákazníkov
│   ├── [id].vue ..... cesta na profil zákazníka podľa jeho id
│   └── index.vue ..... cesta na zobrazenie prehľadu o všetkých zákazníkoch
├── orders ..... adresár pre cesty ohľadom objednávok
│   ├── [id].vue ..... cesta na danú objednávku
│   └── index.vue ..... cesta na správu všetkých objednávok používateľa
├── products ..... adresár pre cesty týkajúce sa produktov
│   ├── admin ..... adresár ciest na správu produktov
│   │   ├── [id] ..... adresár ciest na správu produktu podľa jeho id
│   │   │   ├── questions ..... adresár ciest na správu otázok produktu
│   │   │   │   └── [questionsId].vue ..... cesta na správu otázky podľa jej ID
│   │   │   ├── configurations.vue ..... cesta na správu cenových modelov produktu
│   │   │   ├── details.vue ..... cesta na správu detailných informácií produktu
│   │   │   ├── index.vue ..... cesta na správu základných informácií produktu
│   │   │   ├── input.vue ..... cesta na správu konfiguračného formuláru produktu
│   │   │   └── store.vue ..... cesta na správu karty produktu na hlavnej obrazovke
│   │   └── index.vue ..... cesta na správu všetkých produktov
│   └── [slug].vue ..... cesta k produktu podľa jeho slugu
├── [...path].vue ..... cesta na zachytenie všetkých neznámych ciest
├── forgot-password.vue ..... cesta na obnovu zabudnutého hesla
├── index.vue ..... hlavná stránka s prehľadom všetkých verejných produktov
└── invoices.vue ..... cesta na prehľad všetkých faktúr používateľa
```

Obr. 3.1 Štruktúra adresáru na definovanie ciest

Pokiaľ chceme mať dynamicky udávanú cestu, symbolizovanú nejakým parametrom tak názov toho parametru dáme do hranatých zátvoriek pri pomenovaní danej cesty.

3.3.5 Vuetify

Pri implementácií frontendu som zvolil knižnicu Vuetify [54]. Táto knižnica obsahuje komponenty, z ktorých sa následne jednoduchšie poskladá výsledný výzor aplikácie. Komponenty v rámci tejto knižnice sa držia Material Designu. Spolu s komponentami obsahuje táto knižnica aj preklady na text v komponentoch. Na použitie týchto prekladov je potreba tieto preklady zaregistrovať v rámci I18n inštancie a pridať I18n adaptér. Knižnica Vuetify zároveň umožňuje jednoduchú definíciu témy aplikácie. V rámci použitia v mojej bakalárskej práce som sa rozhodol využiť predvolenú svetlú a tmavú tému.

3.3.6 API klient

Na odosielanie požiadaviek na Backend potrebujeme v aplikácií napísať klienta API, ktorý nám bude sprostredkovávať všetky požiadavky posielané na Backend API. Na začiatku implementácie som sa rozhodol tohto klienta vygenerovať automaticky z OpenAPI dokumentácie Backendu pomocou nástroju openapi [55]. Výstupom z tohto nástroja boli tri všeobecné súbory na podporu API operácií a jeden dlhý súbor obsahujúci všetky požiadavky. Toto riešenie je, ale omnoho náročnejšie na udržiavanie. Z tohto dôvodu som sa rozhodol prerobiť API klienta ručne s využitím funkcie useFetch [56].

3.3.7 Znovupoužitie alertu

Na zobrazovanie spätnej väzby som využil komponentu Alert z knižnice Vuetify. Alert z knižnice Vuetify obsahuje štyri druhy štýlov ako sa môže komponent zobraziť: úspech, chyba, upozornenie a informovanie. Tieto druhy sa líšia vo farbe zobrazeného Alertu a v ikonke pred textom. V mieste, v ktorom nastala udalosť, som textový popis tejto udalosti, identifikátor Alertu, kde sa má zobraziť, a typ udalosti uložil do globálneho stavu. Z tohto globálneho stavu si následne všetky Alert komponenty načítali túto udalosť a iba Alert so správnym identifikátorom ju zobrazil.

Kapitola 4
Testovanie aplikácie

Táto kapitola sa venuje testovaniu aplikácie. Z úvodu bude vysvetlené na čo je testovanie vlastne dobré. Následne budú predstavené spôsoby, akými môžme aplikácie testovať.

4.1 Dôvody na testovanie a obmedzenia spojené s ním

Testovanie aplikácií nám slúži, ako overovanie kvality danej aplikácie. Pri testovaní overujeme, či aplikácia spĺňa všetky požiadavky, ktoré na ňu boli na začiatku kladené. Testovaním skúšame, či je aplikácia funkčná, či pri niektorých operáciach nespadne, alebo sa nepoškodí. Okrem toho môže testovanie slúžiť aj na overenie, či sa aplikácia nedá zneužiť na prístup k citlivým údajom. Obmedzuje chyby a problémy ešte pred vypustením do produkcie. Kompletne otestovať aplikáciu je prakticky nemožné a nezaručuje, že aplikácia je bez chýb, ale značne znižuje riziko výskytu chýb[57].

4.2 Druhy testov

Existuje veľa typov techník testovania softvéru, ktoré možno použiť na zabezpečenie toho, aby zmeny v kóde fungovali podľa očakávania.

Je dôležité rozlišovať medzi manuálnymi a automatickými testami. Manuálne testovanie sa vykonáva osobne, preklikaním sa cez aplikáciu, alebo interakciou so softvérom a API s príslušnými nástrojmi.

Na druhej strane automatizované testy vykonáva stroj, ktorý vykonáva testovací skript, ktorý bol napísaný vopred. Tieto testy sa môžu líšiť v zložitosti, od kontroly jednej metódy v triede, až po uistenie sa, že vykonávanie sekvencie zložitých akcií v používateľskom rozhraní vedie k rovnakým výsledkom. Je to oveľa robustnejšie a spoľahlivejšie ako manuálne testy, ale kvalita automatických testov závisí od toho, ako dobre boli napísané testovacie skripty.

Jednotkové testy sú veľmi detailné a blízke k zdroju aplikácie. Spočívajú v testovaní jednotlivých metód a funkcií tried, komponentov, alebo modulov používaných softvérom.

Integračné testy overujú, či rôzne moduly, alebo služby používané aplikáciou dobre spolupracujú. Môže to byť napríklad testovanie interakcie s databázou, alebo uistenie sa, že mikroslužby spolupracujú podľa očakávania. Spustenie týchto typov testov je náročnejšie, pretože vyžadujú spustenie viacerých častí aplikácie.

Funkčné testy sa zameriavajú na obchodné požiadavky aplikácie. Overujú len výstup určitej akcie a nekontrolujú medzistavy systému počas vykonávania tejto akcie.

Niekedy dochádza k zámene medzi integračnými testami a funkčnými testami, pretože oba vyžadujú interakciu viacerých komponentov medzi sebou. Rozdiel spočíva v tom, že integračný test môže jednoducho overiť, či je možné vykonávať dotazy do databázy, zatiaľ čo funkčný test by očakával, že z databázy získa konkrétnu hodnotu, ako je definovaná požiadavkami na produkt.

Testovanie od začiatku do konca replikuje správanie používateľa so softvérom v kompletnom prostredí aplikácie. Overuje, či rôzne používateľské toky fungujú podľa očakávania a môže ísť o jednoduché činnosti, ako načítanie webovej stránky, prihlásenie sa, alebo o oveľa zložitejšie scenáre, ktoré overujú emailové upozornenia, online platby atď..

Testy od začiatku do konca sú veľmi užitočné, ale sú nákladné na vykonávanie a môžu byť ťažko udržateľné, ak sú automatizované. Odporúča sa mať niekoľko kľúčových testov od začiatku do konca a viac sa spoľahnúť na testovanie na nižšej úrovni (jednotkové a integračné testy), aby bolo možné rýchlo identifikovať zmeny, ktoré spôsobujú problémy.

Akceptačné testy sú formálne testy, ktoré overujú, či systém spĺňa obchodné požiadavky. Vyžadujú, aby počas testovania bola spustená celá aplikácia a sú zamerané na replikáciu správania používateľov. Avšak môžu ísť aj ďalej a merať výkonnosť systému a odmietnuť zmeny, ak nie sú splnené určité ciele.

Výkonnostné testy vyhodnocujú, ako sa systém správa pri konkrétnej pracovnej záťaži. Tieto testy pomáhajú merať spoľahlivosť, rýchlosť, škálovateľnosť a odozvu aplikácie. Výkonnostné testy môžu napríklad sledovať časy odozvy pri vykonávaní veľkého počtu požiadaviek, alebo určiť, ako sa systém správa pri veľkom množstve dát. Môže určiť, či aplikácia spĺňa požiadavky na výkon, lokalizovať úzke miesta, merať stabilitu počas špičkového zaťaženia a ďalšie.

Dymové testy sú základné testy, ktoré kontrolujú základnú funkčnosť aplikácie. Sú určené na rýchle vykonanie a ich cieľom je poskytnúť istotu, že hlavné funkcie systému fungujú podľa očakávania. [58]

4.3 Používateľské testovanie

Na otestovanie použiteľnosti a funkčnosti aplikácie som zvolil manuálne neriadené používateľské testovanie, ktoré je formou testovania od začiatku do konca. Týmto používateľským testovaním som sa pokúšal nasimulovať situácie, keď zákazník navštívi aplikáciu, po prvýkrát a interaguje s ňou bez akejkoľvek predchádzajúcej skúsenosti. Pri tomto používateľskom testovaní som nechal respondenta neriadene prechádzať aplikáciou a zaznamenával som jeho všeobecný pocit z aplikácie.

Pri výbere osôb, ktoré by testovali aplikáciu som sa snažil obsiahnuť, čo najväčšie spektrum možných používateľov.

4.3.1 Výsledky používateľského testovania

- Muž, 69 rokov, pravidelný používateľ počítača

  Aplikácia bola hodnotená všeobecne kladne, s rýchlou odozvou na vstup používateľa. Avšak počas testovania vzniklo niekoľko chaotických momentov, ktoré viedli k zmätku testujúceho respondenta.

  Na začiatku bolo náročné zorientovať sa v rámci aplikácie. Najmä s úplne skrytým bočným menu, prístupným cez hamburgerovú ikonu.

  Pri konfigurovaní objednávky došlo ku nepochopeniu tlačidla na zrušenie objednávky, pri tomto nepochopení došlo ku zrušeniu objednávky, ktorá sa už naďalej nezobrazovala medzi

objednávkami používateľa. V rámci objednávky ďalej chýbal respondentovi deň, v ktorom bola táto objednávka vytvorená, na bližšiu identifikáciu objednávky.

V rámci odoslanej objednávky splýval stav objednávky a stav faktúry pri zobrazení iba jednej faktúry vystavenej k tejto odoslanej objednávke.

Najväčším problémom, ktorý prekážal respondentovi, bola chýbajúca možnosť na úpravu a prehľad fakturačných údajov, prístupných cez tlačidlo s menom používateľa nachádzajúcim sa v pravom hornom rohu aplikácie.

- Muž, 24 rokov, pravidelný používateľ počítača

  Aplikácia bola hodnotená taktiež kladne respondentom. Pri pozorovaní respondenta som zaregistroval silný zmätok, v rámci administrácie produktu, ktorý vychádzal z neznalosti častí produktu. Ďalším bodom bolo chýbajúce upozornenie o neuložených údajoch pri úprave produktu. Po upozornení na neuloženie údajov, respondent namietal polohu tlačidla na uloženie, ktoré podľa neho bolo zle umiestnené na pravej strane obrazovky, zatiaľ čo je väčšina prvkov zarovnaná na ľavú stranu, čo mohlo viesť k prehliadnutiu tohto tlačidla.

  Tester ďalej prešiel ku skúšaniu používateľských vstupov, čím odhalil chýbajúcu spätnú väzbu pri požiadavkách na backend.

  Nastavenie minimálnej a maximálnej medze otázky, vzhľadom ku nadradenej entite, bolo náročne na nájdenie po svojvoľnej osi, na rozdiel od ostatných súčastí produktu. Po úspešnom nájdení medzí nastalo však zmätenie respondenta pri tvorbe premenných produktu.

- Muž, 23 rokov, pravidelný používateľ počítača

  Respondentovi v rámci aplikácie chýbalo vysvetlenie komplikovanejšieho objektu produktu pri jeho nastavovaní. Pri práci s tabuľkami v rámci aplikácie kladne ocenil možnosť všetkých dát z tabuľky, namiesto nejakého umelého obmedzenia počtu položiek v tabuľke.

  Celkový dojem z aplikácie bol kladný, dizajn hodnotil respondent pozitívne a ďalej sa mu páčila responzívnosť a jednoduchosť aplikácie v zmysle, že sa mu v rámci aplikácie nepodarilo sa stratiť.

4.3.2 Úpravy vyplývajúce z používateľského testovania

Na základe používateľského testovania som dospel k zmene bočného menu. Bočné menu som upravil z úplne skrytého, na zobrazenie bočnej koľajnice v prípade kedy by malo toto menu byť skryté.

Zrušenie objednávky som zmenil zo zrušenia bez akéhokoľvek ďalšieho potvrdenia na vyžadovanie potvrdenia na zrušenie objednávky.

4.4 Akceptačné testovanie

Akceptačné testovanie prebehlo s Jiřím Hunkou formou osobného stretnutia. Akceptačné testovanie prebiehalo formou prezentácie funkčností aplikácie. Počas prezentácie boli kladené rôzne prípady použitia a ich prípadná realizácia v rámci aplikácie. Pri akceptačnom testovaní sme zároveň prechádzali všeobecné fungovanie aplikácie. Z akceptačného testovania vzišli ešte viaceré požiadavky. Jedna z požiadaviek bol na obsah faktúry. Ostatnou požiadavkou bolo zmenšenie medzier v rámci konfiguračného formuláru objednávky. Tieto požiadavky som následne ešte zapracoval do finálnej verzie.

Kapitola 5
Budúci vývoj aplikácie

V rámci tejto kapitoly opisujem možnosti ďalšieho pokračovania tohto projektu.

Budúce smerovanie vývoju v rámci License Managera je do značnej miery ovplyvnené požiadavkami, ktoré boli identifikované počas počiatočnej analýzy projektu, no z časových dôvodov nedošlo k ich realizácií do výsledného produktu mojej bakalárskej práce. K niektorým požiadavkám, navyše opíšem možné riešenia ktoré vyplynuli počas mojej práce, no ich realizácia by bola príliš časovo náročná a uprednostnil som pred nimi realizáciu iných požiadaviek.

5.1 Backend

Jednou z vecí, ktorú by bolo dobré doriešiť v rámci ďalšieho vývoja License Managera je vyčistenie backendu od kódu ktorý slúžil na komunikáciu s používateľom, no tým že sa backend pretransformoval do role iba RESTful API tak tieto časti kódu už nie sú naďalej potrebné.

Backend by išiel ďalej rozšíriť o stránkovanie výsledkov, aby sa predišlo zdĺhavému načítaniu všetkého obsahu. Spolu so stránkovaním by bolo možné ešte dokončiť viacjazyčnosť backendu.

Ďalšou no náročnejšou možnosťou, by bolo využiť beckendové riešenie z diplomovej práce Viktora Holého, ktoré je založené na architektúre mikro-služieb. Pri zvolení tohto postupu, by však bolo nutné implementovať v rámci uvádzaného backendu požiadavky frontendu. Ďalej na použitie tohto riešenia, by bolo nutné zmeniť API klienta v rámci frontendu, ktorý aktuálne komunikuje s RESTful API na klienta, ktorý by komunikoval s GraphQL API.

Ostatným vylepšením backendu License Manageru, by bol koncový bod, na ktorom by si aplikácia po autentizácií používateľa vedela overiť, či tento používateľ má platnú licenciu na daný produkt. Následne by mu na základe odpovede od License Managera bolo možné prípadne zablokovať prístup k tej aplikácií, pri porušení licenčných podmienok.

5.2 Frontend

Pri začiatku projektu som mal ambíciu spraviť stránku produktu, z ktorej sa zákazník dozvie podrobnejšie informácie o produkte a je mu umožnené si v nej spraviť objednávku na jeden zo zverejnených cenových plánov, omnoho viac prispôsobiteľnú potrebám produktu. Napríklad, aby sa na túto stránku dali pridať obrázky produktu, podľa rozloženia, ktoré si zvolí administrátor v rámci konfigurácie produktu. Na riešenie tohto problému prispôsobovania stránky produktu, som mal viacero nápadov. Prvým bolo štruktúrovať opis detailu v jazyku HTML a na stránke ho vykresľovať. Avšak riešenie má veľa obmedzení na prípadnú úpravu.

Ďalším riešením, ktoré mi napadlo, by bola kompilácia Vue.js komponentov za behu programu, čo by umožnilo administrátorovi dosiahnuť širokú škálu možností prispôsobenia stránky produktu. Lenže riešenie bolo veľmi časovo náročné v rámci tejto bakalárskej práce.

5.3 Nerealizované požiadavky

V rámci mojej bakalárskej práce som implementovať niektoré požiadavky a z časových dôvodov som vypustil požiadavky: P2 - Zákazníkom definovaný cenový plán, P10: Manuálne zostavenie objednávky, P11: Prihlásenie pomocou sociálnych sietí, P12: Doplnenie zostavenej licencie, P14: Monitoring systému, P17: Integrácia API rozhrania služby ARES.

5.3.1 P10: Manuálne zostavenie objednávky a P12: Doplnenie zostavenej licencie

Jedným z riešení týchto požiadaviek, by bolo pridanie tlačidla na stránke objednávky, ktoré by po kliknutí preplo objednávku do stavu zostavená. Toto tlačidlo by sa zobrazovalo, iba v prípade, že by prihlásený používateľ mal rolu administrátora. Keď už je objednávka v stave zostavená, tak by sa stlačením tohto tlačidla zobrazilo modálne okno, v ktorom by bolo možné zadať prvky patriace zostavenej aplikácii.

5.3.2 P11: Prihlásenie pomocou sociálnych sietí

Možné riešenie, ktoré som navrhol v prípravnej fáze, by bolo časovo náročnejšie na nastavenie. Riešením by bolo pridanie tlačidiel na prihlásenie pomocou sociálnych sieti v rámci modálneho okna na prihlasovanie. Následne by bolo nutné nastaviť na týchto sociálnych sieťach autentizáciu aplikácie a veci s tým spojené.

5.4 Testy

V rámci svojej bakalárskej práce som neimplementoval žiadne automatické testy z časových dôvodov a spoliehal som sa iba na manuálne testovanie počas vývoja a pri používateľskom testovaní. Toto by išlo ďalej rozvíjať automatickými testami frontendu pomocou nástrojov, ako sú napríklad Selenium [59] alebo Webdriver.IO [60], prípadne nejakým ďalším nástrojom.

Kapitola 6
Záver práce

Cieľom mojej práce bolo spraviť nový frontend pre komponent License Managera v rámci širšieho systému Apps Manager. Čo by umožnilo jednoduchšie a príjemnejšie používanie komponentu License Manager pre potreby spoločnosti Jagu s. r. o..

V rámci práce bola bola prevedená analýza problematiky softvérového licencovania a predchádzajúceho vývoja. Na základe analýzy bol vypracovaný návrh riešenia. Grafický návrh prešiel viacero iterácií, počas ktorých bola doladená finálna podoba grafického rozhrania frontendu.

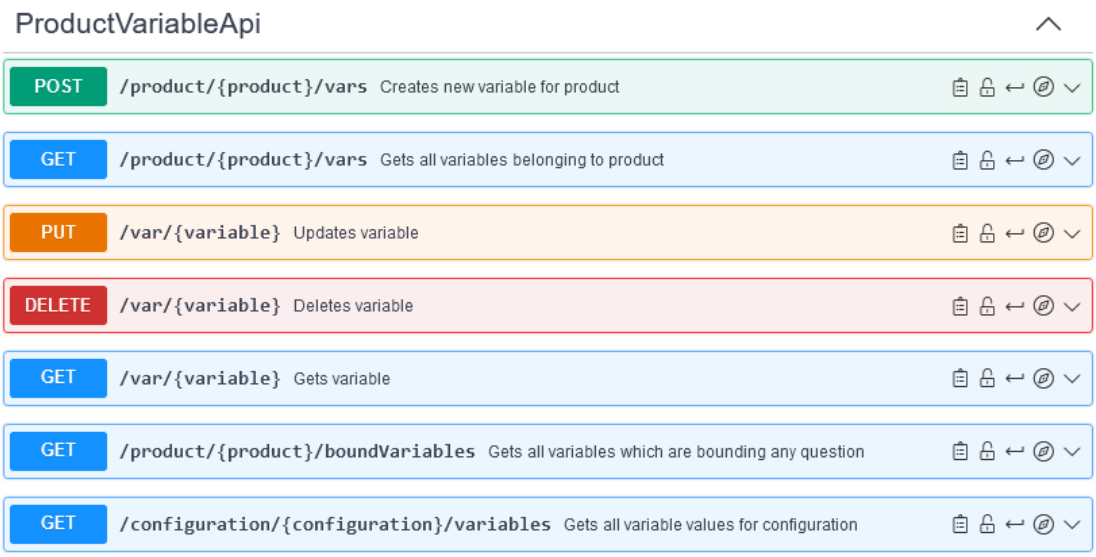
Pri implementácií bola implementovaná úplne nová frontendová aplikácia, ktorá komunikuje s backendovou aplikáciou. Frontendová aplikácia je responzívna jednostránková aplikácia, ktorá bola pri implementácií pripravovaná ako multijazyčná. Backendová aplikácia vychádzala z aplikácie, ktorá vznikla v rámci bakalárskej práce Viktora Holého, ktorá bola rozšírená o RESTful API rozhranie.

Počas vývoja aplikácie prebiehalo manuálne vývojárske testovanie, ktoré slúžilo na testovanie práve implementovaných častí. Po dokončení implementácie prebehlo neriadené používateľské testovanie. Výsledky testovania boli kladné. Spätná väzba na použiteľnosť aplikácie bola pozitívna u všetkých respondentoch.

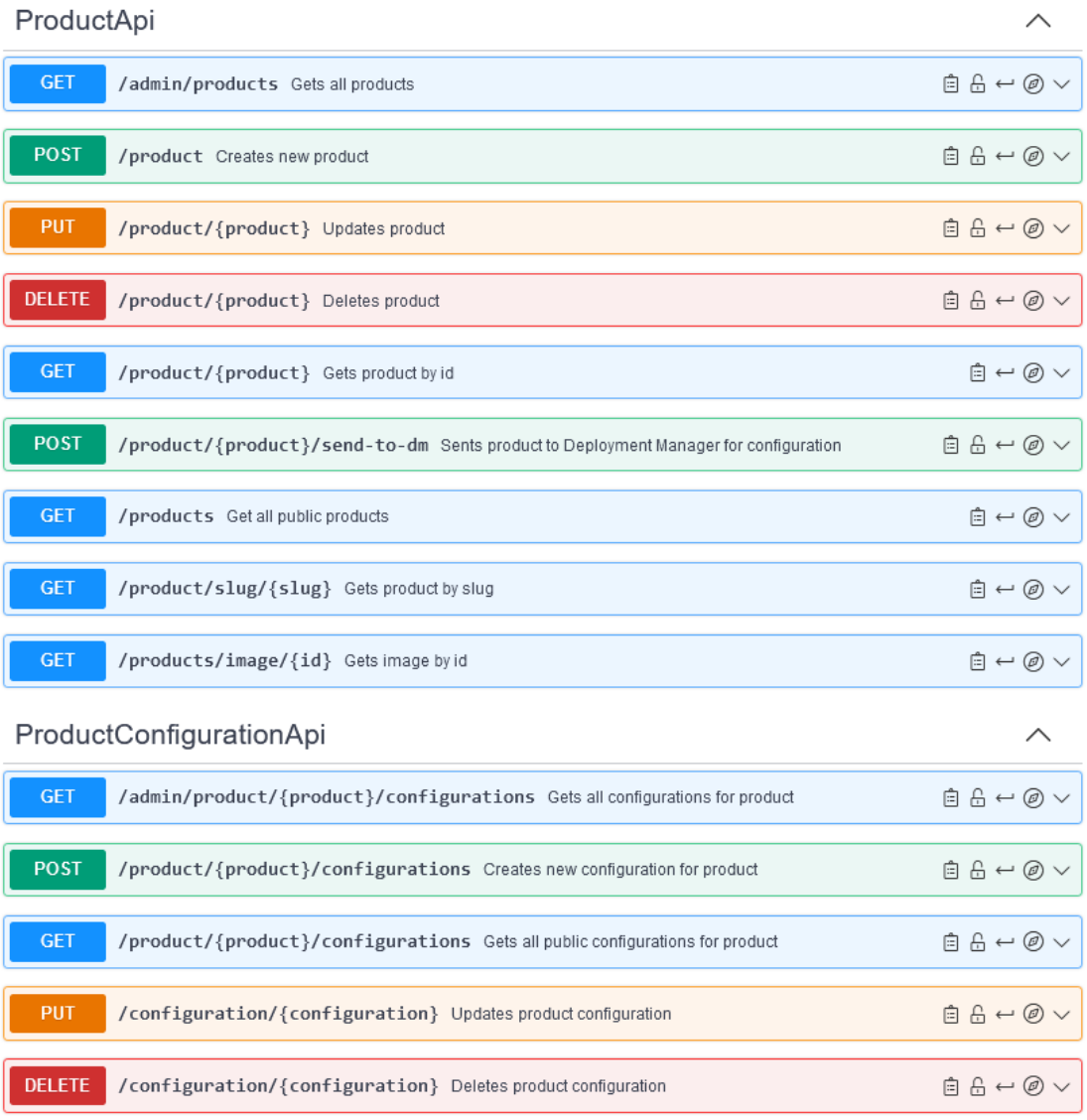
Mojím hlavným zámerom a motiváciou v rámci tejto bakalárskej práce bolo zjednodušiť používateľské rozhranie aplikácie. Do veľkej miery bol tento zámer úspešný, až na zopár požiadaviek, ktoré som nestihol zapracovať do výslednej verzie aplikácie. Na záver boli identifikované možnosti budúceho vývoja, kde boli pomenované požiadavky, ktoré sa nepodarilo realizovať počas vývoja z časových dôvodov.

Dodatok A

Koncové body API



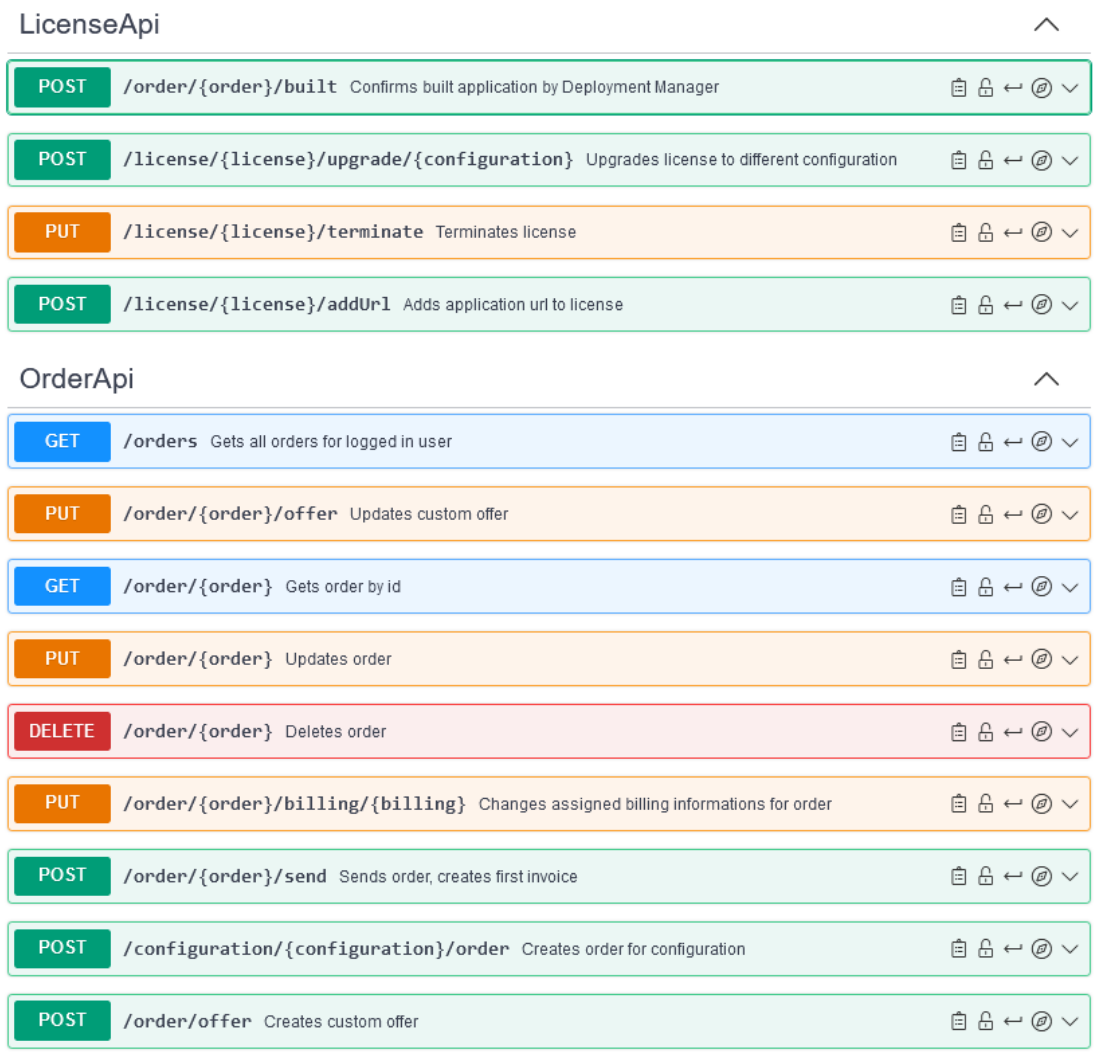
Obr. A.1 Koncové body premenných produktu



Obr. A.2 Koncové body produktu

Obr. A.3 Koncové body otázok produktu

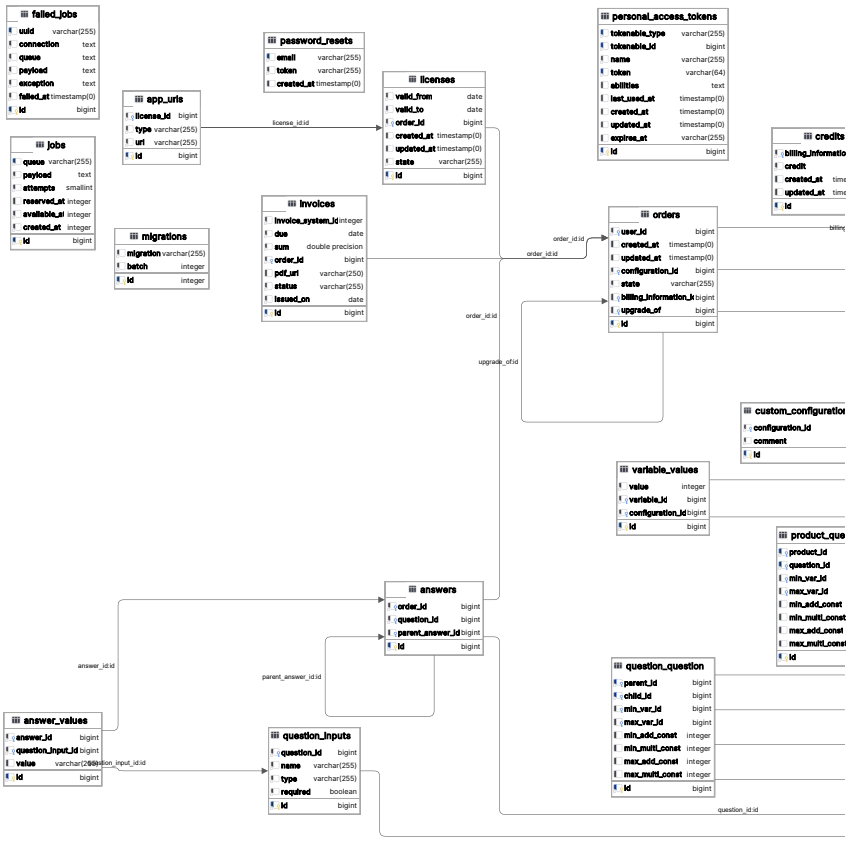
Obr. A.4 Koncové body fakturačných údajov používateľa a koncové body používateľa



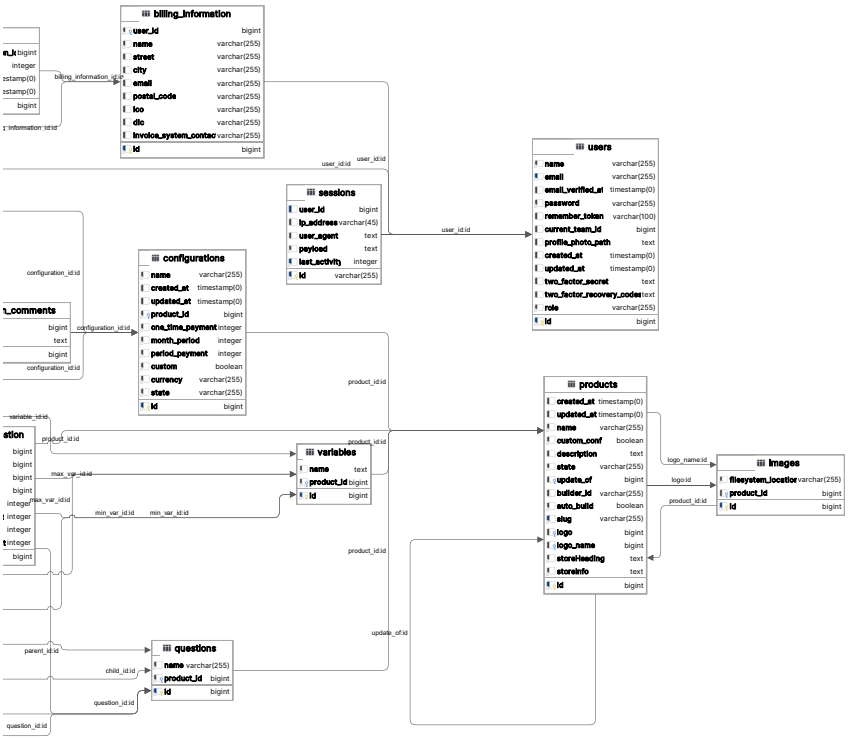
Obr. A.5 Koncové body licencií a objednávok

Dodatok B

Databázový model



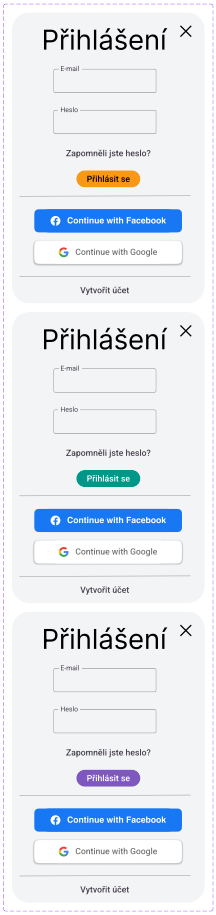
Obr. B.1 Databázový model License Manager Backendu strana 1



Obr. B.2 Databázový model License Manager Backendu strana 2

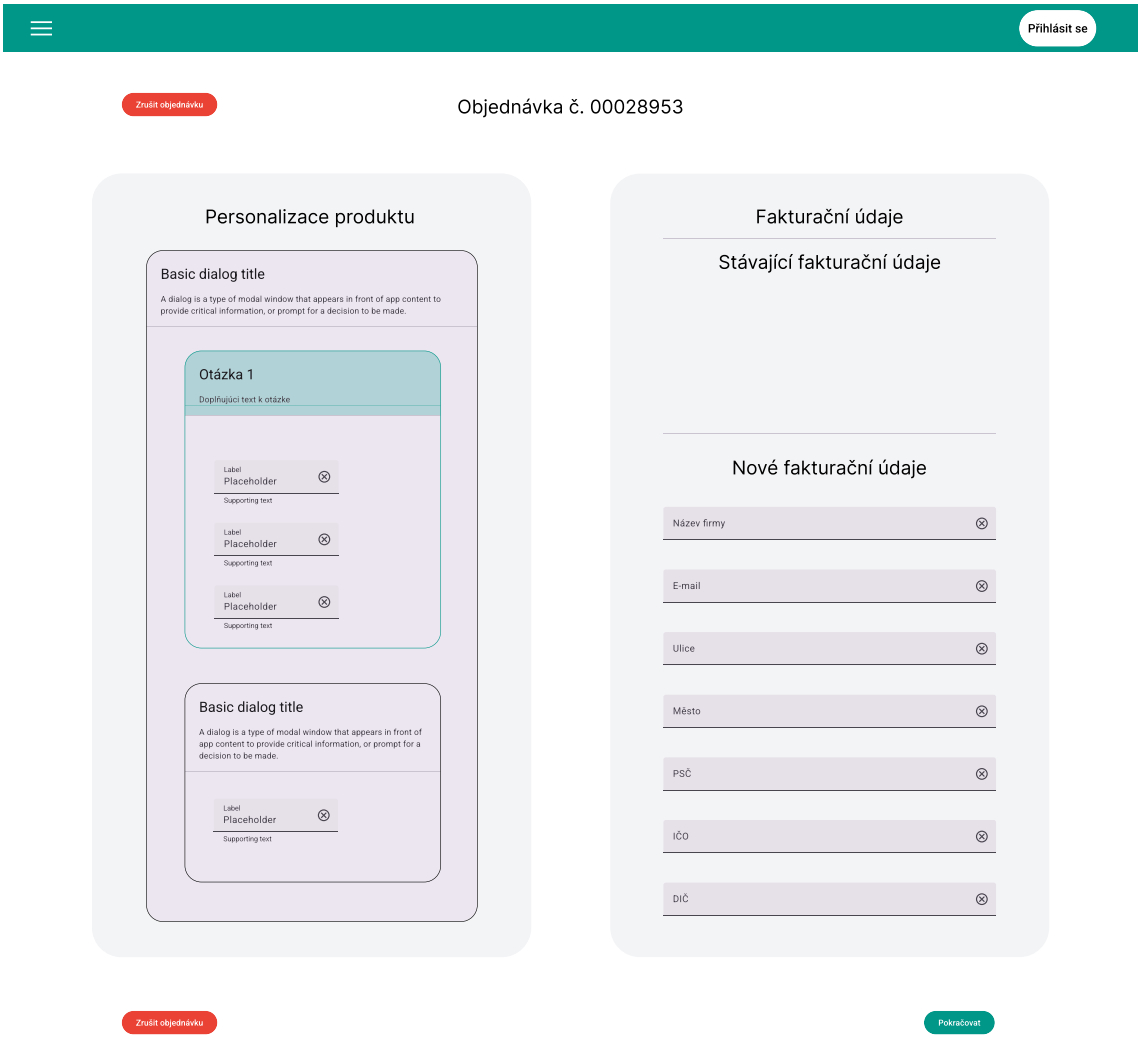
Dodatok C

Grafický návrh používateľského rozhrania - prvý model

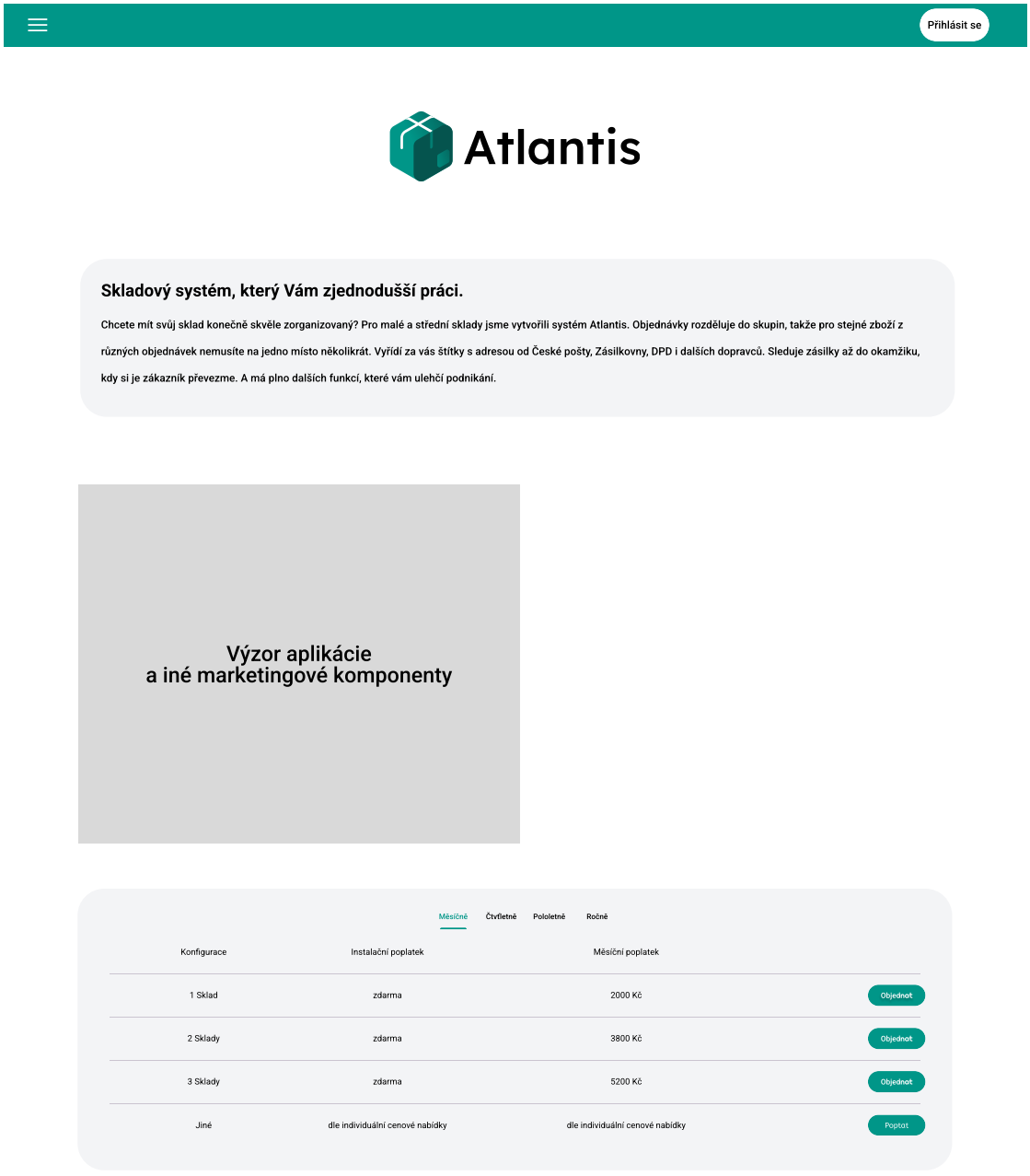


Obr. C.1 Modálne okno na prihlásenie

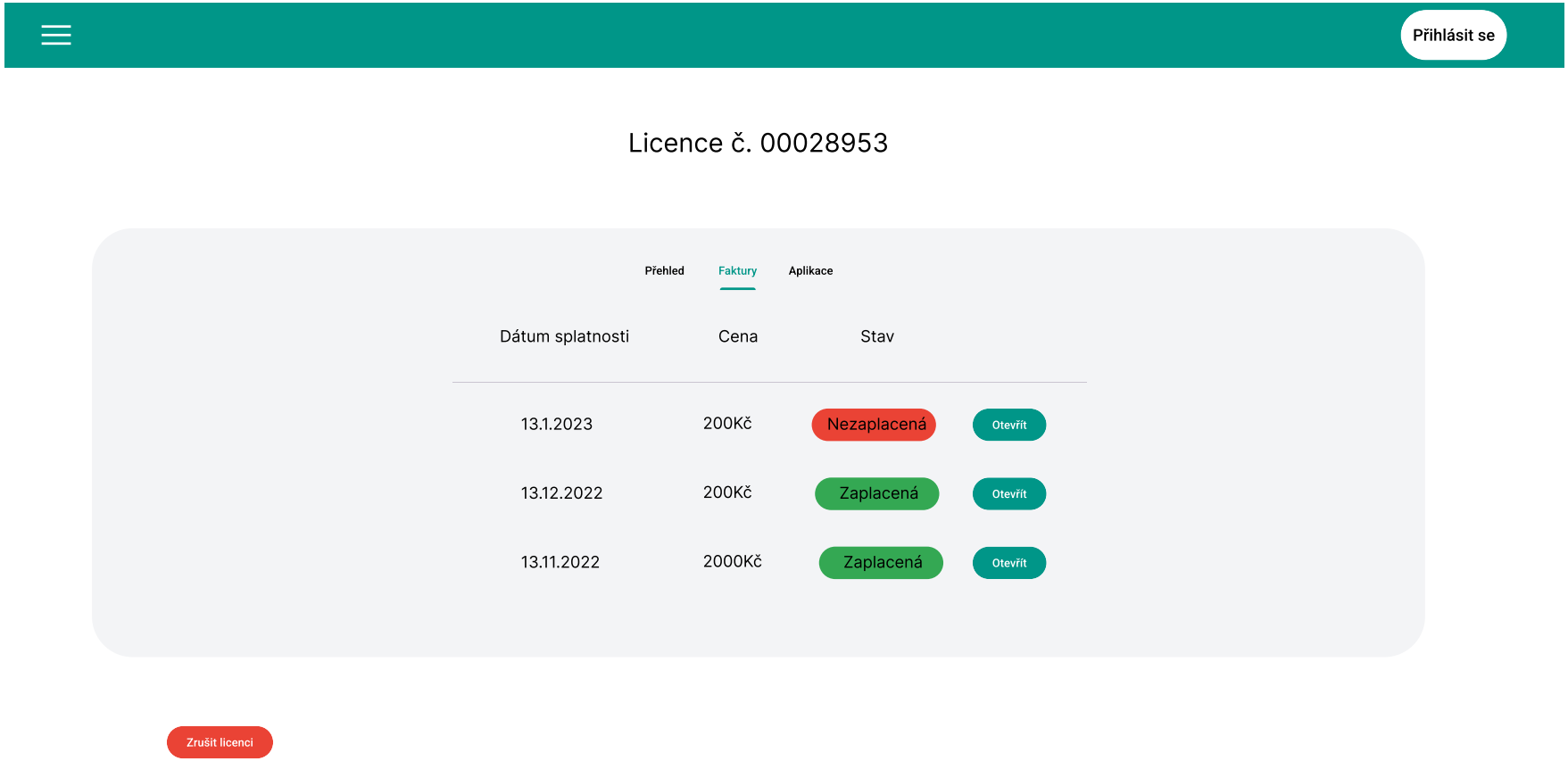
Obr. C.2 Prehľad produktov



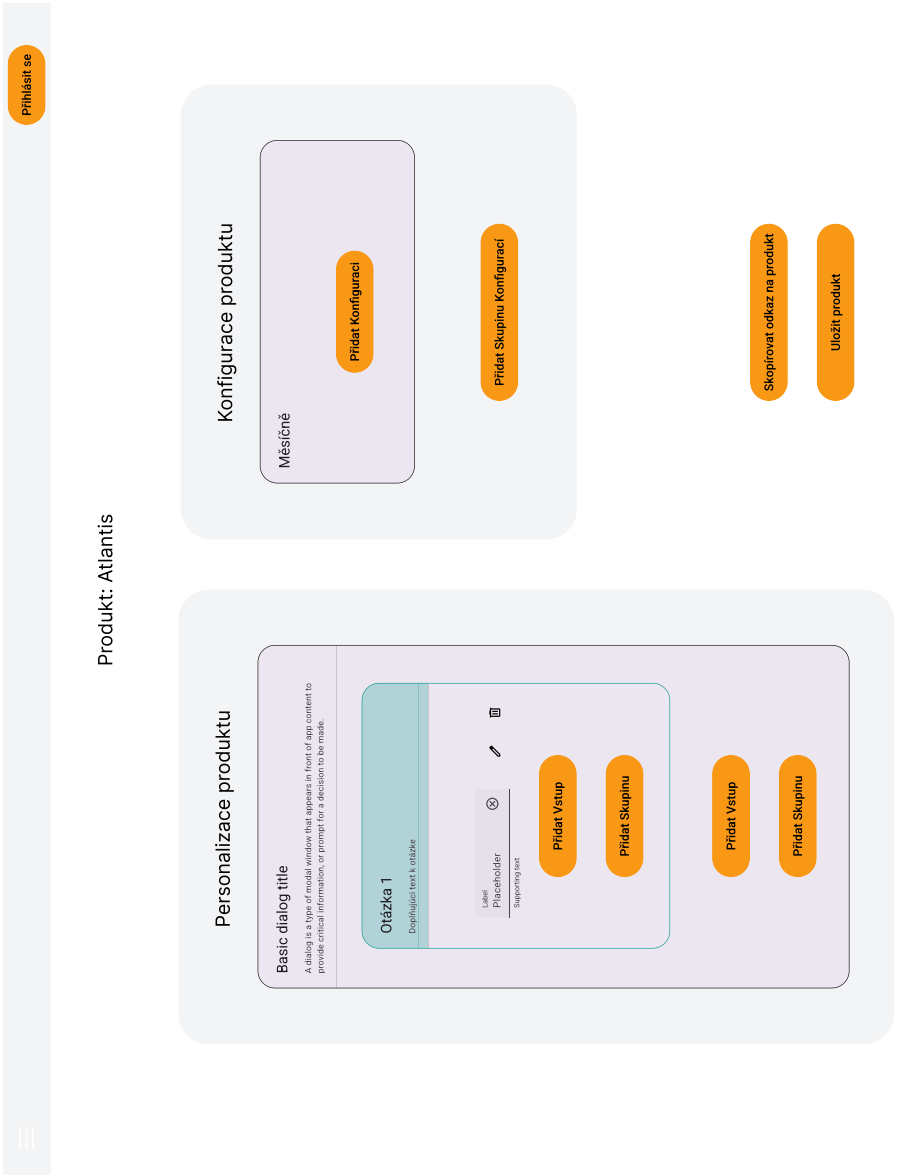
Obr. C.3 Konfigurácia objednávky



Obr. C.4 Detailná stránka produktu



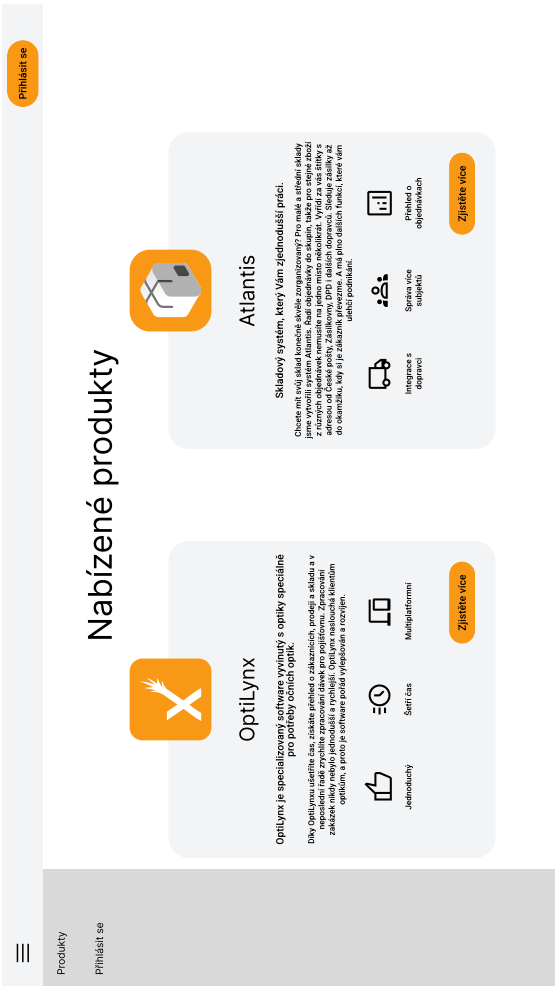
Obr. C.5 Prehľad faktúr vystavených k licencii



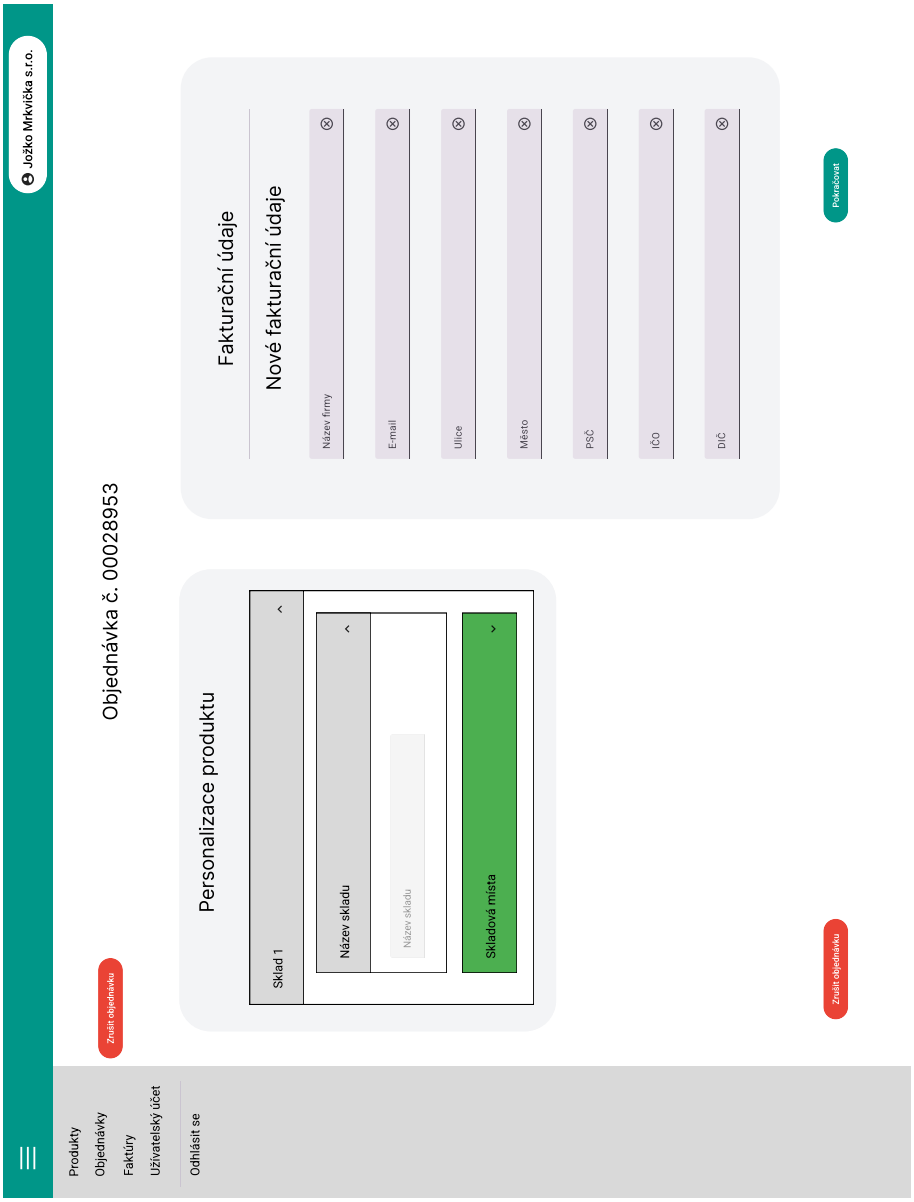
Obr. C.6 Administrácia produktu

Dodatok D

Grafický návrh používateľského rozhrania - druhý model



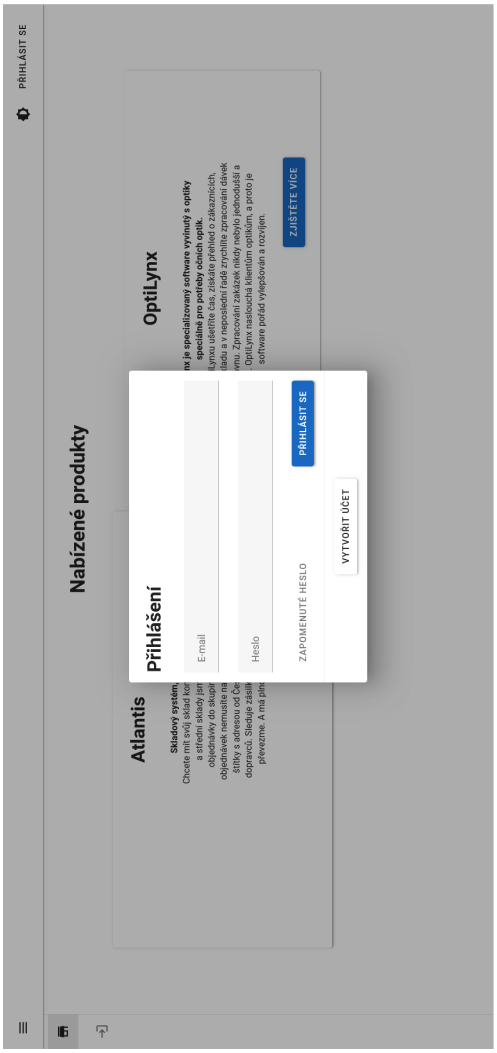
Obr. D.1 Administrácia produktu



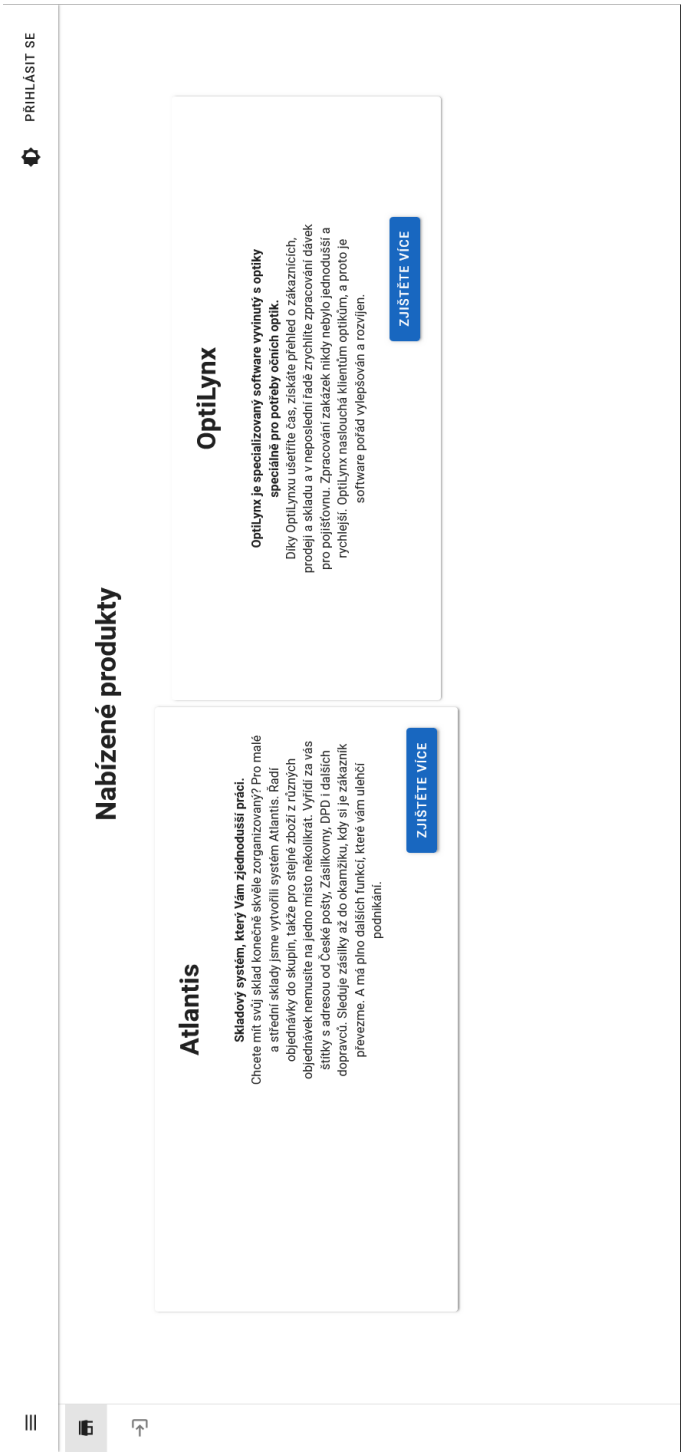
Obr. D.2 Administrácia produktu

Dodatok E

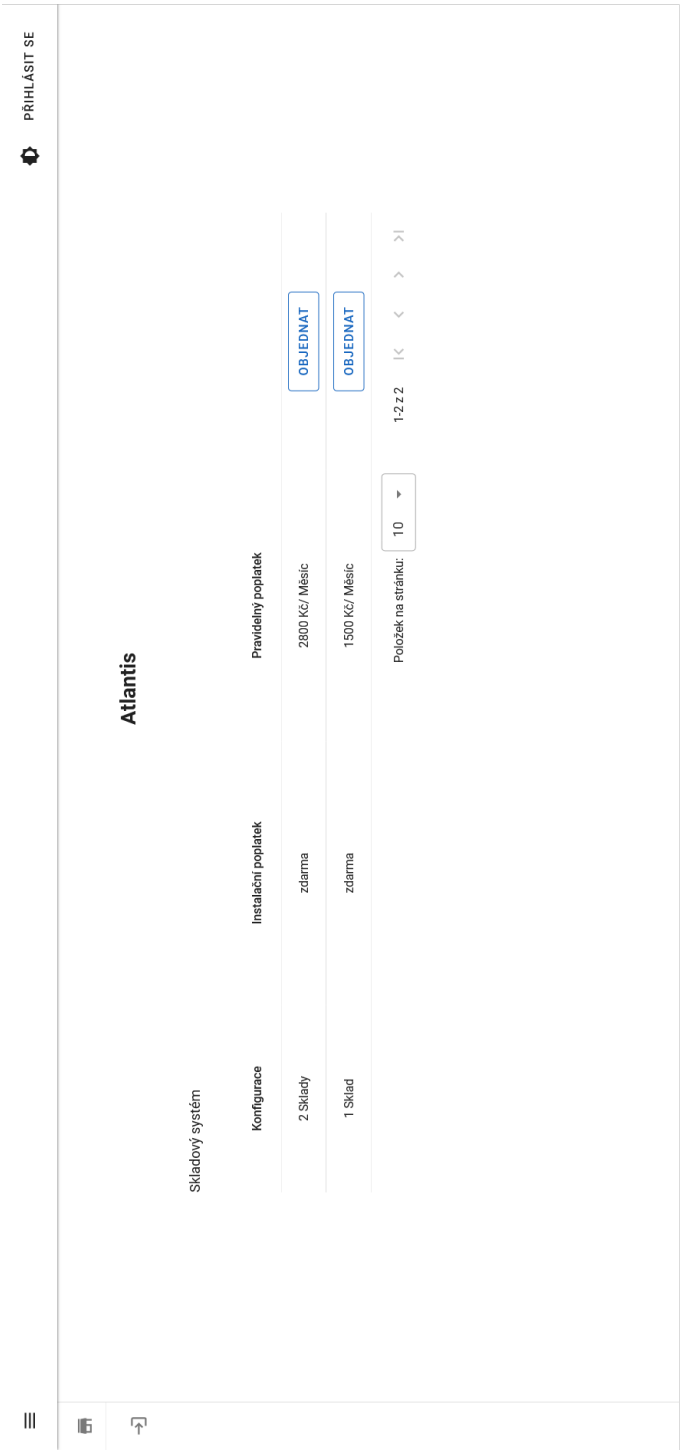
Snímky obrazovky aplikácie



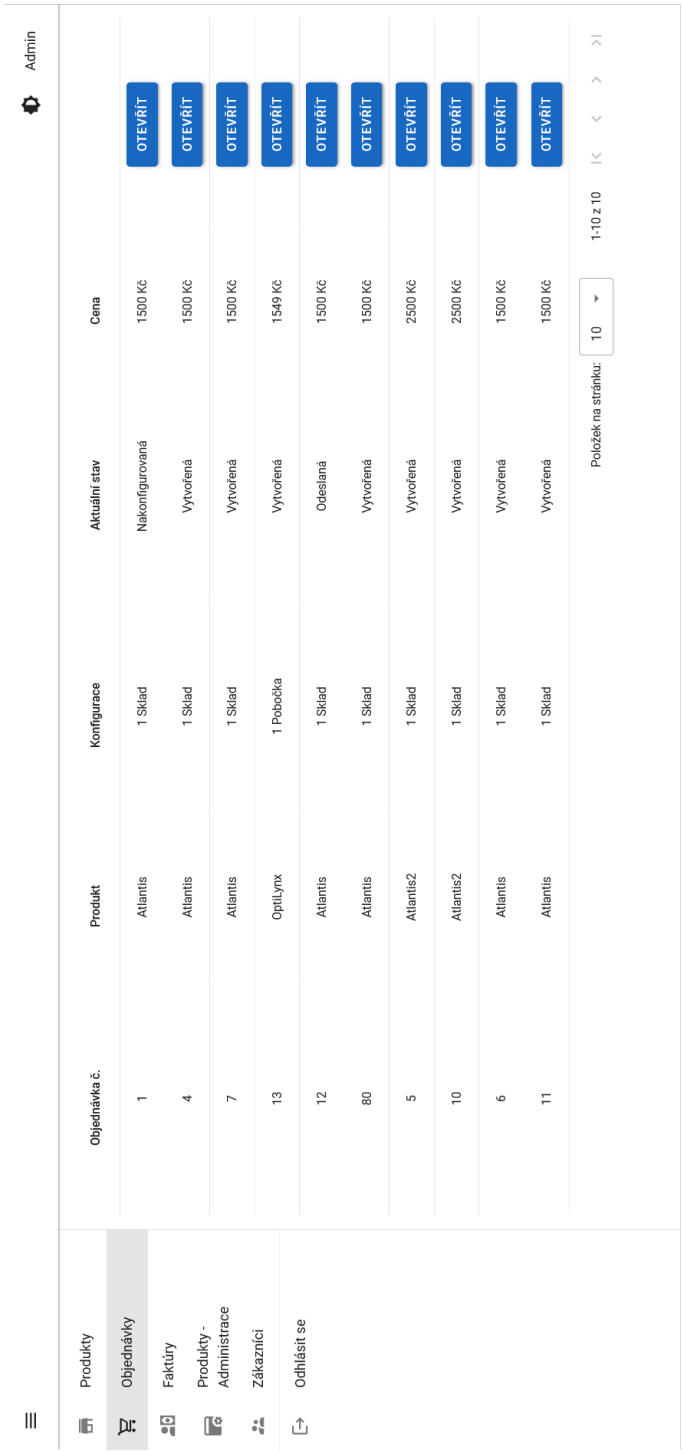
Obr. E.1 Prihlásenie používateľa



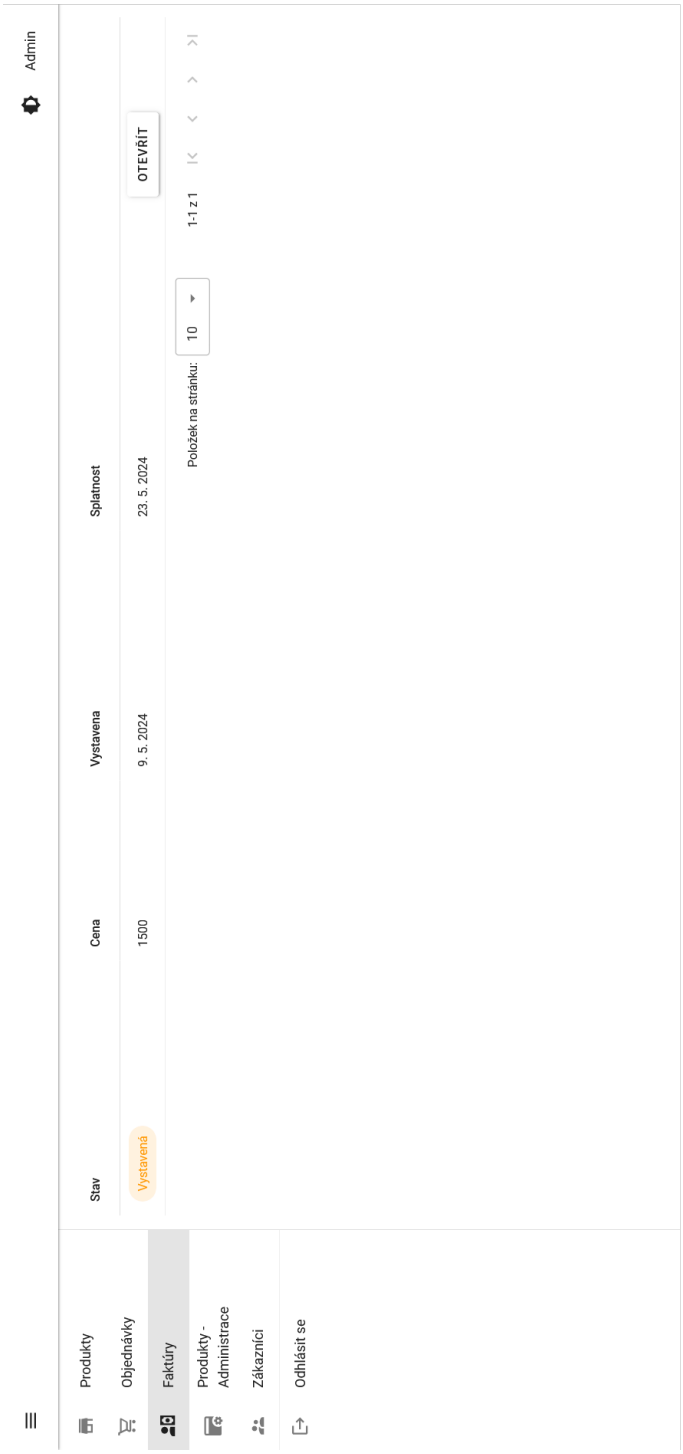
Obr. E.2 Prehľad produktov



Obr. E.3 Detail produktu

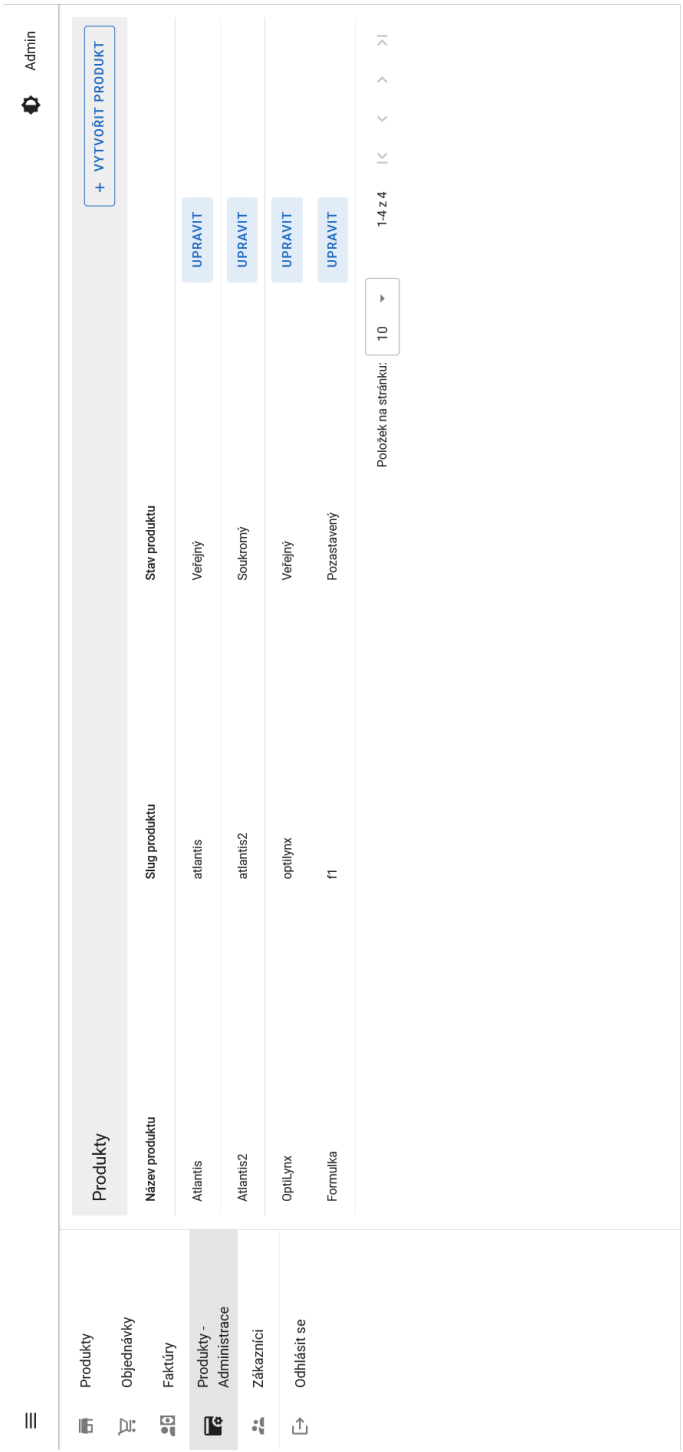


Obr. E.4 Prehľad objednávok

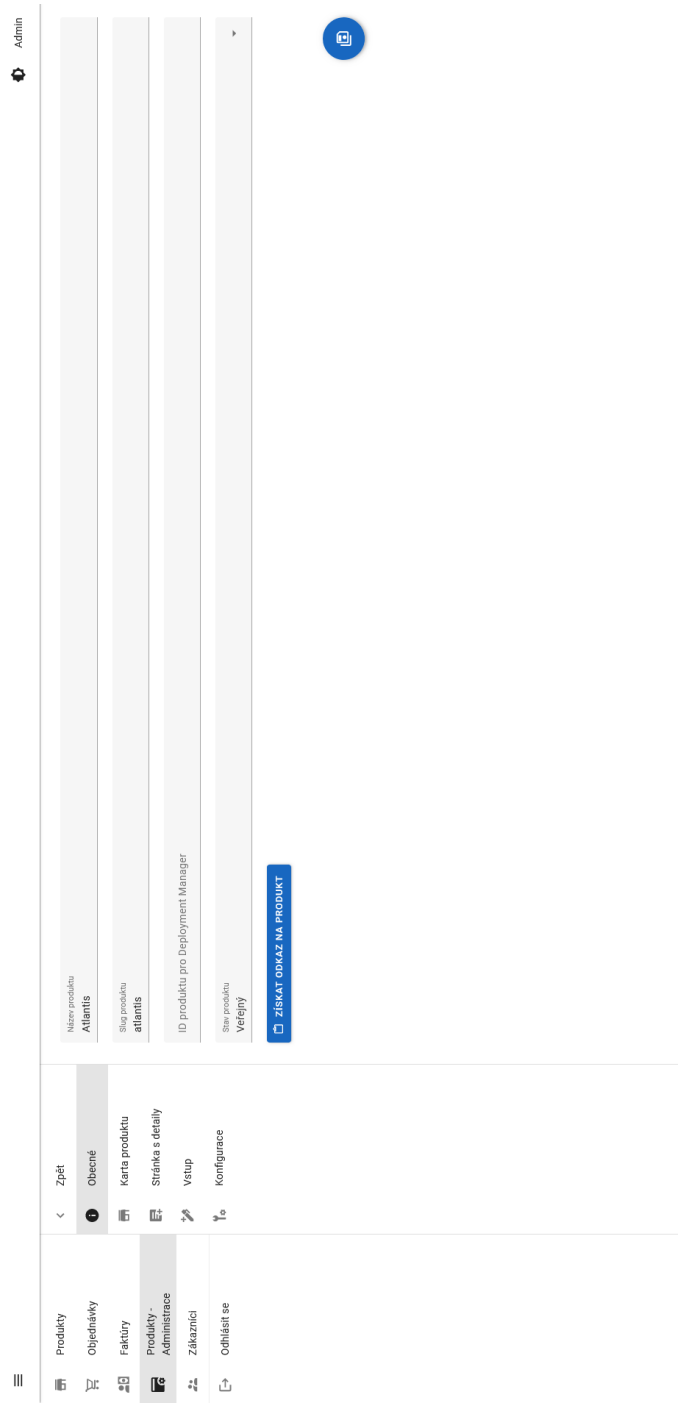


Obr. E.5 Prehľad faktúr

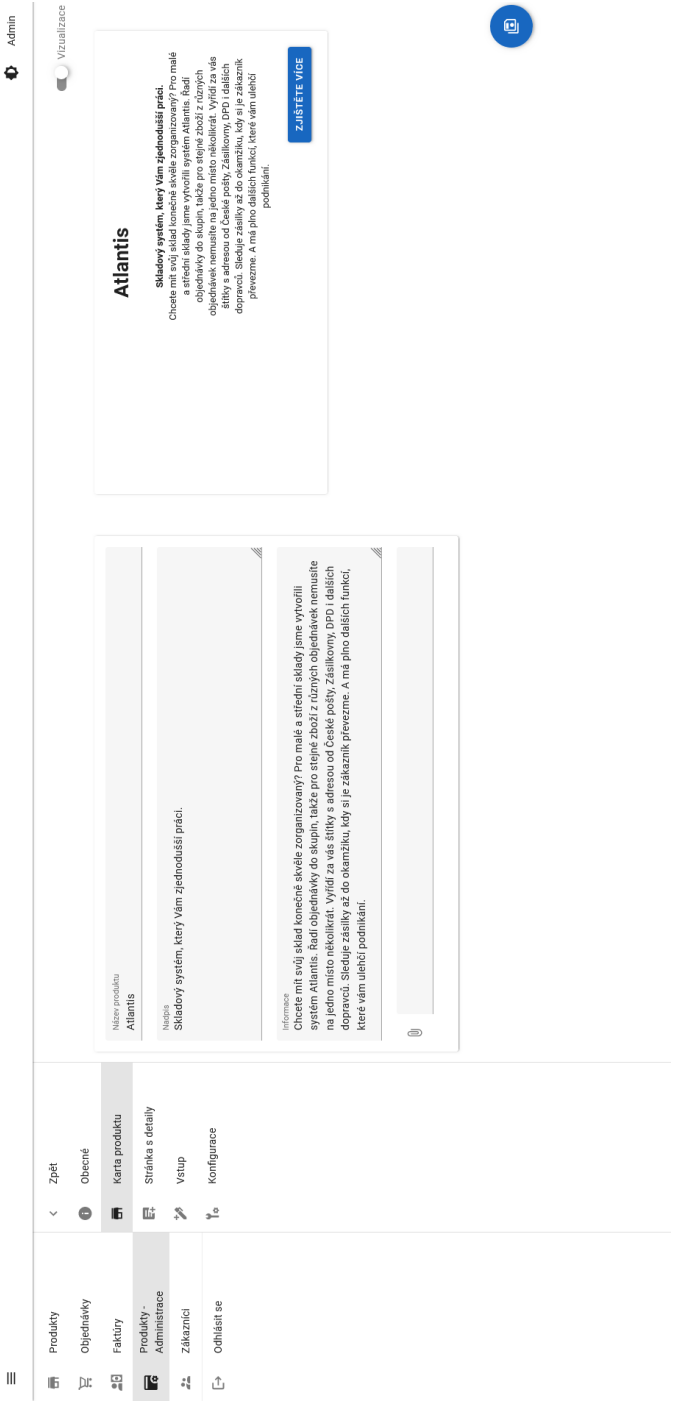
Obr. E.6 Prehľad zákazníkov



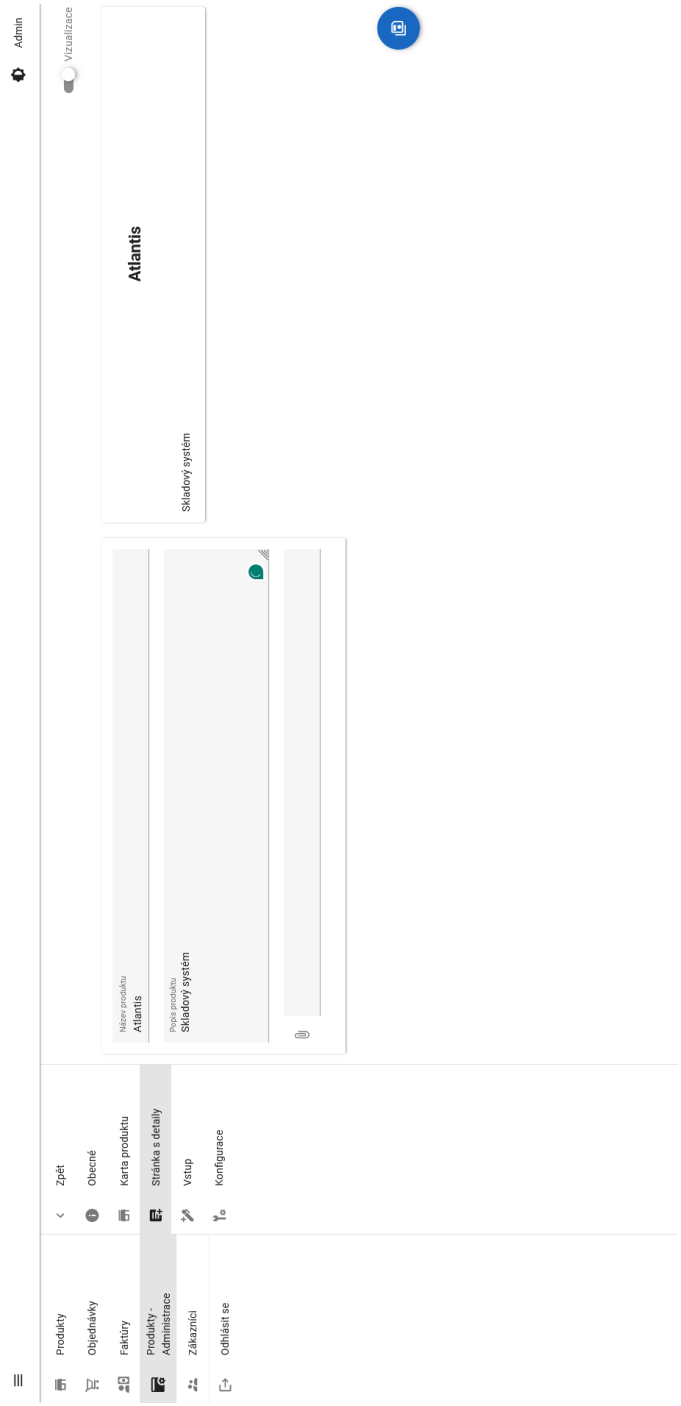
Obr. E.7 Administrácia produktov



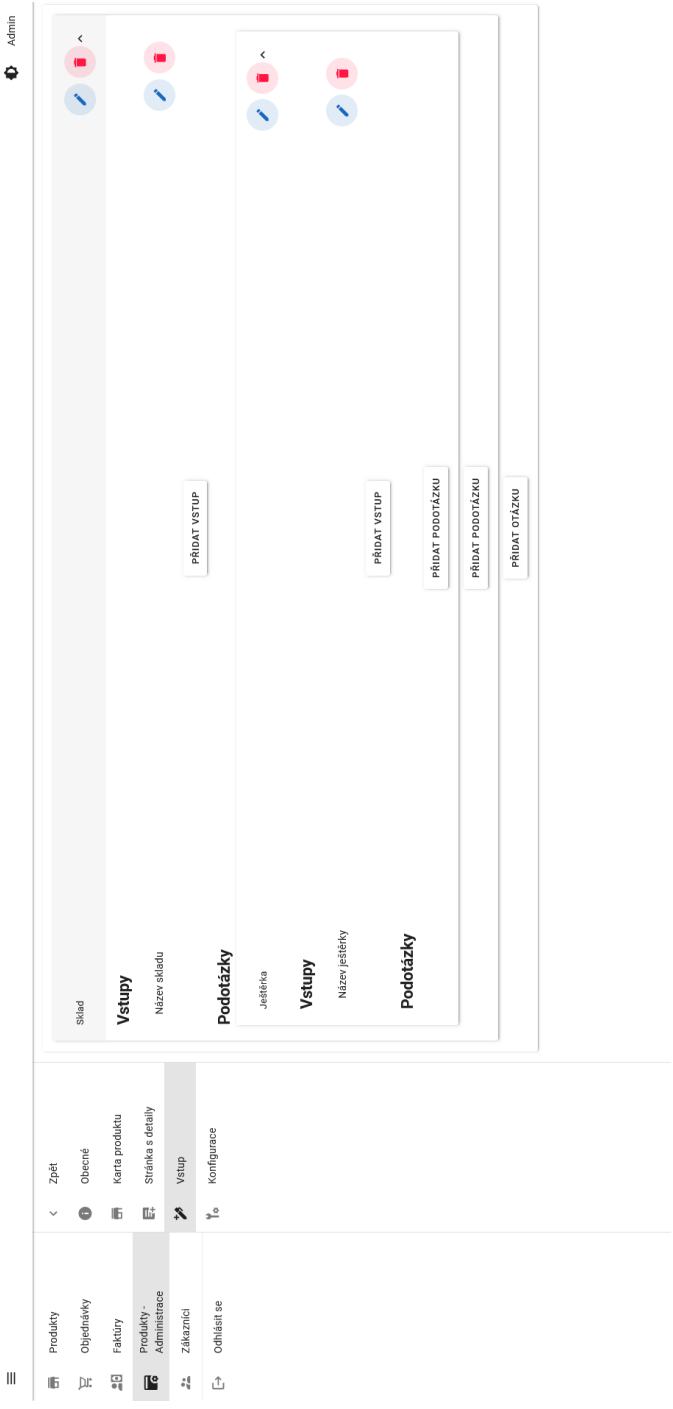
Obr. E.8 Administrácia produktu - všeobecné informácie o produkte



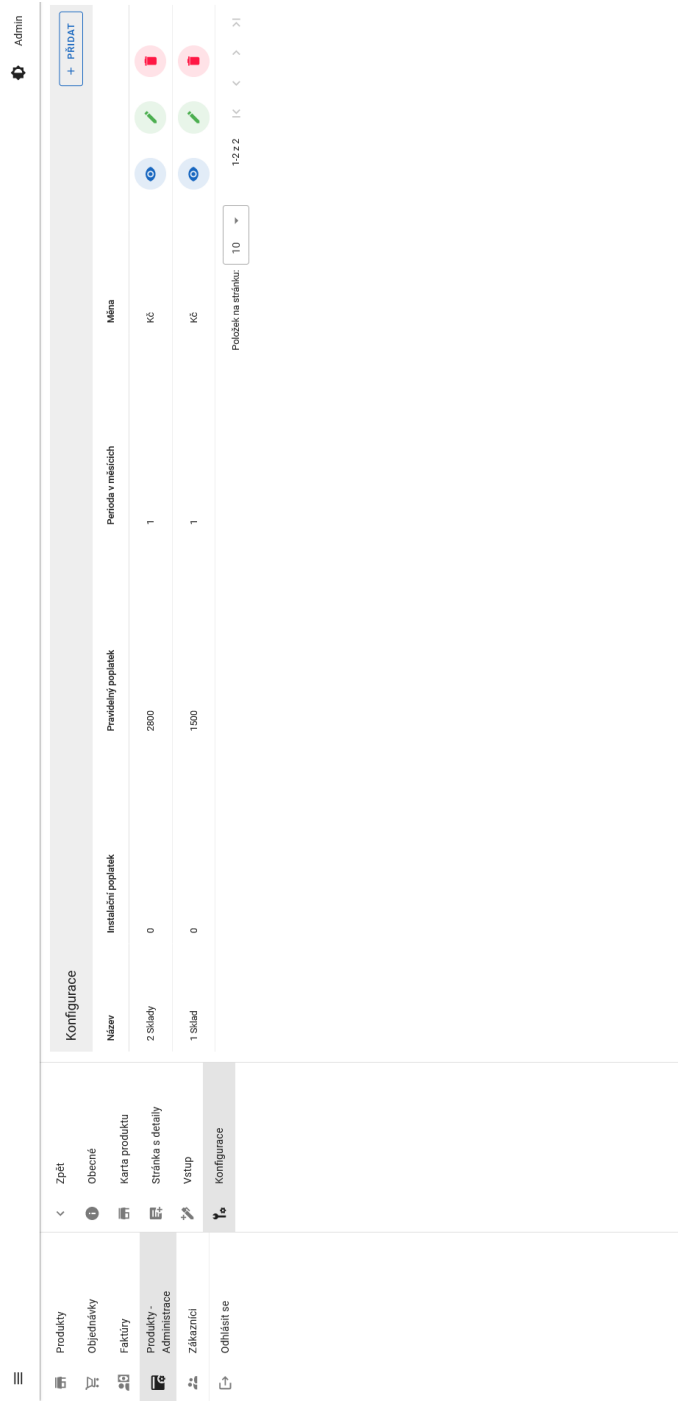
Obr. E.9 Administrácia produktu - karta produktu v prehľade produktov



Obr. E.10 Administrácia produktu - detailné informácie o produkte



Obr. E.11 Administrácia produktu - konfiguračný formulár produktu



Obr. E.12 Administrácia produktu - cenové plány produktu

Bibliografia

1. STAŠ, Adam. *Webová aplikace Deployment Manager*. 2023. Bak. pr. České vysoké učení technické.
2. JEŽKOVÁ, Alena. *Webová aplikace - Builder*. 2023. Bak. pr. České vysoké učení technické.
3. HOLÝ, Viktor. *Cloud native vývoj a jeho aplikování na projekt License Manager*. 2023. Dipl. pr. České vysoké učení technické.
4. HOLÝ, Viktor. *License manager - webová aplikace*. 2021. Bak. pr. České vysoké učení technické.
5. *Software License Manager* [online]. [cit. 2024-04-11]. Dostupné z : `https://www.capterra.com.au/directory/10016/license-management/software`.
6. *Seller issued licenses in License Manager* [online]. [cit. 2023-04-09]. Dostupné z : `https://docs.aws.amazon.com/license-manager/latest/userguide/seller-issued-licenses.html`.
7. *Software License Management* [online]. [cit. 2023-04-11]. Dostupné z : `https://www.10duke.com/solutions/software-license-management/`.
8. *What is a software license?* [online]. [cit. 2024-04-10]. Dostupné z : `https://www.techtarget.com/searchcio/definition/software-license`.
9. *What is SaaS?* [online]. [cit. 2024-03-31]. Dostupné z : `https://www.salesforce.com/in/saas/`.
10. *Software Licensing Models* [online]. [cit. 2024-04-15]. Dostupné z : `https://blog.tangent.com/software-licensing-models/`.
11. *Requirement gathering* [online]. [cit. 2024-04-11]. Dostupné z : `https://bootcamp.uxdesign.cc/requirement-gathering-793df91631a2`.
12. *What is FURPS+?* [online]. [cit. 2024-04-16]. Dostupné z : `https://businessanalysttraininghyderabad.wordpress.com/2014/08/05/what-is-furps/`.
13. *MoSCoW Prioritization* [online]. [cit. 2023-04-29]. Dostupné z : `https://www.productplan.com/glossary/moscow-prioritization/`.
14. S.BALAJI; MURUGAIYAN, Dr.M.Sundararajan. WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC. 2012, roč. 02, s. 26–30.
15. *What is an API?* [online]. [cit. 2024-04-03]. Dostupné z : `https://www.mulesoft.com/resources/api/what-is-an-api`.
16. *REST vs. SOAP* [online]. [cit. 2024-04-03]. Dostupné z : `https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest`.

17. *REST* [online]. [cit. 2024-04-11]. Dostupné z : https://www.redhat.com/en/topics/api/what-is-a-rest-api.
18. *What is gRPC?* [online]. [cit. 2024-04-11]. Dostupné z : https://aws.amazon.com/compare/the-difference-between-grpc-and-rest/.
19. *GraphQL | A query language for your API* [online]. [cit. 2024-05-08]. Dostupné z : https://graphql.org/.
20. *How does GraphQL work?* [online]. [cit. 2024-05-08]. Dostupné z : https://www.techtarget.com/searchapparchitecture/definition/GraphQL.
21. *What is a webhook?* [online]. [cit. 2024-04-11]. Dostupné z : https://mailchimp.com/resources/webhook-vs-api/.
22. *What is web application architecture?* [online]. [cit. 2024-04-16]. Dostupné z : https://mdevelopers.com/blog/what-is-web-application-architecture-.
23. *Front End Development* [online]. [cit. 2024-04-16]. Dostupné z : https://www.geeksforgeeks.org/frontend-vs-backend/.
24. *What is the backend?* [online]. [cit. 2024-04-16]. Dostupné z : https://airfocus.com/glossary/what-is-a-front-end/.
25. *Decoupled architecture: how to modernise your frontend* [online]. [cit. 2024-04-11]. Dostupné z : https://inviqa.com/blog/decoupled-architecture-how-modernise-your-frontend.
26. *What Are Single Page Applications? Examples, Frameworks, and More* [online]. [cit. 2024-03-14]. Dostupné z : https://geekflare.com/single-page-applications/.
27. *Multi-Page Application* [online]. [cit. 2024-05-08]. Dostupné z : https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58.
28. *What is a multi page application* [online]. [cit. 2024-05-08]. Dostupné z : https://www.ohmycrawl.com/faq/what-is-a-multi-page-application/.
29. *Why Backend For Frontend Pattern ?* [online]. [cit. 2024-04-16]. Dostupné z : https://medium.com/tech-lead-hub/why-backend-for-frontend-pattern-a67fcf7dfb8e.
30. *Material Design* [online]. [cit. 2024-03-05]. Dostupné z : https://m3.material.io/get-started.
31. *What Is Material Design and How Should It Be Used?* [online]. [cit. 2024-03-05]. Dostupné z : https://elementor.com/blog/what-is-material-design/.
32. *Why Use Material Design? Weighing the Pros and Cons* [online]. [cit. 2024-03-05]. Dostupné z : https://www.toptal.com/designers/ui/why-use-material-design.
33. *What is flat design?* [online]. [cit. 2024-04-24]. Dostupné z : https://uxplanet.org/flat-design-vs-material-design-whats-your-flavor-43a27c295f62.
34. *REQUIREMENTS & USE CASES* [online]. [cit. 2024-03-30]. Dostupné z : https://www.stratechi.com/requirements-use-cases/.
35. *Jednodušší a chytřejší podnikání pro vás* [online]. [cit. 2024-05-08]. Dostupné z : https://www.fakturoid.cz/.
36. *Figma: The Collaborative Interface Design Tool* [online]. [cit. 2024-05-08]. Dostupné z : https://www.figma.com/.
37. *Frictionless time tracking software for teams* [online]. [cit. 2024-04-15]. Dostupné z : https://toggl.com/.
38. *Timer2Ticket* [online]. [cit. 2024-04-15]. Dostupné z : https://app.timer2ticket.com/.
39. *Git –fast-version-control* [online]. [cit. 2023-04-29]. Dostupné z : https://git-scm.com/.

40. *Upgrading To 9.0 From 8.x* [online]. [cit. 2024-04-09]. Dostupné z : https://laravel.com/docs/9.x/upgrade.
41. *Upgrading to 10.0 from 9.x* [online]. [cit. 2024-04-09]. Dostupné z : https://laravel.com/docs/10.x/upgrade.
42. *Database: Migrations* [online]. [cit. 2023-05-01]. Dostupné z : https://laravel.com/docs/10.x/migrations.
43. *What Is API security?* [online]. [cit. 2024-04-21]. Dostupné z : https://brightsec.com/blog/api-security/.
44. *Laravel Sanctum* [online]. [cit. 2024-05-08]. Dostupné z : https://laravel.com/docs/10.x/sanctum.
45. *What is the OpenAPI Specification?* [online]. [cit. 2024-05-08]. Dostupné z : https://spec.openapis.org/oas/latest.html.
46. *Scramble – Laravel OpenAPI (Swagger) Documentation Generator* [online]. [cit. 2023-04-29]. Dostupné z : https://scramble.dedoc.co/.
47. *State of JavaScript 2022* [online]. [cit. 2024-05-02]. Dostupné z : https://2022.stateofjs.com/en-US/.
48. *Front-end Frameworks* [online]. [cit. 2024-05-02]. Dostupné z : https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/.
49. *Vue vs. React: How to Choose for 2024 - Learning curve* [online]. [cit. 2024-05-02]. Dostupné z : https://prismic.io/blog/vue-vs-react#learning-curve.
50. *Vue I18n - Internationalization plugin for Vue.js* [online]. [cit. 2024-05-02]. Dostupné z : https://vue-i18n.intlify.dev/.
51. *Getting started | Vuelidate* [online]. [cit. 2024-05-03]. Dostupné z : https://vuelidate-next.netlify.app/.
52. *Vue Router The official Router for Vue.js* [online]. [cit. 2024-04-19]. Dostupné z : https://router.vuejs.org/guide/.
53. *Unplugin Vue Router Typed, file-based routing for Vue 3* [online]. [cit. 2024-04-19]. Dostupné z : https://uvr.esm.is/.
54. *Vuetify - A Vue Component Framework* [online]. [cit. 2024-05-03]. Dostupné z : https://vuetifyjs.com/en/.
55. *openapi/openapi* [online]. [cit. 2024-05-02]. Dostupné z : https://github.com/openapi/openapi.
56. *VueUse - useFetch* [online]. [cit. 2024-05-02]. Dostupné z : https://vueuse.org/core/useFetch/.
57. KANER, Cem; BACH, James; PETTICHORD, Bret. *Lessons Learned in Software Testing.* New York: John Wiley & Sons, Inc, 2002. ISBN 0-471-08112-4.
58. *The different types of software testing* [online]. [cit. 2024-05-12]. Dostupné z : https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing.
59. *Selenium Overview* [online]. [cit. 2024-05-11]. Dostupné z : https://www.selenium.dev/documentation/overview/.
60. *Why Webdriver.IO?* [online]. [cit. 2024-05-11]. Dostupné z : https://webdriver.io/docs/why-webdriverio/.

Obsah priloženého média

- `readme.txt` ..... stručný opis obsahu média
- `src`
  - `frontend` ..... zdrojové kódy implementácie frontendu
  - `backend` ..... zdrojové kódy implementácie backendu
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `thesis.pdf` ..... text práce vo formáte PDF