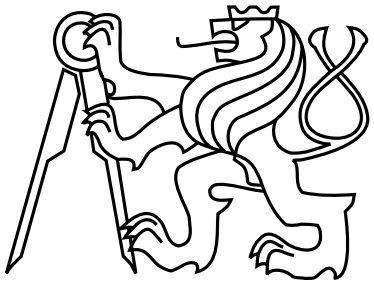
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídící techniky



Diplomová práce

Systém pro ověření původu zboží

Bc. Jan Pěček

Vedoucí práce: Ing. Lukáš Vojtěch, Ph.D.

Studijní program: Otevřená informatika (magisterský)

Obor: Počítačové inženýrství

17. prosince 2013

Prohlášení

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

V Praze dne 17. prosince 2013

Poděkování

V prvé řadě bych chtěl touto cestou poděkovat vedoucímu práce za podporu a trpělivost při psaní této práce. Dále chci poděkovat všem, kteří se mnou práci konzultovali a při jejím psaní mi poskytovali cenné rady. Slova díků patří také rodině, která mě po celé studium podporovala a byla vždy oporou, stejně jako mí přátelé a kamarádi. Poslední malé poděkování věnuji Ing. Petru Chlumskému, díky němuž jsem se k tématu této práce vůbec dostal.

Abstract

The process of goods origin verification may be sometimes difficult. This thesis is about design and implementation system for goods origin verification which contains strict security rules. It's divided by good practices of software engineering design to parts: requirements specification, analysis and design and the implementation. In the first part, functional and non-functional requirements are specified to build a real and working solution. There are security rules required too. In depend on the previous part with requirements the analysis and design is done, including use-cases definition (in a few views), used technologies and security rules. These rules describe processes and communication in the system and how to secure data stored with products. The last part is about system realization. It describes components structure, used technologies and principles. The goal of this work is to design and realize the secure system for goods origin verification.

Abstrakt

Původ zboží prodávaného v obchodech je stále nejasnější. Předmětem této práce je navrhnout a realizovat do funkčního řešení systém na ověření původu výrobků, který bude splňovat zadaná bezpečnostní kritéria. Práce je koncipována z hlediska návrhu softwaru na část, které stanovuje požadavky, analytickou kapitolu, jež na základě stanovených aspektů provádí analýzu a návrh řešení, a část samotné realizace systému. Jsou určeny funkční požadavky, které by řešení mělo splňovat, aby bylo reálně použitelné. I pro skutečné nasazení jsou definovány také nefunkční požadavky, především pak oblast bezpečnosti celého řešení. Část analýzy podrobně oba druhy požadavků rozebírá, důraz klade na přesnou definici případů užití z různých úhlů pohledu, použité technologie a také bezpečnostní kritéria, jak z pohledu zabezpečení zboží, tak z pohledu dalších procesů a komunikace. Nemálo důležitá je samotná realizační část popisující strukturu vlastních komponent systému a způsob jejich vytvoření. Cílem celé práce je fungující řešení systému pro ověření původu zboží.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Zahlceni levnými výrobky často pochybné kvality si lidé začínají uvědomovat, že kvalita něco stojí a někteří jsou ochotni si za ni připlatit. Zároveň se často nepozná, zda výrobek dané značky byl pod touto značkou skutečně vyroben (a výrobce si kvalitu výrobku pohlídal), anebo jej pod danou značkou vyrobil někdo jiný nelegálně. Setkáváme se tak s padělky zboží renomovaných světových značek, a to už nejen u obchodníků z Asie. Známá značka pak ve výsledku trpí a ztrácí svého věhlasu tím, že ji, neoprávněně, reprezentují výrobky, které ona na trh neuvedla.

Na první pohled spotřebitel nepozná, zda se jedná o výrobek původní značky nebo jeho napodobeninu popř. přímo padělek. Obal takového výrobku je shodný s originálem, stejně jako veškerá jeho označení. Protože i velké korporace nechávají své produkty vyrábět tam, kde je to levné (tedy především v Asii – Číně, Indonésii apod.), nepoznáme padělek ani podle země výroby. Samotný výrobek pak může zcela jistě dobře fungovat jako originál, ovšem nikdo již nedává záruku, že bude tak kvalitní a vydrží po stejnou dobu, jaká by byla životnost pravého kusu.

V některých oblastech, např. lihovin, tabákových výrobků či léků, se o původ a kvalitu zboží zajímají také státní úřady. Kromě výběru daní z těchto výrobků se tak snaží chránit občany před nepravým zbožím, které jim v konečném důsledku může způsobit nejen újmu na zdraví. Stát chrání také zájmy podniků, aby se do prodeje nedostávalo zboží, které z jejich výrobních linek nepochází, ale jejich značky obsahuje. Zákazník by tak měl vědět, co kupuje a kdo to vyrobil.

Právě nemožnost dostatečně zajistit původ daného zboží či jej ochránit před neoprávněnou manipulací vede k tvorbě různých ochranných mechanizmů, jak ze strany výrobců, tak ze strany státních orgánů. Takovými způsoby jsou např. označování zboží různými nálepkami, kolkování lihovin, vedení registru léčiv či razie na tržnicích, kde se padělané zboží vyskytuje nejčastěji. Aby nebyl spotřebitel klamán či výrobky zavedené značky neztrácely svou pověst, je třeba vytvořit způsob, jakým zcela jednoznačně určit, jaký je původ daného zboží, kdo jej vyrobil a případně kdo s ním manipuloval.

Každý zákazník nebo i státní kontrolní orgán by měl mít možnost pro daný produkt určit jeho původ, a to ideálně způsobem, který nebude možné snadno padělat, podvrhnout. Nabízí se tak prvky elektronické ochrany spolu s vytvořením takového procesu ověření, který bude důvěryhodný, nezaměnitelný a nepopiratelný, budou mu všichni věřit. Takové řešení musí

být dostatečně zabezpečené, aby nemohlo dojít k jeho kompromitaci, tedy aby bylo možné prokazatelně určit původ produktu.

Bude-li možné s dostatečnou přesností určit původ produktu, může být nepravost nelegálních kopií důsledněji postihována, zákazník si bude více vybírat, bude více ochráněn před nekvalitními či závadnými výrobky, kterými je současný trh zaplaven. Způsob ověření původu musí být univerzálně použitelný pro různé druhy výrobků a i při zachování spolehlivosti dostatečně rychlý, aby bylo možné ověřování provádět v masové míře. Díky téměř plošné existenci mobilního či jiného spojení s okolním světem je možné kromě vlastního označení produktů provést ověření původu pomocí internetové služby, která zajistí potřebnou záruku, že výrobek skutečně vyprodukoval uvedený výrobce. Tato služba i komunikace s ní musí splňovat určitá bezpečnostní kritéria, aby bylo možné s důvěrou prohlásit, že produkt je daného původu.

Systém pro ověření původu zboží, jenž je předmětem této práce, představuje návrh způsobu, jakým označovat zboží a zároveň provádět kontrolu jeho původu tak, aby bylo nepopiratelné, že se jedná o výrobek daného výrobce. Vše musí být důvěryhodné, protože jedině tak budou v řešení všichni věřit a používat jej. Proces ověření původu nesmí být zaměnitelný, jinak by nebylo možné původ produktu dostatečně prokázat. Systém má být přístupný komukoliv, aby kdokoliv mohl kontrolu původu provést a nepotřeboval k tomu žádné speciální postupy ani zařízení. Proto je navržena mobilní aplikace, kterou si do svého zařízení může nainstalovat každý. Serverové řešení pak této aplikaci poskytuje dostatečnou podporu, a to i pro případ, že původ výrobku zkoumá kontrolní orgán (státní úřad).

Práce se zabývá návrhem a zároveň řešením vlastního způsobu ověřování původu zboží, které bude dostatečně zabezpečené. Kromě samotných funkčních požadavků na dané řešení specifikuje také nefunkční požadavky, právě především s ohledem na bezpečnost, spolehlivost a stabilitu řešení. Analytická část pak rozebírá dané požadavky, představuje architekturu, zavádí do systému základní procesy a podrobně rozebírá veškeré vznesené požadavky z různých úhlů pohledu. Kromě analýzy technologií je velká část věnována právě rozboru bezpečnosti, jak vlastních dat uložených u produktů, tak i komunikace se serverovými součástmi.

Část realizační pak pokrývá problematiku vytvoření jednotlivých komponent a podle návrhu popisuje samotnou realizaci celého řešení. Výsledkem realizace je fungující celek, který lze s navrhovanými úpravami ve většinové míře reálně použít. Proto je rozebírán způsob implementace řešení, použité postupy a ukázky integrace použitých knihoven.

Cílem celé práce je vytvoření fungujícího systému pro ověření původu zboží, kdy je kromě analytického rozboru platného pro softwarové projekty provedeno samotné naprogramování jednotlivých komponent a jejich spojení tak, aby systém kompletně v rámci prototypového řešení fungoval.

Kapitola 2

Popis problému a specifikace cíle

V současné době existuje v podstatě jediný způsob označování zboží, a to pomocí tzv. čárových kódů. Zavádění elektronických bezkontaktních technologií je zatím v raném stádiu a jejich rozvoj můžeme očekávat až v následujících letech. Čárový kód zboží má mnohá omezení, např.:

- Produkt určitého druhu má shodný kód. Nelze tak odlišit dva produkty vyrobené např. v jiném časovém horizontu či jiné výrobní lince.
- Kód nemusí být za každých okolností čitelný. Stačí menší poškození a skener si se sledem čar již neporadí.
- Čárový kód obsahuje pouze identifikaci produktu. Všechny další informace musí být uloženy jinde (např. v nějakém informačním systému).
- Unikátnost kódu nemusí být zaručena napříč zeměmi či kontinenty.
- Znalost kódu může vést k vytvoření falešného obalu a tedy nepravé kopie originálního výrobku.

Výše zmíněné nevýhody vedou k vytvoření elektronického označení zboží, které některé z nedostatků odstraní při zachování minimálních nákladů. S výhodou se předpokládá využití RFID (Radio Frequency Identification) [57] technologie popř. jí podobných (NFC – Near Field Communication [36]). Každý z čipů, tagů, má jedinečný identifikátor, ID, který lze použít i k identifikaci daného výrobku. Do tagu lze pak s výhodou uložit další informace, např. informace o druhu zboží, spotřebě i výrobci.

Právě data vložená do tagu umožní také ověření pravosti daného výrobku. V součinnosti s výrobcem produktu můžeme jednoznačně stanovit, zda výrobek opustil linku dané firmy. V souvislosti s vývojem v posledních několika letech je tendence více hlídat pravost produktu i ze strany státních orgánů.

2.1 Legislativa

Česká legislativa upravuje vlastnosti výrobků v několika různých zákonech. Za zmínku stojí Zákon č. 102/2001 Sb., o obecné bezpečnosti výrobků a o změně některých zákonů (zá-

kon o obecné bezpečnosti výrobků) [43]. Ten upravuje již platné zákony a předpisy (např. Zákon č. 22/1997 Sb., o technických požadavcích na výrobky a o změně a doplnění některých zákonů [44] či Zákon č. 632/1992 Sb., o ochraně spotřebitele [15], ve znění pozdějších předpisů) a hovoří o průvodní dokumentaci výrobků jako i o označení původu zboží a kontrole Českou obchodní inspekcí. Výrobky vyžadující zvláštní kontrolu pak podléhají dalším nařízením a zákonům.

V roce 2005 vstoupil v platnost Zákon č. 676/2004 Sb., o povinném označování lihu a o změně zákona č. 586/1992 Sb., o daních z příjmů, ve znění pozdějších předpisů [47]. Část první tohoto zákona byla posléze zrušena a nahradil jej nově Zákon č. 307/2013 Sb., o povinném značení lihu [45]. Ten požaduje označovat lihoviny kontrolní páskou za přesně definovaných podmínek. Vydavatelem a regulačním orgánem je v tomto případě příslušný celní úřad. Tímto se státní správa snaží hlídat výrobu lihovin a bránit jejich padělání. Zákon vyžaduje registraci výrobce, vedení inventury kontrolních pásek a dokládání jejich použití. Státní správa tak má přehled o vyrobených lihovinách a spotřebiteli za určitých okolností zaručuje, že daný produkt pochází od prověřeného, registrovaného, výrobce.

Další státem kontrolovanou oblastí jsou léčiva. V zájmu ochrany zdraví obyvatel musí být každý produkt registrován, stejně jako jeho výrobce. Upravující právní předpis je v tomto směru především Zákon č. 378/2007 Sb., o léčivech a o změnách některých souvisejících zákonů (zákon o léčivech) [46], který definuje kromě náležitostí, jež daný přípravek musí obnášet, i způsob s jejich nakládání a kontroly Státním ústavem pro kontrolu léčiv.

2.2 Související události

Při zmínce o lihovinách nesmíme opomenout kauzu Metanol z roku 2012 [51], ve které došlo k pančování destilátů, falšování původu lihu a především nesprávnému zacházení s ním a míchání, čímž vznikl zdravotně závadný obsah s množstvím metanolu. S tím došlo i k vyhlášení tzv. prohibice v Česku a zavedení dalších restrikcí na označování obalů, změny a lepší evidenci kontrolních pásek i koncesování prodeje alkoholu. Každá lihovina pak musí mít dle Nařízení vlády č. 317/2002 [61] doklad o původu.

Kromě toho vznikl pod patronací Unie výrobců a dovozců lihovin web [58], pomocí něhož mohou zákazníci prověřit původ daného produktu a stáhnout jeho elektronicky podepsaný rodný list. Proti nekalým praktikám bojují i vlastní výrobci, např. Stock Plzeň – Božkov, s.r.o. webovou stránkou PijBezpečně.cz [55]. V neposlední řadě vznikla také mobilní aplikace pro ověření původu láhve [30].

2.3 Existující řešení

Na základě uvedené kauzy si poctiví výrobci hájí své produkty před padělám. Pomocí různých metod dávají zákazníkovi možnost si ověřit, že daný produkt pochází od nich, ověřit si jeho původ. Zmíněná legislativa to u některých typů výrobků přímo vyžaduje, jinde to může být všeobecně doporučováno. Padělky zboží světových značek se stále více objevují na trhu, především od výrobců z Asie. Zákazník je tak klamán a zavedená značka ztrácí nejen své renomé nad většinou nekvalitními výrobky, ale pochopitelně i zisk z neprodaných vlastních kusů.

Pod záštitou organizace GS1 [28] vzniklo mnoho různých standardů pro označování zboží, jeho distribuci i ukládání dat o produktech. Specifikace existuje i pro nejpoužívanější čárový kód s využitím kódu produktu EAN (European Article Number). Standardy jsou veřejně k dispozici včetně implementačních poznámek. Vlastní implementace nalezneme v různých zemích světa, případové studie jsou uvedeny na webu organizace. Nejzajímavější oblastí pro tuto práci, kterou se GS1 zabývá, je vývoj pod značkou EPCglobal [27], která vytváří standardy Eletronic Product Code (EPC). S využitím RFID se budují nejrůznější inteligentní dodavatelské sítě, inventarizační mechanizmy či vlastní řešení obchodů.

S GS1 spolupracuje Auto-ID Labs [8] založená na MIT (Massachusetts Institute of Technology) s podporou dalších světových univerzit. Sdruženo je zde několik výzkumných pracovišť zabývajících se nejen elektronickým označováním zboží. V rámci laboratoře bylo realizováno mnoho různých projektů, které méně či více souvisejí s tématem této práce. Nalezneme zde mimo jiné open-source implementaci standardu EPCIS, informačního systému pro trasování pohybu zboží distribučním řetězcem. Za povšimnutí jistě stojí technický manuál [39] k implementaci standardu Object Name Service (ONS) [29].

Služba ONS staví na principu fungování jedné ze základních služeb internetu – DNS (Domain Name System). Pomocí unikátních kódů produktu (získaného z EPC) sestavuje doménové jméno, které pak již za pomoci DNS převádí na volání služby daného subjektu. Informace o produktu jsou tak uloženy v cílové databázi např. výrobce, k níž je klient pomocí ONS nasměrován. Služba tak funguje jako jakýsi směrovač, který za pomoci identifikátoru firmy (umístěného na začátku kódu) určí cíl požadavku. Vlastní komunikace pak probíhá již přímo se službou subjektu. Ačkoliv může být tento informační tok zabezpečen, nikdo již nezaručuje, že došlo k nasměrování na správného výrobce popř. zda jsou zaslaná data platná.

2.4 Požadavky na funkčnost řešení

Cílem této práce je navrhnout a vytvořit systém splňující zadání. Požadované funkce velmi odpovídají standardu ONS, který však nezajišťuje požadovanou bezpečnost na systém a neobsahuje mechanizmy k zajištění integrity, nezaměnitelnosti a nepopiratelnosti dat. Implementace standardu je poměrně triviální úlohou, je popsána v uvedeném technickém manuálu a navíc v případě komerční realizace musí být certifikována pod patronací GS1.

Z hlediska bezpečnosti je řešení ONS více náchylné na možný útok podvržením dat nebo odposlechnutím. Není zde definována důvěryhodná autorita, která by data poskytovala tak, aby bylo nepopiratelné, že pocházejí z jasně určeného zdroje. Práce si tak klade za cíl navrhnout celý systém, který kromě funkcionálních požadavků splní i zadaná bezpečnostní kritéria.

Hlavní součásti

Celý projekt bude rozdělen do třech částí: serverové aplikace, klientské aplikace a ukázkové implementace informačního systému (IS) výrobce. Poslední část slouží pouze jako modelový příklad, aby celé řešení mohlo fungovat. V reálném provozu bude nahrazeno úpravou již existujících podnikových systémů typu SAP, Helios a další. Systém výrobce slouží jako zdroj informací o produktech, tyto údaje nebudou nikde centralizovaně ukládány.

A Klientská aplikace

Uživatel bude komunikovat se systémem pomocí klientské aplikace. Bude se jednat o jednoduchého klienta (thin-client), který se bude připojovat na vzdálenou službou. V této práci bude jako klient realizována mobilní aplikace pro operační systém Android. Klientská aplikace může existovat také pro jinou mobilní platformu, popř. se může jednat o nativní aplikaci určitého zařízení. Pro tento případ stačí, aby zařízení ovládalo síťovou komunikaci na úrovni TCP/IP a dokázalo zabezpečeně volat vzdálené služby dle definovaného protokolu.

Klientská aplikace bude připravena tak, aby mohla fungovat ve dvou variantách: pro běžného uživatele a pro privilegovaného uživatele. Privilegovaný uživatel bude mít možnost se do aplikace přihlásit a přistupovat tak ke chráněným datům za účelem vedení evidence či kontroly.

A.1 Přihlašování

Pro běžného uživatele bude použit účet tzv. hosta, o němž prakticky nemusí ani vědět. Ve druhé variantě aplikace musí počítat s možností přihlášení uživatele, které následně použije pro volání služby. Tato varianta aplikace pro privilegovaného uživatele bude obsahovat přihlašovací dialog, popř. přihlášení pomocí bezkontaktního autentizačního mechanizmu (např. formou načtení NFC tagu se zabezpečenými údaji příp. karty typu SmartCard – Mifare apod.). Zároveň je třeba realizovat i odhlášení, a to např. i po delší době nečinnosti.

A.2 Načtení NFC tagu

Aplikace bude umět číst identifikátor, ID, NFC tagu a data v něm uložená. V uložených datech se bude nacházet identifikace výrobce a základní data o produktu. Zároveň by mělo být možné rozhodnout, že tato data nikdo nemodifikoval. Aplikace by se měla integrovat do operačního systému a při načtení jakéhokoliv NFC tagu by měla být nabídnuta ke spuštění.

A.3 Zobrazení vyčtených informací

Aplikace ve strukturované podobě zobrazí načtené informace. Uživatel má dále možnost získat další volitelné informace online.

A.4 Načtení dat o produktu online

Pokud je dostupné datové připojení, aplikace se umí připojit na vzdálený server a získat doplňující informace o produktu.

A.5 Načtení dat o výrobci

Aplikace umožňuje zobrazit detailní informace o výrobci produktu popř. dalších subjektech, pokud jí server tyto informace poskytne.

A.6 Ověření dat

Aplikace obsahuje ověřovací způsob, který dokáže určit, že data v tagu nebyla nikým či ničím modifikována. Zprávu o úspěšném či neúspěšném ověření zobrazí uživateli.

B Server – webová služba

Server bude sloužit pro připojení klientské aplikace. Jednotlivým klientům poskytuje rozhraní, které klienti konzumují. Služby mají řízení přístupových práv, aby nemohlo dojít ke kompromitaci některé ze služeb neautorizovanou osobou/systémem.

B.1 Služba pro klientskou aplikaci

Rozhraní webové služby pro připojení klientské aplikace bude postaveno na technologii REST [17] a bude používat pro komunikaci formát JSON [14]. Rozhraní by mělo být navrženo univerzálně a co nejjednodušeji, aby bylo možné jej implementovat i v jiné klientské aplikaci.

B.2 Služba pro aktualizaci stavu produktu

Druhá webová služba bude postavena na SOAP [62] technologii, počítá se s rozšířením i na další rozhraní (XML-RPC, RMI). Pomocí této služby budou zasílány informace o předávce zboží výrobci, který tuto informaci u sebe eviduje. Pro snazší komunikaci musí rozhraní umět pracovat i s distribuční jednotkou (větším balením výrobků), nejen pouze se samotnými produkty.

B.3 Autentizace

Služba bude umožňovat autentizaci pomocí autentizačních mechanizmů HTTP protokolu [37]. Autentizační údaje obsahuje každý požadavek na server na základě použité autentizační metody. Aplikace samotná nebude uchovávat stav přihlášení klienta. Podle druhu rozhraní pak mohou být přijímaná data šifrována a podepsána.

B.4 Autorizace

Na základě zaslaných autentizačních údajů aplikace ověřuje přístup k datům (určitý uživatel může mít přístup zcela zakázán) a sestavuje požadavek pro odeslání výrobci.

B.5 Komunikace se serverem výrobce

Za pomoci údajů přijatých od klienta sestaví aplikace dotaz pro server výrobce. Ten je dohledán na základě identifikátoru, který údaje obsahují. K tomu budou nalezeny potřebné údaje pro sestavení požadavku (použité rozhraní, technologii, cílovou URL). Požadavek je odeslán na službu výrobce a následně je zpracována odpověď. Server musí počítat s časovým limitem pro příjem odpovědi a možnými chybovými stavy, které může volaná služba vrátit (chybné rozhraní, neexistující produkt).

B.6 Získání informací o firmě

Na základě autorizace bude možné načíst v databázi uložené informace o požadované firmě (výrobci). Tyto informace budou ve strukturované podobě vráceny klientské aplikaci.

B.7 Odpověď klientské aplikaci

Podle definovaného požadavku je sestavena odpověď pro klientskou aplikaci se získanými údaji (od výrobce či z databáze). Odpověď bude obsahovat pevné a volitelné položky, obsahuje strukturovaná data ve formátu JSON.

B.8 Podpora ověření dat v tagu

Služba bude poskytovat klientovi podporu a potřebné informace k tomu, aby mohl ověřit, že data v tagu nebyla změněna.

B.9 Rozhraní pro zaslání seznamu produktů

Služba obsahuje rozhraní, které přijímá identifikátor výrobce a seznam identifikátorů produktů uvedených na dodavatelském listu. Tyto údaje jsou použity pro odeslání výrobci, čímž dojde k aktualizaci stavu produktu u něj uloženém.

B.10 Rozhraní pro zaslání distribuční jednotky

Služba obsahuje rozhraní, které kromě identifikátoru výrobce přijímá identifikátor distribuční jednotky, jež obsahuje určité produkty. Rozhraní tak pouze předává daný identifikátor výrobci, který má danou distribuční jednotku přiřazenu k určitým vyrobeným produktům.

B.11 Odpověď klientovi

Podle definovaného požadavku je sestavena odpověď pro klienta, který použil rozhraní SOAP služby, v daném formátu, jímž se klient dotazoval. Odpověď může obsahovat informaci o chybovém stavu, který vznikl při komunikaci s výrobcem.

C Server – administrační rozhraní (Admin aplikace)

Druhou částí serverové strany je administrační rozhraní, které slouží pro správu přístupových údajů klientů a správu výrobců. Aplikace musí být zvlášť chráněna proti přístupu nepovolanou osobou, ideálně pomocí dvou-faktorové autentizace, aby nemohlo dojít k neoprávněnému přístupu.

C.1 Přihlášení

Do administračního rozhraní bude nutné se přihlásit. Proces přihlášení by měl být nejlépe zajištěn pomocí dvoufaktorové autentizace.

C.2 Autorizace

Přihlášený uživatel může mít přístup pouze do některé části aplikace na základě definovaných práv. Práva jsou uložena v databázi a řízení je možné na úrovni akcí, které aplikace obsahuje.

C.3 Správa uživatelů, klientů

Aplikace obsahuje správu uživatelů administrace a klientů. Možné bude mezi uživateli vyhledávat, přidávat nové uživatele, klienty, editovat jejich údaje, odebírat je a případně jim dočasně odebrat přístup. Součástí editace bude i nastavení práv pro přístup ke službám i administraci jako takové.

C.4 Správa firem, výrobců

Administrace slouží ke správě výrobců (údajů o nich), případně dalších subjektů. Uživatel může mezi údaji vyhledávat, přidávat a editovat záznamy, mazat je. Definuje se také rozhraní, které slouží pro komunikaci s výrobcem.

D Prototyp aplikace výrobce

Poslední součástí celé práce je prototyp informačního systému výrobce. U výrobce jsou ukládány informace o vyrobených produktech. Aplikace je navržena jako webová a slouží pro demonstraci celého řešení. Předpokládá se, že komunikace bude u výrobce integrována do již existujícího informačního systému.

Informační systém výrobce udržuje informace o vyprodukovaných výrobcích, jejich stav, spotřebu a cestu dodavatelským řetězcem až do obchodu (popř. až k likvidaci produktu). Jedná-li se o produkt, kde to vyžaduje legislativa, je uchovávána informace o každém článku při distribuci produktu. V opačném případě se systém omezuje pouze na uchování stavu produktu a informací o něm. Předpokládá se, že tuto část má již IS vyřešenu a není tedy součástí prototypu. Prototyp se omezuje pouze na stav produktu v distribučním řetězci.

D.1 Vytvoření nového produktu

Nahrání nového produktu je v reálném provozu automatizovaný proces. V tomto případě se jedná o založení produktu s daným ID tagu a doplňujícími informacemi (spotřebě, složení apod.) ve webovém rozhraní aplikace.

D.2 Rozhraní pro načtení informací o produktu

Komunikační službě je poskytnuto definované rozhraní, pomocí něhož může získat informace o produktu uložené v databázi. Rozhraní ověřuje existenci produktu a poskytuje pouze informace, které jsou požadovány.

D.3 Rozhraní pro aktualizaci stavu produktu

Rozhraní slouží pro aktualizaci stavu produktu v distribučním řetězci (informací o přijetí/odeslání např. distributorem). V případě, že se nejedná o výrobky, kde by toto bylo legislativou vyžadováno, služba nemusí být nakonec vůbec implementována. Aktualizace může probíhat na úrovni daného produktu (produktů) příp. identifikátoru expedované distribuční jednotky.

2.5 Požadavky na systém

Kromě funkčních požadavků na systém popsaných v předchozí části musí výsledné řešení splňovat i množství nefunkčních požadavků. Navrhovaný systém bude řešit především oblast bezpečnosti, neméně důležité jsou požadavky na výkon, dostupnost či spolehlivost. Vydefinována jsou také určitá pravidla pro použité technologie.

2.5.1 Architektura

Součástí realizovaného řešení by měl být i návrh architektury celého systému. Musí být stanoveno, jaké části se v systému objevují a jak spolu komunikují. Každá část by měla mít stanovenou oblast působnosti a způsob, jak si předává data s ostatními. V rámci toho je třeba se zaměřit na důvěryhodnost celého řešení.

Architektura systému by zároveň měla být navržena jako modulární, tj. počítá se s možným rozšiřováním systému. Do systému tak mohou přistoupit další subjekty, bude požadována další funkcionalita, další rozhraní.

2.5.2 Bezpečnost

Velmi důležitým faktorem celého řešení je oblast bezpečnosti. Musíme počítat nejen se zabezpečenou komunikací mezi jednotlivými rozhraními, ale i se zabezpečením samotných dat. Jednotlivým aplikacím jsou vždy poskytována jen určitá data, která jsou vyžádána.

Pro zasílání dat mezi službami se počítá s vytvořením zabezpečeného kanálu, samotný přenos může být podepisován a šifrován. Pro přístup k jakýmkoliv datům bude nutné přihlášení klienta, stejně jako pro přístup do webových rozhraní aplikací. K tomuto budeme používat již známých a prověřených řešení, která splňují nutná bezpečnostní kritéria.

Vlastní serverové řešení musí být navrženo proti neoprávněnému přístupu zvenčí. Přístupné bude pouze autorizovaným a pověřeným osobám, které budou mít přidělené jednotlivé oblasti působnosti. Vlastní databáze by pak měla být umístěna ideálně na šifrovaném disku, aby bez patřičných oprávnění nebylo možné se dostat k uloženým datům.

Systém má působit důvěryhodně, na tomto základě je celé řešení postaveno. Nebudou-li dodržena nastavená bezpečnostní pravidla, důvěra v systém může klesnout či zaniknout, což by ohrozilo provoz celého řešení. Data, která systém poskytuje, musí být nepopiratelná, tedy systém ručí za jejich původ. Pokud by klient nedostal data, která pocházejí z ověřeného zdroje, nemohl by být systém pro uživatele věrohodný. Každá poskytovaná data nesmí být možné zaměnit s jinými, tj. nesmí dojít k jejich podvržení. Systém se proti tomuto musí bránit a v případě narušení daného bezpečnostního mechanizmu uživatele varovat. Uživatel si také musí být jistý, že předkládaná data jsou pravá, autentická. Nikdo nesmí do dat zasáhnout, aby byla jejich pravost narušena. Pakliže se tak stane, systém upozorní uživatele, který může podniknout další kroky na základě svých pravomocí.

2.5.3 Provozní aspekty

Další nedílnou součástí nefunkčních požadavků jsou požadavky na provoz systému. Musíme počítat s určitým výkonem, škálovatelností, dostupností, spolehlivostí či auditovatelností. Protože se očekává poměrně vysoká zátěž systému, měli bychom tyto aspekty již od začátku zohlednit při návrhu řešení.

2.5.3.1 Výkon

Problematikou výkonu se zabývá každá aplikace. Od každého řešení se očekává maximální výkon, který však musí být podpořen volbou správných postupů či technologií. Očekáváme-li

vysokou zátěž systému, musíme to zohlednit při návrhu serverové infrastruktury či návrhu vlastních aplikací. Použijeme-li například technologie, které neběží v daném prostředí nativně (např. software v C/C++) ale ve vlastním běhovém prostředí (např. software v jazyce Java), můžeme očekávat, že výkon takového řešení nebude nejvyšší možný, neboť do běhu vstupuje ještě další mezivrstva. Naopak s výhodou můžeme takové řešení použít, pokud máme dostatek systémových prostředků (teoreticky nekonečné množství), ale požadujeme např. běh na univerzální platformě či lepší rozšiřitelnost nebo udržitelnost.

Podle [38] uživatel čeká maximálně 100 ms na reakci systému po provedení určité akce. Do jedné sekundy pak očekává, že dostane od systému nějakou odpověď a nebude na ni déle čekat. Aby uživatel dostal odpověď včas, musí navrhovaný systém 90 % požadavků zpracovat do dvojnásobku očekávaného reakčního času, tj. do 200 ms. Jedná se o dobu, kterou má aplikace na vlastní činnost, výsledná odpověď může být prodloužena při čekání na další komponenty. Aby uživatel na odpověď nečekal příliš dlouho, je stanovena maximální hranice pěti sekund (5 s) pro odeslání odpovědi. Pakliže některá ze součástí nereagovala dostatečně včas, systém o tom informuje klienta.

2.5.3.2 Škálovatelnost

Navržená architektura by měla být snadno škálovatelná pro další možný růst. Se vzrůstající zátěží nesmí být omezen provoz jednotlivých komponent, řešení musí být připravené na jejich rozšiřování, vertikální i horizontální (scale up, scale out). To znamená, že běžící aplikace musí být schopna fungovat i po horizontálním škálování (přidáním nového hardware), není nutné ji upravovat ani za tímto účelem rozšiřovat.

Systém bude postupně navyšovat svou kapacitu podle vzrůstajících nároků na dobu jeho odezvy se zachováním stálého výkonu. Řešení musí být postaveno tak, aby bylo škálování co nejjednodušší, v ideálním případě automatizované.

2.5.3.3 Dostupnost

Každá veřejně provozovaná služba s určitým počtem uživatelů i návštěvností musí mít zaručenou maximální dostupnost. Není možné, aby některá z komponent byla nedostupná déle než několik minut, a to pouze v případě, že není jiné možnosti, jak stoprocentní dostupnost zabezpečit.

Architektura systému musí být navržena tak, aby výpadkem kterékoliv komponenty nebyl ohrožen běh systému, tj. služba byla i nadále navenek dostupná, i když interně může být část systému mimo provoz. Zároveň je třeba počítat s pokrytím výpadku napájení či selhání celého systému (např. ztráta konektivity serverovny), systém musí být navržen tak, aby podobný fatální výpadek byl pokryt a uživatel nic nepoznal.

Celé řešení musí být postavené tak, aby splňovalo požadavky na vysokou dostupnost (high availability [50]) v každém časovém okamžiku. I provozovaná aplikace by měla být koncipována tak, aby výpadkem některé z důležitých součástí nedošlo k pádu aplikace nebo aby aplikace přestala uživateli odpovídat. Může však odpovídat chybovým stavem s vysvětlením problému a jeho případným nahlášením.

2.5.3.4 Spolehlivost

Nespolehlivý systém neodolá částečnému výpadku komponent či vyššímu počtu uživatelů. I při vysoké zátěži musí systém pokračovat ve zpracování žádostí od uživatele tak, jako v době menší zátěže. Integrita dat v tomto případě nesmí být narušena.

Jelikož uživatelé systému nejsou jenom fyzické osoby, ale mohou jimi být i další systémy, musí řešení být spolehlivé i proto, aby nenarušilo chod dalších aplikací. Propojenost systémů klade na spolehlivost ještě vyšší nároky, než kdyby cílovými uživateli byli pouze lidé.

2.5.3.5 Auditovatelnost

Aby byl systém spolehlivý a důvěryhodný, je třeba vést záznamy o jeho změnách, zvláště pak, jedná-li se o zásahy do systému samotného či vlastní databáze. Ke každé změně musí být uvedeno, kdo a kdy ji provedl, případě doplňující, vysvětlující popis. Tyto údaje se ukládají a uchovávají nejméně po dobu šesti měsíců zpětně. Všechny procesy v systému musí být pod dozorem, každé podezřelé chování musí být ihned prozkoumáno.

Serverové řešení by mělo být pod dohledem monitorovacího systému, který včas oznámí případné anomálie. Existovat musí i systém, který bude dohlížet na výkon a využití platformy, vést o tom záznamy, do nichž bude možné s historií řádově měsíců nahlížet.

2.5.3.6 Zálohování a obnovitelnost

Jednotlivé komponenty systému musí být pravidelně zálohovány. Zálohu je také třeba tvořit před rozsáhlejšími úpravami, kdy by mohlo dojít k situaci, že by bylo třeba zálohu použít pro provoz systému. Pravidelně, v řádu měsíce, se tvoří kompletní kopie komponent, spolu se záznamem změn je tedy možné se vždy vrátit k předchozí konfiguraci.

Databáze musí být zálohována minimálně jednou denně a vytvořená záloha uložena mimo celou architekturu na bezpečném místě a v zabezpečené podobě. Administrátoři popř. automatizovaný systém by měli zálohy často, jednou týdně, kontrolovat na úplnost dat. Nesmí se stát, že by vytvořenou zálohu nebylo možné v případě nutnosti použít.

2.5.4 Obecné požadavky na technologie

Z hlediska technologií je projekt rozdělen na dvě části: program pro OS Android a serverové, webové aplikace. Každá část vyžaduje použití odlišného přístupu při návrhu i při programování. Celý systém počítá s využitím open-source technologií tak, aby vývoji nebránily žádné licenční poplatky a bylo možné systém nadále rozšiřovat. Navíc to umožňuje komukoliv se znalostí dané technologie do projektu vstoupit a podílet se na jeho dalším rozvoji.

Pro serverové pozadí se počítá s využitím některé z volně šiřitelných linuxových distribucí používaných i v komerčním prostředí. Bude otázkou analýzy, zda to bude distribuce postavená na principech Debian či RedHat. Z těchto základních distribucí vzniklo během let mnoho různých derivátů, tento projekt by se měl držet osvědčeného a zavedeného řešení. Samotné serverové aplikace by neměly být závislé ani na jedné ze jmenovaných distribucí, stejně

tak by mělo být možné je vyvíjet a provozovat pod systémy Microsoft Windows. Vybrané řešení je potřeba pravidelně aktualizovat pro vyloučení chyb v systému.

Pro tvorbu aplikací není požadován žádný konkrétní programovací jazyk. Pro OS Android se zdá být nejvhodnější použít jazyk Java, pro serverové aplikace nechť je výběr jazyka podroben analýze. Preferován je všeobecně rozšířený a používaný programovací jazyk, založený na open-source principech a bez vazby na konkrétní platformu či komerční řešení.

Při vlastním vývoji je třeba se dodržet předepsaných standardů, dobrých zvyklostí a zásad, které vedou ke správně vytvořenému softwaru. Kladeny jsou tak nároky na přehledný a komentovaný kód, který lze snadno rozšiřovat a udržovat.

Serverové aplikace budou pro svůj běh používat databázový systém. Stejně jako u programovacího jazyka, je upřednostněno volně dostupné, osvědčené, bezpečné a všeobecně rozšířené řešení. Výběr by měl najít alternativu k databázovému řešení od společnosti Oracle, které je vyloučeno z důvodu licenční politiky (nároků na finanční prostředky). Daná platforma by měla být stavěna na vysokou zátěž, popř. je vyžadována taková funkcionalita, která pokryje množství dotazů směřujících na server. Velmi důležitá je stabilita systému a konzistence dat v něm uložených.

Kapitola 3

Analýza a návrh řešení

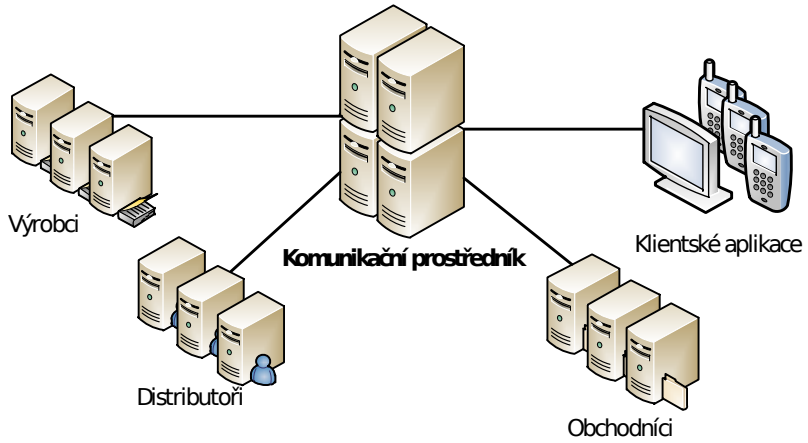
Podle požadavků specifikovaných v předchozí kapitole provedeme nyní analýzu možného řešení. Navrhneme tedy architekturu celého systému, základní procesy a na základě požadavků sestavíme případy užití (use-case) pro daný systém. V rámci analýzy budeme kromě funkčních požadavků zkoumat i nefunkční požadavky (bezpečnost, technologie). Součástí návrhu bude také seznam služeb, které budou jednotlivé části architektury vystavovat a konzumovat.

3.1 Architektura systému

Hlavním cílem této práce je sestavení řešení, které bude sloužit jako důvěryhodná autorita, jež poskytuje data dalším subjektům. Zároveň bude vytvořen klient v podobě mobilní aplikace a pro celkovou funkčnost i exemplární IS výrobce produktu. Součástí architektury mohou být IS jednotlivých článků distribučního řetězce, které spolu nějakým způsobem komunikují, jak je zobrazeno na obr. 3.1. Infrastrukturu tak tvoří výrobci, obchodníci a distributoři, z nichž každý má svůj informační systém. Tyto systémy mohou být již nějak propojeny, v navrhovaném případě sdílí data přes tzv. **komunikačního prostředníka**. Ten obstarává propojení jednotlivých komponent a poskytuje rozhraní pro připojení dalších IS. Navenek vystavuje služby. Klientské aplikace (mobil, terminál apod.) se připojují na komunikačního prostředníka, který pro ně získává data z daného IS výrobce.

Výrobci jsou v pojetí této práce informačními systémy výrobců obsahující uložené informace o produktech, jejich matriční záznamy. Realizovány jsou již existujícími podnikovými systémy nejrůznějšího druhu (např. SAP, Helios). K těmto systémům bude komunikační prostředník vznášet své dotazy.

Distributoři jsou jednotlivé články distribučního řetězce, resp. informační systémy daných firem. Každý produkt po opuštění výrobní linky putuje pomocí distributorů až do obchodu (mezitím může být kdekoliv uložen na skladě, u výrobce distributora i obchodu). Distributor může také přebírat nepoužitý produkt zpět z obchodu. Tento systém, opět zajištěný již existujícím řešením, pak může pomocí komunikačního prostředníka aktualizovat stav daného produktu u výrobce.



Obrázek 3.1: Infrastruktura celého projektu

Obchodníci představují cílovou destinaci produktu, odkud jej kupuje zákazník. Obchodem opět rozumíme spíše informační systém, který může pomocí komunikačního prostředníka odeslat výrobci informaci, že produkt byl uveden do obchodu, prodán (aktualizovat stav).

Klientské aplikace se připojují na komunikačního prostředníka a pokládají dotazy na ověření původu produktu. Touto aplikací může být mobilní klient, terminál či nativní zařízení, které umožňuje spolupracovat s komunikačním prostředníkem.

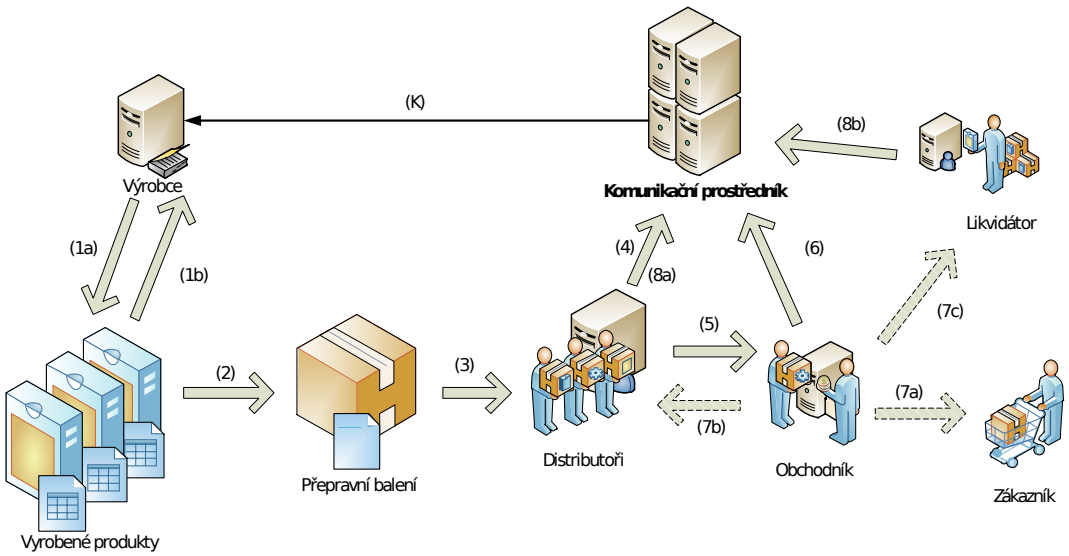
Pozn.: Distributory a obchodníky nazýváme v této práci též jako **subjekty**, pokud není řečeno jinak.

Právě komunikační prostředník se má stát onou důvěryhodnou autoritou, která získává data od výrobce a předává je dále. Představujeme si jej taky jako „entitu“, která stojí uprostřed veškerého dění a směřuje požadavky na správnou adresu. Klientům se tak poskytuje jednotné rozhraní pro přístup do více systémů zároveň. Zaručen je také jasný původ dat, protože k prostředníkovi se data dostávají speciální zabezpečenou cestou. Vlastní databáze prostředníka neobsahuje žádná data o produktech ani neshromažďuje data již vyčtená (může pouze vést záznamy o požadavcích). To dává výrobcům důvěru, že poskytnutá data nemohou být zneužita a nelze z nich vyčíst například obchodní údaje. K dispozici je databáze zúčastněných firem, kterou klient může číst. Uložena jsou také data potřebná k ověření původu výrobku. Kromě toho je získání podmíněno určitým oprávněním uživatele. Klientům tak nemusí být poskytována kompletní množina dat, o které prostředník výrobce požádal, popř. žádá jenom určitou část.

Paralelu důvěryhodného subjektu můžeme nalézt v podobě certifikační autority [16]. Ta ve světě internetové bezpečnosti hraje velmi důležitou roli při šifrování (zabezpečená spojení), ověřování autenticity, nepopiratelnosti, důvěryhodnosti a nezaměnitelnosti dat (digitální podpis). Certifikační autoritu uznávají všichni uživatelé i aplikace, neboť byla stanovena společným konsenzem (viz problematika tzv. Root certifikátů), popř. jí byla dána pravomoc vykonávat svou činnost tak, aby ji všichni důvěřovali (a může se jednat i o komerční společnost). Komunikační prostředník v celém systému hraje roli důvěryhodného elementu, kterého všechny zúčastněné strany uznávají i proto, že sjednocuje komunikační rozhraní a data od něj pocházející jsou vždy z nepopiratelného zdroje (tj. ví, že pocházejí od skutečného výrobce produktu).

3.2 Životní cyklus produktu

Jak již bylo nastíněno v části o architektuře systému (viz 3.1), každý produkt prochází určitým životním cyklem. Jeho matriční záznamy jsou stále drženy v systému výrobce, pomocí komunikačního prostředníka se provádí jejich aktualizace (resp. změna stavu produktu) a čtení pro klienty. Celý proces je znázorněn na obr. 3.2.



Obrázek 3.2: Životní cyklus produktu

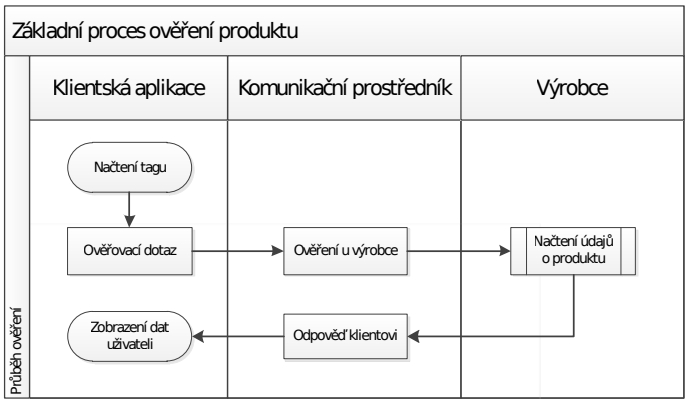
Po vyrobení je produkt elektronicky označen, přičemž součástí označení je vložení určitých informací o něm ze systému výrobce (1a). Tato data jsou zároveň zabezpečena tak, aby bylo později možné prohlásit, že nebyla změněna. Unikátní identifikátor příslušející ke každé produktové položce (jednotlivému výrobku) je zapsán do systému výrobce (1b), kde jsou k němu připojeny další možné informace (kromě těch zapsaných). Výrobky jsou zabaleny do přepravního balení (2) a předány distributorům (3). Součástí dodání (dodavatelského listu) je i seznam identifikátorů, které dané přepravní balení obsahuje. Distributoři si mezi sebou zboží předávají a pomocí komunikačního prostředníka, který je spojený s výrobcem kanálem (K), odešlou informace o tom, že mají zboží u sebe uložené, resp. s ním v daný okamžik operují (4). Distributor odesílá seznam identifikátorů, které dostal v dodacím listu. Distributorů může být na cestě více, všichni postupují stejným způsobem (tj. odesílají informace o průběhu cesty zboží).

Přes řetězec distributorů se zboží dostane až k obchodníkovi (5), který jej přijme a odesílá pomocí komunikačního prostřední informaci o přijetí zboží (6). Produkt může být následně zakoupen zákazníkem (7a), kde jeho životní cyklus končí. Stejně tak může být zboží vráceno zpět distributorovi (7b) např. po vypršení určité lhůty. Distributor opět provádí odeslání informace komunikačnímu prostředníkovi (8a) o převzetí zboží, ten informuje výrobce pomocí kanálu (K). Speciálním případem může být subjekt likvidátora, kterému obchod předává (7c) zboží určené k likvidaci (vynucené například právní úpravou). Likvidátor, stejně jako distributor, odesílá informaci o přijetí komunikačnímu prostředníkovi (8b). Výrobce je pak obeznámen se zlikvidováním daného výrobku.

Pomineme-li fakt, že se produkt může od distributorů dostat zpět k výrobci, životní cyklus produktu končí u zákazníka, distributora či likvidátora. Alternativně se může produkt k likvidátorovi dostat z obchodu přes distributora. Protože subjekt likvidátora je čistě specifický, nebudeme jej nadále v textu uvádět a budeme předpokládat, že jeho funkci nahradí některý z distributorů. Z pohledu systému komunikačního prostředníka se jedná o další zaregistrovaný subjekt, který odesílá identifikátor produktu, čímž v systému výrobce aktualizuje daný stav.

3.3 Ověření a aktualizace produktu

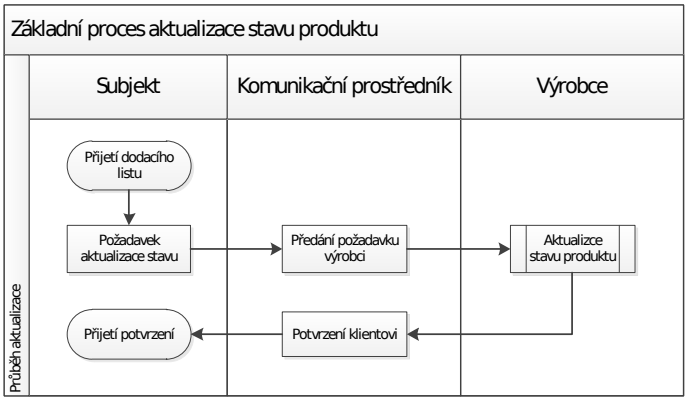
Na základě životního cyklu produktu a požadavků vytvořených v části 2.4 sestavíme základní procesy pro ověření a aktualizaci produktu. Oba procesy nezohledňují chybové ani neočekávané stavy, stejně jako nerozebírají zabezpečení celé komunikace. Tomu se budeme věnovat až v další analýze.



Obrázek 3.3: Ověření produktu u výrobce – základní proces

Ověření produktu je iniciováno, jak znázorňuje obr. 3.3, z klientské aplikace, která načítá tag produktu. Poté vytváří ověřovací dotaz, který je poslán komunikačnímu prostředníkovi. Ten dotaz přijme a sestaví zprávu pro ověření produktu u samotného výrobce. Výrobce tuto zprávu přijímá a načte další informace o produktu, čímž potvrdí jeho existenci ve svém systému. Nalezené informace odesílá zpět komunikačnímu prostředníkovi, který na jejich základě sestaví odpověď klientské aplikaci. Ta data přijme a zobrazí uživateli. Pokud by produkt nebyl v IS výrobce nalezen, šla by zpět zpráva o neúspěšném pokusu o ověření.

Druhým základním procesem objevujícím se v systému je proces aktualizace produktu u výrobce, jak je uvedeno na obr. 3.4. Inicializace komunikace probíhá u klienta, jímž je distributor či obchod (obecně subjekt), který z dodacího listu produktu vyčítá jeho informace (unikátní identifikátor), sestaví požadavek pro aktualizaci stavu a odešle jej komunikačnímu prostředníkovi. Ten požadavek přijme a sestaví zprávu pro výrobce. Výrobce tuto zprávu přijímá a ve vlastním systému provede aktualizaci stavu produktu na základě přijatých dat. Zpět zasílá pouze potvrzovací odpověď, kterou komunikační prostředník předává dále subjektu. Ten si může u daného produktu poznamenat, že již odeslal zprávu o jeho aktualizaci výrobci, který jako jediný tyto údaje udržuje. Pokud by výrobce u sebe produkt nenašel, odpovídal by chybovou zprávou.



Obrázek 3.4: Aktualizace stavu produktu – základní proces

3.4 Případy užití – analýza funkčních požadavků

Nadefinovali jsme předpokládanou architekturu jednotlivých částí systému, představili životní cyklus produktu a základní procesy v systému probíhající. Nyní se budeme konkrétně zabývat analýzou vlastních funkčních požadavků, tedy požadavků na vlastní aplikaci. Nefunkčním požadavkům jsou pak věnovány další kapitoly.

Součástí analýzy požadavků definovaných v části 2.4 je definice případů užití (use-cases) pro vytvářený systém. Rozeberme si nyní veškerou požadovanou funkcionalitu podrobně. Protože se projekt skládá z několika různých částí, jsou rozebrány všechny případy, kdy účastníkem (actor) se může stát nějaký systém a naopak. Kromě grafického vyobrazení patří ke všem případům i scénáře jednotlivých akcí.

3.4.1 Definice pojmů a účastníků

Každý z následujících diagramů obsahuje hlavní a vedlejší účastníky (primary and secondary actors). Hlavní účastník vyvolává událost v systému, vedlejší účastník pak slouží k její realizaci. Hlavní i vedlejší účastník v některém z případů užití může být systémem, v němž jiní účastníci provádějí své akce. Přehled všech účastníků a jejich generalizace je zobrazen na obr. 3.5.

Uživatelem systému se rozumí zákazník, který si ověřuje informace o zboží. Uživatelem může být také *Kontrolor* (privilegovaný zákazník). Oba používají pro přístup k systému *Klientskou aplikaci*.

Administrátor používá *Admin aplikaci* pro správu údajů uložených v *Databázi*. Jeho přístup je řízen speciálními pravidly definovanými požadavky. Přistupuje přímo k serverové aplikaci (nikoliv ke klientovi) a spravuje uložená data. Obecně jej můžeme také považovat za *Uživatele* (protože se jedná o fyzickou osobu).

Databáze je rozdělená schématicky na dvě hlavní části: *Databáze výrobců* a *Auth databáze*. Jako celek poskytuje ukládání a výběr dat pro aplikaci. První zmíněná část obsahuje informace o výrobcích (identifikaci, sídlo, adresu, kontakty) včetně definice komunikačního

rozhraní (URL, typ služby, parametry a názvy volaných metod), druhá část ukládá data pro autentizaci a autorizaci *Klientů* a *Administrátorů*.

Klientskou aplikací se rozumí mobilní klient komunikující s *Komunikačním prostředníkem* pomocí definovaného rozhraní. Aplikace slouží jako hlavní interface pro *Uživatele* systému.

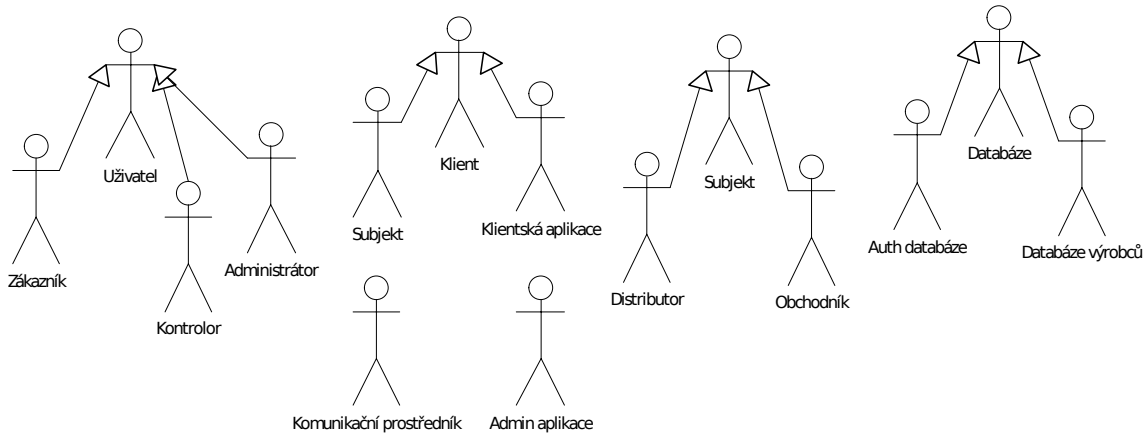
Komunikační prostředník je centrálním propojovacím bodem, který poskytuje rozhraní svým *Klientům* a konzumuje služby *Výrobců*. Využívá *Databázi* pro ukládání dat. Obsahuje front-end, na který se připojují služby, a back-end, který slouží pro připojení k *Databázi*.

Admin aplikace slouží pro správu dat v *Databázi*. Připojuje se přímo na back-end systému. Přístup do ní má pouze *Administrátor*.

Klientem se rozumí aplikace, která se připojuje na *Komunikačního prostředníka*. Kromě *Klientské aplikace* to může být i informační systém *Subjektu*. Klient konzumuje služby, které prostředník poskytuje.

Subjekt komunikuje s *Komunikačním prostředníkem*, jsou to *Distributoři* a *Obchodníci*, jak již bylo uvedeno. Každý subjekt má vlastní informační systém (IS), který konzumuje poskytované rozhraní *Komunikačního prostředníka*.

Výrobce komunikuje výhradně s *Komunikačním prostředníkem*, kterému poskytuje rozhraní pro zjištění/ověření informací o produktu a aktualizaci stavu produktu.



Obrázek 3.5: Účastníci případů užití

3.4.2 Uživatel – Klientská aplikace

Požadavek na funkcionalitu z části A Klientská aplikace zobrazuje na obr. 3.6 případ užití, kde je hlavním účastníkem *Uživatel*, který používá *Klientskou aplikaci* jako systém, jenž komunikuje s *Komunikačním prostředníkem* jako vedlejším účastníkem.

Obrázek 3.6: UC 1: Uživatel – Klientská aplikace

Případ užití	**Přihlásit**
Požadavek	A.1 Přihlašování
Cíl	Přihlásit uživatele do klientské aplikace a získat tak privilegovaný přístup, uložit přihlašovací údaje do paměti.
Vstupní podmínky	Uživatel není přihlášen.
Úspěšný výsledek	Uživatel je přihlášen.
Neúspěšný výsledek	Přihlášení uživatele se nezdařilo.
Hlavní účastníci	Uživatel
Vedlejší účastníci	*nejsou*
Akce	Uživatel se chce přihlásit do aplikace.

Hlavní scénář	1. Uživatel stiskne přihlašovací tlačítko v aplikaci. 2. Uživatel vyplní přihlašovací údaje: jméno a heslo. 3. *include::Uložit přihlašovací údaje*: Aplikace uloží přihlašovací údaje do paměti. 4. Uživatel je přihlášen. 5. Namísto přihlašovacího tlačítka je tlačítko pro odhlášení a identifikace přihlášeného uživatele (jeho jméno).
Možná změna	2. 1. Uživatel místo vyplnění přihlašovacích údajů přiloží svoji osobní kartu (NFC tag s šifrovanými daty, karta typu Mifare apod.). 2. 2. Zobrazí se pole pro zadání PINu pro přístup k údajům na kartě. 2. 3. Z karty jsou načteny přihlašovací údaje uživatele použité pro přihlášení.
Výjimky	2. a. Uživatel zadal prázdné údaje – aplikace neakceptuje prázdné přihlášení. 2. b. Uživatel opustil přihlašovací obrazovku. 2. c. Nebyl zadán správný PIN při ověření přihlášení pomocí karty. Aplikace umožní jeho opakované zadání.

Případ užití	**Odhlásit**
Požadavek	A.1 Přihlašování
Cíl	Odhlásit uživatele z klientské aplikace, smazat přihlašovací údaje z paměti.
Vstupní podmínky	Uživatel je přihlášen.
Úspěšný výsledek	Uživatel není přihlášen.
Neúspěšný výsledek	*není*
Hlavní účastníci	Uživatel
Vedlejší účastníci	*nejsou*
Akce	Uživatel se chce odhlásit z aplikace.
Hlavní scénář	1. Uživatel stiskne odhlašovací tlačítko v aplikaci. 2. *include::Smazat přihlašovací údaje*: Aplikace smaže přihlašovací údaje z paměti. 3. Uživatel je odhlášen. 4. Namísto odhlašovacího tlačítka je tlačítko pro přihlášení.
Možná změna	1. 1. Uživatel je odhlášen po delší době nečinnosti aplikace, příp. jejím ukončení.
Výjimky	*nejsou*

Případ užití	**Číst tagy**
Požadavek	A.2 Načtení NFC tagu
Cíl	Načíst data uložená v NFC tagu.
Vstupní podmínky	Zařízení obsahuje podporu pro čtení NFC tagů, aplikace je spuštěna.
Úspěšný výsledek	Data o produktu jsou načtena v paměti.
Neúspěšný výsledek	Data v tagu nemají správný formát, neexistují – chybová hláška na obrazovce.
Hlavní účastníci	Uživatel
Vedlejší účastníci	*nejsou*
Akce	Uživatel chce načíst tag umístěný na produktu.
Hlavní scénář	1. Uživatel přiloží zařízení do blízkosti tagu. 2. Aplikace přečte informace uložené v tagu. 3. Přečtené informace jsou uloženy do paměti.
Možná změna	1. 1. Aplikace nemusí být spuštěna. Její spuštění může vyvolat až načtení tagu a vybrání obslužné aplikace z nabídky.

Výjimky	2. a. Tag neobsahuje žádné informace – aplikace informuje chybovou hláškou uživatele. 2. b. Přečtená data nejsou uložena v definovaném formátu – aplikace informuje chybovou hláškou uživatele.

Případ užití	**Ověřit platnost dat v tagu**
Požadavek	A.6 Ověření dat
Cíl	Ověřit, že data načtená z tagu nebyla nikým, ničím změněna a pochází přímo do výrobce.
Vstupní podmínky	Načtená data z tagu, dostupné informace ze serveru pro ověření platnosti dat.
Úspěšný výsledek	Data v tagu nebyla nijak modifikována, zobrazit je na obrazovce.
Neúspěšný výsledek	Nelze získat data nutná pro ověření, ověření selhalo.
Hlavní účastníci	Uživatel
Vedlejší účastníci	Komunikační prostředník
Akce	Uživatel chce ověřit platnost zapsaných dat, tj. že nebyla nikým/ničím zmodifikována.
Hlavní scénář	1. Z lokálně uložených dat jsou vyčteny potřebné informace pro ověření. 2. *include::Ověřit platnost dat*: Platnost načtených dat je ověřena vůči jejich „podpisu“, který byl taktéž načten spolu s daty. 3. Úspěšné ověření je vyobrazeno na obrazovce. Pomocí dané volby je možné zobrazit detaily o ověření.
Možná změna	*extends::Načíst potřebná data ze serveru*: 1. 1. Potřebná data pro ověření nemusí být uložena lokálně nebo vypršela jejich platnost. 1. 2. Aplikace si tak musí vyžádat online spojení a dotázat na komunikačního prostředníka pro stažení dat, resp. ověření jejich platnosti. 1. 3. Při ověření aplikace zasílá otisk dat a identifikátor výrobce. 1. 4. Po stažení jsou data uložena do lokálního úložiště, resp. aktualizováno poslední datum ověření platnosti.
Výjimky	1. a. Nebylo možné získat příslušná data pro ověření (ani lokálně ani vzdáleně) – zobrazena je chybová hláška. 1. b. Platnost dat vypršela a nebyla dodána nová – zobrazena je chybová hláška. 2. a. Platnost dat se nepodařilo ověřit, jsou pravděpodobně změněna – na obrazovce je zobrazeno varování o možné nepravosti produktu.

Případ užití	**Zobrazit data o produktu**
Požadavek	A.3 Zobrazení vyčtených informací
Cíl	Zobrazit data načtená z tagu nebo online.
Vstupní podmínky	V paměti jsou uložená data načtená z tagu, mají správný formát, popř. data načtená online.
Úspěšný výsledek	Informace o produktu jsou zobrazena uživateli.
Neúspěšný výsledek	V paměti nejsou načtena žádná data (neměla správný formát).
Hlavní účastníci	Uživatel
Vedlejší účastníci	*nejsou*
Akce	Po načtení dat z tagu nebo online se uživateli zobrazují informace o produktu.
Hlavní scénář	1. Data z paměti jsou zobrazena uživateli.
Možná změna	1. 1. Data jsou načítána online (viz případ Uživatel – Klientská aplikace)

Výjimky	1. a. V paměti nebyla načtena žádná data, načtení bylo neúspěšné – chybová hláška na obrazovce.

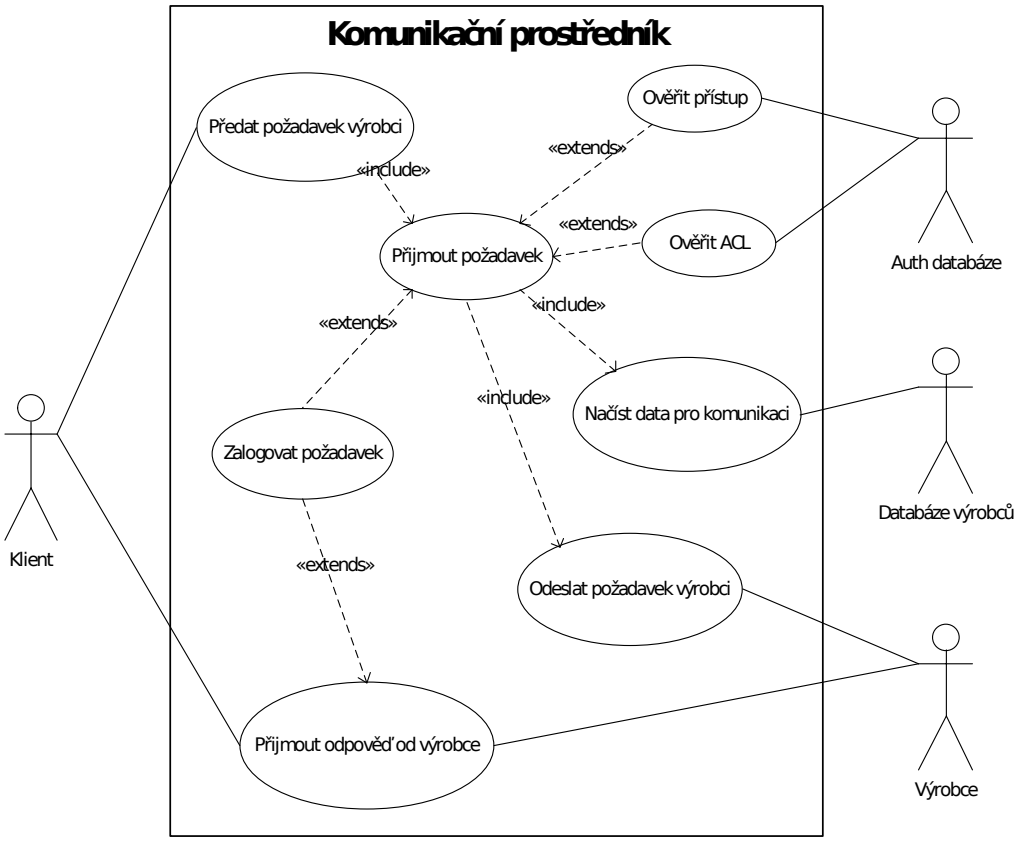
Případ užití	**Získat data o produktu**
Požadavek	A.4 Načtení dat o produktu online
Cíl	Získat další údaje o produktu od výrobce online.
Vstupní podmínky	Načtené ID tagu a známý výrobce (z dat uložených v tagu), uživatel může být přihlášen, aktivní online spojení.
Úspěšný výsledek	Doplněna data o produktu v paměti.
Neúspěšný výsledek	Stažení dat bylo neúspěšné.
Hlavní účastníci	Uživatel
Vedlejší účastníci	Komunikační prostředník
Akce	Uživatel chce stáhnout další data o produktu.
Hlavní scénář	1. Uživatel pomocí tlačítka (příp. může automaticky, pakliže je přihlášen) vyvolá stažení dalších dat o produktu. 2. Aplikace provede dotaz na službu komunikačního prostředníka spolu s načtenými údaji (identifikátor produktu a výrobce) a přihlášením uživatele (v případě nepřihlášeného použije univerzální údaje). Návratová data mohou obsahovat informace potřebné pro ověření dat v tagu. 3. Aplikace dostane odpověď s daty o produktu, doplní data uložená v paměti, která jsou následně zobrazena.
Možná změna	2. 1. Aplikace nemá uložena data pro ověření dat v tagu, žádá službu jejich poskytnutí.
Výjimky	2. a. Zařízení nemá aktivováno datové/síťové připojení. Aplikace vyzve k jeho aktivaci. 2. b. Komunikační prostředník neodpověděl do určité doby. Aplikace umožní odeslání opětovného dotazu či zrušení požadavku na online dotaz. 2. c. Odpověď komunikačního prostředníka obsahuje chybový stav (např. uživatel nemá práva k načítání dalšího obsahu) – dle typu chyby aplikace informuje uživatele vlastní hláškou.

Případ užití	**Zobrazit info o výrobci**
Požadavek	A.5 Načtení dat o výrobci
Cíl	Načíst a zobrazit uživateli doplňující informace o výrobci.
Vstupní podmínky	Známý identifikátor výrobce načtený z tagu, uživatel může být přihlášen, je aktivní online spojení.
Úspěšný výsledek	Zobrazeny detailní informace o výrobci.
Neúspěšný výsledek	Stažení dat bylo neúspěšné.
Hlavní účastníci	Uživatel
Vedlejší účastníci	Komunikační prostředník
Akce	Uživatel chce načíst a zobrazit podrobné informace o výrobci.
Hlavní scénář	1. Uživatel pomocí tlačítka vyvolá stažení dalších dat o výrobci. 2. *include::Získat info o výrobci*: Aplikace provede dotaz na službu komunikačního prostředníka spolu s načtenými údaji (identifikátor výrobce) a přihlášením uživatele (v případě nepřihlášeného použije univerzální údaje). 3. Aplikace dostane odpověď s daty o výrobci a zobrazí je v okně aplikace. Návratová data mohou obsahovat informace potřebné pro ověření dat v tagu.
Možná změna	2. 1. Aplikace nemá uložena data pro ověření dat v tagu, žádá službu jejich poskytnutí.

Výjimky	2. a. Zařízení nemá aktivováno datové/síťové připojení. Aplikace vyzve k jeho aktivaci. 2. b. Komunikační prostředník neodpověděl do určité doby. Aplikace umožní odeslání opětovného dotazu či zrušení požadavku na online dotaz. 2. c. Odpověď komunikačního prostředníka obsahuje chybový stav (např. uživatel nemá práva k načítání dalšího obsahu) – dle typu chyby aplikace informuje uživatele vlastní hláškou.

3.4.3 Klient – Komunikační prostředník

Požadavek na funkcionalitu z části B Server – webová služba zobrazuje na obr. 3.7 případ užití, kde je hlavním účastníkem *Klient* (*Klientská aplikace* nebo *Subjekt*), který používá



Obrázek 3.7: UC 2: Klient – Komunikační prostředník

Komunikačního prostředníka jako systém, jenž komunikuje s *Auth databází*, *Databází výrobců* a *Výrobcem* jako vedlejšími účastníky. Tento případ popisuje obecné dotazování a odpovědi na systém komunikačního prostředníka s jejich transformací na/od výrobce.

Případ užití	**Předat požadavek výrobci**
Požadavek	B.3 Autentizace, B.4 Autorizace, B.5 Komunikace se serverem výrobce
Cíl	Přijmout informace zaslané klientem a před je výrobci.

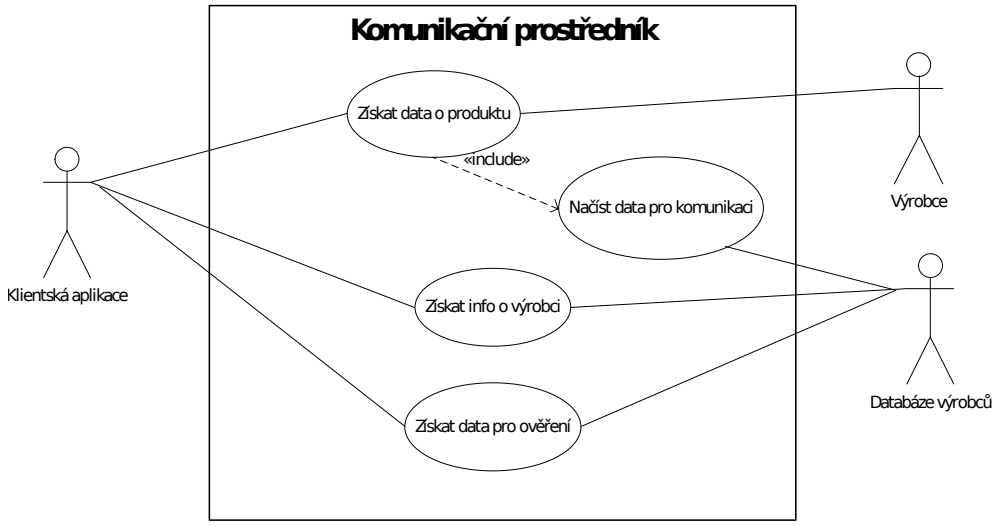
Vstupní podmínky	Klient využívá rozhraní komunikačního prostředníka, zasílá minimálně identifikátor výrobce
Úspěšný výsledek	Odeslaný požadavek předán výrobci.
Neúspěšný výsledek	Požadavek výrobci nelze odeslat.
Hlavní účastníci	Klient
Vedlejší účastníci	Auth databáze, Databáze výrobců, Výrobce
Akce	Zpracovat, zkontrolovat zprávu přijatou od klienta a předat ji dále výrobci.
Hlavní scénář	1. *include::Přijmout požadavek*: Aplikace přijme požadavek, zprávu klienta, provede její rozbalení a ověření formátu. 2. *extends::Ověřit přístup*: Aplikace v Auth databázi nalezne klienta a ověří jeho přístupové údaje. 3. *extends::Ověřit ACL*: Aplikace v Auth databázi ověřuje, zda nalezený klient může vykonat požadovanou akci. Případně jsou načtena i data pro sestavení dotazu. 4. *include::Načíst data pro komunikaci*: Aplikace v Databázi výrobců načítá výrobce dle zaslaného identifikátoru. Podle získaných informací sestaví požadavek pro výrobce. 5. *include::Odeslat požadavek Výrobci*: Aplikace odesílá požadavek na informační systém výrobce a čeká na odpověď.
Možná změna	1. 1. *extends::Zalogovat požadavek*: Aplikace může logovat přijatý požadavek.
Výjimky	1. a. Přijatá zpráva neodpovídá danému formátu, nelze ji rozbalit – aplikace odpovídá určenou chybou. 2. a. Klient nebyl nalezen v Auth databázi nebo jeho údaje jsou nesprávné – aplikace odpovídá určenou chybou. 3. a. Klient nemůže vykonat požadovanou akci – aplikace odpovídá určenou chybou. 4. a. Nebyl nalezen výrobce dle zaslaného identifikátoru – aplikace odpovídá určenou chybou. 4. b. Uložené údaje u výrobce nepostačují k sestavení požadavku (mohlo dojít k narušení databáze) – aplikace odpovídá chybou neočekávaného stavu. 5. a. Požadavek výrobci nemohl být odeslán (nebyla nalezena cílová URL apod.) – aplikace odpovídá chybou.

Případ užití	**Přijmout odpověď od výrobce**
Požadavek	B.7 Odpověď klientské aplikaci
Cíl	Odeslat odpověď klientovi na jeho požadavek.
Vstupní podmínky	Klient – Komunikační prostředník
Úspěšný výsledek	Klient přijal odpověď od výrobce, resp. komunikačního prostředníka.
Neúspěšný výsledek	Klient dostal chybovou zprávu.
Hlavní účastníci	Klient
Vedlejší účastníci	Výrobce
Akce	Po odeslání požadavku na výrobce je očekávána odpověď.
Hlavní scénář	1. Výrobce odpověděl na zaslaný požadavek. 2. Aplikace zkontroluje obsah zprávy (zda neobsahuje chybovou hlášku). 3. Aplikace z přijatých dat sestaví odpověď pro klienta a odešle ji.
Možná změna	1. 1. *extends::Zalogovat požadavek*: Aplikace může logovat přijatý požadavek.

Výjimky	1. a. Výrobce neodpověděl do stanovené doby – aplikace zasílá odpověď s chybou. 2. a. Odpověď od výrobce neobsahuje požadovaný formát dat – aplikace zasílá odpověď s neočekávanou chybou. 2. b. Odpověď výrobce obsahuje chybový stav – dle formátu a závažnosti chyby předává aplikace odpověď s chybou. 3. a. Z přijatých dat nelze sestavit odpověď (např. žádná nejsou) – aplikace posílá chybovou zprávu klientovi.

3.4.4 Klientská aplikace – Komunikační prostředník

Požadavek na funkcionalitu z části B.1 Služba pro klientskou aplikaci zobrazuje na obr. 3.8 případ užití, kde je hlavním účastníkem *Klientská aplikace*, která používá *Komunikačního prostředníka* jako systém, jenž komunikuje s *Databází výrobců* a *Výrobcem* jako vedlejšími účastníky. Tento případ neřeší vlastní ověřování přístupu, autorizaci či vlastní způsob komunikace (sestavování zpráv, jejich ověřování).



Obrázek 3.8: UC 3: Klientská aplikace – Komunikační prostředník

Případ užití	**Získat data o produktu**
Požadavek	B.5 Komunikace se serverem výrobce, B.7 Odpověď klientské aplikaci, B.8 Podpora ověření dat v tagu
Cíl	Zaslat požadavek na výrobce pro získání dalších informací o produktu, jeho stavu.
Vstupní podmínky	Zaslané ID tagu a identifikátor výrobce.
Úspěšný výsledek	Doplňující informace o produktu předaná klientské aplikaci.
Neúspěšný výsledek	Předání chybového hlášení klientské aplikaci.
Hlavní účastníci	Klientská aplikace
Vedlejší účastníci	Výrobce, Databáze výrobců
Akce	Klientskou aplikací zaslán požadavek na další informace o produktu.

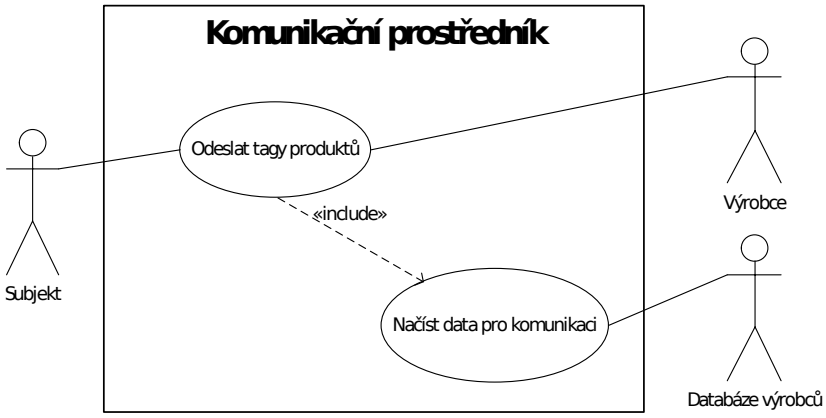
Hlavní scénář	1. Klientská aplikace zašle požadavek na komunikačního prostředníka s ID tagu, s identifikátorem výrobce a příp. otiskem ověřovacích dat používaných k ověření dat v tagu. 2. Komunikační prostředník přijímá požadavek, ověří jej, autentizuje, autorizuje klienta. 3. *include::Načíst data pro komunikaci*: Aplikace načítá data pro komunikaci s výrobcem z databáze výrobců. 4. Výrobci je odeslán požadavek na informace o produktu. 5. Výrobce odpovídá zaslanými informacemi. 6. Získané informace jsou předány klientské aplikaci spolu s daty pro ověřování dat v tagu.
Možná změna	1. 1. Klientská aplikace nemusí zasílat otisk ověřovacích dat. V tom případě se předpokládá, že je má uložena a ani odpověď neobsahuje žádnou informaci o nich, pokud o to aplikace výslovně nepožádala.
Výjimky	1. a. Nebyla přijata dostatečná data, něco chybí – aplikace odpovídá chybou. 2. a. Klient nebyl autentizován, autorizován – odpovědí je chyba. 3. a. Nebyl nalezen výrobce nebo údaje nepostačují k odeslání požadavku výrobci – aplikace odpovídá chybou. 5. a. Výrobce odpověděl chybou – dle závažnosti je předána do odpovědi. 6. a. Certifikát výrobce je neplatný, odesílá se nový (pokud existuje), jinak informace o jeho neplatnosti.

Případ užití	**Získat info o výrobci**
Požadavek	B.6 Získání informací o firmě, B.7 Odpověď klientské aplikaci
Cíl	Poskytnout další úložené informace o výrobci klientské aplikaci.
Vstupní podmínky	Zaslaný identifikátor výrobce.
Úspěšný výsledek	Data o výrobci předaná klientské aplikaci.
Neúspěšný výsledek	Předání chybového hlášení klientské aplikaci.
Hlavní účastníci	Klientská aplikace
Vedlejší účastníci	Databáze výrobců
Akce	Klientskou aplikací zaslán požadavek na informace o výrobci.
Hlavní scénář	1. Klientská aplikace zašle požadavek na komunikačního prostředníka s identifikátorem výrobce, případně otiskem ověřovacích dat používaných k ověření dat v tagu. 2. Komunikační prostředník přijímá požadavek, ověří jej, autentizuje, autorizuje klienta. 3. Aplikace načítá data o výrobci z databáze výrobců. 4. Získaná data jsou předána klientské aplikaci spolu s daty pro ověřování dat v tagu.
Možná změna	1. 1. Klientská aplikace nemusí zasílat otisk ověřovacích dat. V tom případě se předpokládá, že je má uložena a ani odpověď neobsahuje žádnou informaci o nich, pokud o to aplikace výslovně nepožádala.
Výjimky	1. a. Nebyla přijata dostatečná data, něco chybí – aplikace odpovídá chybou. 2. a. Klient nebyl autentizován, autorizován – odpovědí je chyba 3. a. Nebyl nalezen požadovaný výrobce – aplikace odpovídá chybou. 4. a. Certifikát výrobce je neplatný, odesílá se nový (pokud existuje), jinak informace o jeho neplatnosti.

Případ užití	**Získat data pro ověřrní**
Požadavek	B.8 Podpora ověření dat v tagu. B.7 Odpověď klientské aplikaci
Cíl	Poskytnout platná data pro ověření obsahu tagu klientské aplikaci.
Vstupní podmínky	Zaslaný identifikátor výrobce
Úspěšný výsledek	Data potřebná k ověření předána klientské aplikaci.
Neúspěšný výsledek	Předání chybového hlášení klientské aplikaci.
Hlavní účastníci	Klientská aplikace
Vedlejší účastníci	Databáze výrobců
Akce	Klientskou aplikací zaslán požadavek na ověření/stažení ověřovacích dat.
Hlavní scénář	1. Klientská aplikace zašle požadavek na komunikačního prostředníka s identifikátorem výrobce a požadavkem na poskytnutí dat pro ověření obsahu tagu (příp. jejich otiskem). 2. Komunikační prostředník přijímá požadavek, ověří jej, autentizuje, autorizuje klienta. 3. Aplikace načítá potřebné informace z databáze výrobců. 4. Získané informace jsou předány klientské aplikaci.
Možná změna	1. 1. Klientská aplikace může vyžádat jen ověření svých uložených dat nebo jejich kompletní poskytnutí.
Výjimky	1. a. Nebyla přijata dostatečná data, něco chybí – aplikace odpovídá chybou. 2. a. Klient nebyl autentizován, autorizován – odpovědí je chyba. 3. a. Nebyl nalezen požadovaný výrobce – aplikace odpovídá chybou. 4. a. Ověřovací data jsou neplatná, odesílá se nová sada (pokud existuje), jinak informace o tom, že neexistují.

3.4.5 Subjekt – Komunikační prostředník

V předchozí části byl uveden pohled klientské aplikace vůči systému *Komunikačního prostředníka*. Další službu z části B.2 Služba pro aktualizaci stavu produktu zobrazuje obr. 3.9 obsahující případ užití s hlavním účastníkem *Subjektem*, který používá *Komunikačního pro-*



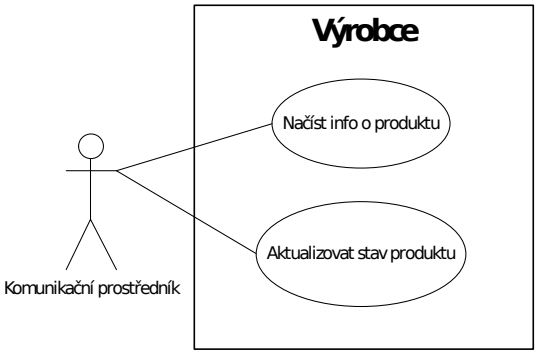
Obrázek 3.9: UC 4: Subjekt – Komunikační prostředník

středníka jako systém, jenž komunikuje s *Databází výrobců* a *Výrobcem* jako vedlejšími účastníky. *Subjekt* je v tomto případu zobecněním různých článků distribučního řetězce: distributora a obchodu. Tento případ, stejně jako předchozí, neřeší vlastní ověřování přístupu, autorizaci či vlastní způsob komunikace (sestavování zpráv, jejich ověřování).

Případ užití	**Odeslat tagy produktů**
Požadavek	B.9 Rozhraní pro zaslání seznamu produktů, B.10 Rozhraní pro zaslání distribuční jednotky, B.5 Komunikace se serverem výrobce, B.11 Odpověď klientovi
Cíl	Předat výrobci seznam produktů nebo identifikaci distribuční jednotky a tím aktualizovat stav produktu.
Vstupní podmínky	Zaslaná ID tagů či identifikátor distribuční jednotky a identifikátor výrobce.
Úspěšný výsledek	Potvrzující zpráva předaná subjektu.
Neúspěšný výsledek	Předání chybového hlášení subjektu.
Hlavní účastníci	Subjekt
Vedlejší účastníci	Výrobce, Databáze výrobců
Akce	Subjektem zaslána zpráva pro aktualizaci stavu produktu u výrobce.
Hlavní scénář	1. Subjekt zašle požadavek na komunikačního prostředníka s identifikátorem výrobce a seznamem ID tagů nebo identifikací distribuční jednotky. 2. Komunikační prostředník přijímá požadavek, ověří jej, autentizuje, autorizuje klienta. 3. *include::Načíst data pro komunikaci*: Aplikace načítá data pro komunikaci s výrobcem z databáze výrobců. 4. Výrobci je předán obsah přijaté zprávy. 5. Výrobce odpovídá potvrzující zprávou. 6. Subjektu je předána potvrzující zpráva.
Možná změna	1. 1. Subjekt zasílá seznam ID tagů podle dodacího listu, nebo identifikátor distribuční jednotky. Výrobce pak ví, které produkty jsou v distribuční jednotce zahrnuty.
Výjimky	1. a. Nebyla přijata dostatečná data, něco chybí – aplikace odpovídá chybou. 2. a. Klient nebyl autentizován, autorizován – odpovědí je chyba. 3. a. Nebyl nalezen výrobce nebo údaje nepostačují k odeslání zprávy výrobci – aplikace odpovídá chybou. 5. a. Výrobce odpověděl chybou – dle závažnosti je předána do odpovědi.

3.4.6 Komunikační prostředník – Výrobce

Požadavek na prototyp výrobce z části D Prototyp aplikace výrobce zahrnuje případ užití zobrazený na obr. 3.10, kde je hlavním účastníkem *Komunikační prostředník*, který pracuje v systému *Výrobce*. Tento případ neřeší detailní způsob, jakým spolu účastník a systém komunikují. Zaměřuje se pouze na vztah použitých systémů.



Obrázek 3.10: UC 5: Komunikační prostředník – Výrobce

Případ užití	**Načíst info o produktu**
Požadavek	D.2 Rozhraní pro načtení informací o produktu
Cíl	Získat z IS výrobce detailní informace, stav produktu.
Vstupní podmínky	Známé ID tagu produktu, funkční spojení.
Úspěšný výsledek	Získaná doplňující data o produktu.
Neúspěšný výsledek	Chybový stav.
Hlavní účastníci	Komunikační prostředník
Vedlejší účastníci	*nejsou*
Akce	Komunikační prostředník položí dotaz výrobci.
Hlavní scénář	1. Komunikační prostředník vytvoří požadavek a odešle ho. 2. Výrobce požadavek přijme, zpracuje ho. Nalezne požadovaný produkt, načte dostupné informace a vrátí je zpět v odpovědi. 3. Komunikační prostředník přijme odpověď.
Možná změna	*není*
Výjimky	1. a. Výrobce odmítne požadavek – má nesprávný formát, chybná data. Odpovědí je chybový stav. 2. a. Požadovaný produkt u výrobce neexistuje, odpověď obsahuje chybový stav. 3. a. Komunikační prostředník nedostane v požadovaném limitu odpověď.

Případ užití	**Aktualizovat stav produktu**
Požadavek	D.3 Rozhraní pro aktualizaci stavu produktu
Cíl	Zaslat IS výrobci aktualizaci stavu produktu.
Vstupní podmínky	Známé ID tagu produktu či identifikátor distribuční jednotky, funkční spojení.
Úspěšný výsledek	Potvrzující odpověď od výrobce.
Neúspěšný výsledek	Chybový stav.
Hlavní účastníci	Komunikační prostředník
Vedlejší účastníci	*nejsou*
Akce	Komunikační prostředník odesílá data výrobci.
Hlavní scénář	1. Komunikační prostředník vytvoří zprávu o daném obsahu (ID tagů nebo distribuční jednotky) a odešle ho. 2. Výrobce požadavek přijme, zpracuje ho. Odpověď obsahuje potvrzení příjmu dat. 3. Komunikační prostředník přijme odpověď.
Možná změna	*není*
Výjimky	2. a. Výrobce odmítne požadavek – má nesprávný formát, chybná data. Odpovědí je chybový stav. 2. b. Požadovaný produkt nebo identifikátor distribuční jednotky u výrobce neexistuje, odpověď obsahuje chybový stav. 3. a. Komunikační prostředník nedostane v požadovaném limitu odpověď.

3.4.7 Administrátor – Admin aplikace

Součástí systému je i požadavek na Administrační rozhraní (Admin aplikaci) definovaný v části C Server – administrační rozhraní (Admin aplikace). Pro tento požadavek je případ užití zobrazený na obr. 3.11, kde je hlavním účastníkem *Administrátor*, tj. fyzická osoba, která pracuje v systému *Admin aplikace*. Vedlejšími účastníky jsou pak *Auth databáze* a *Databáze výrobců*, které *Admin aplikace* používá ke své funkcionalitě. Vlastní administrace slouží pouze

pro snazší přístup k datům uloženým v databázi. Jednotlivé případy a scénáře k nim jsou tak napsány obecněji, neboť daná část není pro celý projekt stěžejní a způsob fungování jednotlivých akcí si lze snadno odvodit.

Obrázek 3.11: UC 6: Administrátor – Admin aplikace

Případ užití	**Přihlásit**
Požadavek	C.1 Přihlášení, C.2 Autorizace
Cíl	Přihlásit administrátora do systému.
Vstupní podmínky	Administrátor není přihlášen, ale disponuje přihlašovacími údaji.
Úspěšný výsledek	Administrátor je přihlášen do systému.
Neúspěšný výsledek	Přihlášení administrátora bylo zamítnuto.
Hlavní účastníci	Administrátor
Vedlejší účastníci	Auth databáze
Akce	Administrátor se přihlašuje do Admin aplikace.
Hlavní scénář	1. Administrátor zadává své přihlašovací údaje do aplikace. 2. Po zadání údajů aplikace kontroluje jejich správnost pomocí Auth databáze. 3. Pokud jsou údaje správné, administrátor je přihlášen.
Možná změna	1. 1. Po vyplnění přihlašovacích údajů může dojít k dvoufaktorové autentizaci, tj. použití např. certifikátu uloženého v prohlížeči nebo ověření pomocí tzv. tokenu (známého z bankovních systémů) nebo ověřením pomocí klíče zaslaného na mobilní telefon administrátora.

Výjimky	1. a. Druhý stupeň dvoufaktorové autentizace selhal, přihlášení je neúspěšné. 2. a. Administrátor zadal chybné údaje, přihlášení je neúspěšné. 2. b. Administrátor sice zadal správné údaje, ale při autorizaci mu byl zablokován přístup. Přihlášení je tak neúspěšné.

Případ užití	**Odhlásit**
Požadavek	C.1 Přihlášení
Cíl	Odhlásit administrátora ze systému.
Vstupní podmínky	Administrátor je přihlášen.
Úspěšný výsledek	Administrátor je odhlášen.
Neúspěšný výsledek	*není*
Hlavní účastníci	Administrátor
Vedlejší účastníci	*nejsou*
Akce	Administrátor se odhlašuje z Admin aplikace.
Hlavní scénář	1. Administrátor použije volbu pro odhlášení. 2. Aplikace se postará o smazání údajů o přihlášení, zaloguje odhlášení a provede přesměrování na přihlašovací obrazovku.
Možná změna	1. 1. Administrátor může být odhlášen po delší době nečinnosti.
Výjimky	1. a. Při požadavku na odhlášení již vypršela platnost přihlášení. Přesto je administrátor odhlášen, bez chybové hlášky.

Případ užití	**Spravovat výrobce**
Požadavek	C.4 Správa firem, výrobců
Cíl	Vyhledávat ve výrobcích, přidávat je, upravovat a odebírat.
Vstupní podmínky	Administrátor je přihlášen do systému a má požadované oprávnění.
Úspěšný výsledek	Výrobce je nalezen, přidán, upraven, odebrán.
Neúspěšný výsledek	Došlo k chybě, administrátor zadal nesprávné údaje.
Hlavní účastníci	Administrátor
Vedlejší účastníci	Databáze výrobců
Akce	Administrátor chce provést určitý úkon s databází výrobců.

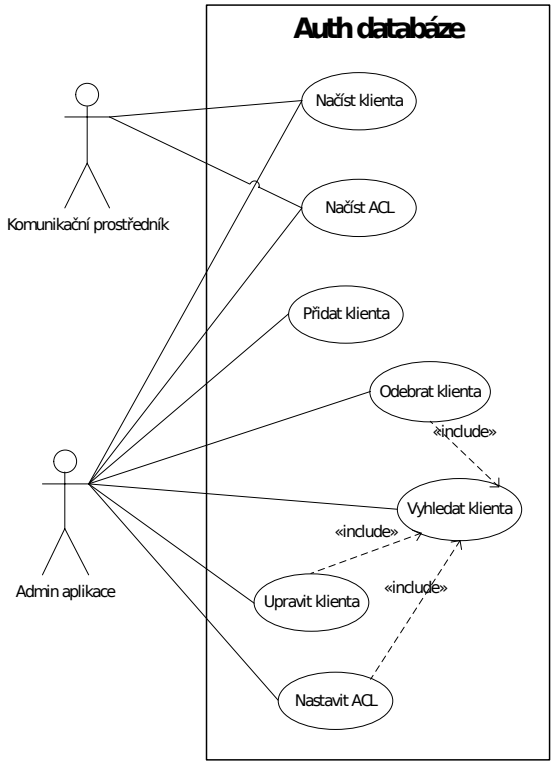
Hlavní scénář	– *include::Přidat výrobce*: Administrátor použije volbu pro přidání výrobce, vyplní požadované údaje a odešle formulář.
	– *include::Vyhledat výrobce*: Administrátor zadá do pole hledaný řetězec (např. název, IČ výrobce) a potvrdí hledání. Zobrazeny jsou nalezení výrobci.
	  – *include::Odebrat výrobce*: Administrátor použije volbu pro odebrání výrobce z nalezeného seznamu. Volba je potvrzena. Jeho úkon je zalogován.
	  – *include::Upravit výrobce*: Administrátor použije volbu pro editaci výrobce z nalezeného seznamu. Otevře se mu formulář pro editaci údajů o výrobci. Po jeho odeslání jsou provedeny změny a úkon je zalogován.

Možná změna	*není*
Výjimky	– Administrátor nemá právo pro přístup k některým akcím – je upozorněn chybovou hláškou. – Do některého z formulářů administrátor zadal neplatné údaje. Je upozorněn chybou a požadována je oprava dat. – Při editaci došlo k systémové chybě (např. výpadek spojení s databází výrobců). Administrátor je informován neočekávaným stavem.

Případ užití	**Spravovat klienty**
Požadavek	C.3 Správa uživatelů, klientů
Cíl	Vyhledávat v klientech, přidávat je, upravovat a odebírat, nastavovat přístup a práva.
Vstupní podmínky	Administrátor je přihlášen do systému a má požadované oprávnění.
Úspěšný výsledek	Klient je nalezen, přidán, upraven, odebrán, jsou mu nastavena práva.
Neúspěšný výsledek	Došlo k chybě, administrátor zadal nesprávné údaje.
Hlavní účastníci	Administrátor
Vedlejší účastníci	Auth databáze
Akce	Administrátor chce provést určitý úkon s auth databází.
Hlavní scénář	– *include::Přidat klienta*: Administrátor použije volbu pro přidání klienta, vyplní požadované přihlašovací údaje, nastaví práva a odešle formulář – *include::Vyhledat klienta*: Administrátor zadá do pole hledaný řetězec (např. uživatelské jméno, vlastní jméno klienta) a potvrdí hledání. Zobrazeny jsou nalezení klienti (tj. i další administrátoři). – *include::Odebrat klienta*: Administrátor použije volbu pro odebrání klienta z nalezeného seznamu. Volba je potvrzena. Jeho úkon je zalogován. Administrátor může odebírat pouze klienty definovaných oprávnění (nižších) a nemůže odebrat sám sebe. – *include::Upravit klienta*: Administrátor použije volbu pro editaci klienta z nalezeného seznamu. Otevře se mu formulář pro editaci údajů o klientovi. Po jeho odeslání jsou provedeny změny a úkon je zalogován. – *include::Nastavit ACL*: Administrátor použije volbu pro nastavení autorizačních údajů (ACL) klienta z nalezeného seznamu. Otevře se mu formulář pro editaci práv klienta. Administrátor může nastavovat pouze oprávnění klientům s nižšími právy než je on sám a nemůže nastavovat oprávnění sám sobě. Po jeho odeslání jsou provedeny změny a úkon je zalogován.
Možná změna	*není*
Výjimky	– Administrátor nemá právo pro přístup k některým akcím – je upozorněn chybovou hláškou. – Do některého z formulářů administrátor zadal neplatné údaje. Je upozorněn chybou a požadována je oprava dat. – Při editaci došlo k systémové chybě (např. výpadek spojení s auth databází). Administrátor je informován neočekávaným stavem.

3.4.8 Auth databáze

Pouze konceptuálně oddělená je *Auth databáze*, jinak je součástí jedné fyzické databáze systému. Obsahuje autentizační a autorizační údaje klientů a administrátorů. V následujícím případu užití na obr. 3.12 figurují jako hlavní účastníci *Komunikační prostředník* a *Admin aplikace*, kteří *Auth databázi* využívají pro svou práci.



Obrázek 3.12: UC 7: Auth databáze

Případ užití	**Načíst klienta**
Požadavek	B.3 Autentizace, C.1 Přihlášení
Cíl	Ověřit existenci klienta.
Vstupní podmínky	Údaje pravděpodobně existujícího klienta.
Úspěšný výsledek	Klient nalezen.
Neúspěšný výsledek	Klient nenalezen.
Hlavní účastníci	Komunikační prostředník, Admin aplikace
Vedlejší účastníci	*nejsou*
Akce	Načíst přihlašovací údaje klienta, administrátora.
Hlavní scénář	1. Komunikační prostředník nebo admin aplikace položí dotaz do Auth databáze. 2. Auth databáze na základě dotazu vyhledá záznam uživatele (klienta, administrátora) a vrátí jeho údaje.
Možná změna	2. 1. Auth databáze přímo ověřuje i přihlašovací údaje klienta, administrátora. Pokud jsou nesprávné, nevrací žádný výsledek.
Výjimky	1. a. Auth databáze není dostupná – neočekávaná chyba. 2. a. Klient, administrátor nebyl v databázi nalezen.

Případ užití	**Načíst ACL**
Požadavek	B.4 Autorizace, C.2 Autorizace
Cíl	Ověřit práva klienta, administrátora k dané akci.
Vstupní podmínky	Identifikace klienta, administrátora a identifikátor dané akce.
Úspěšný výsledek	Práva existují.
Neúspěšný výsledek	Práva neexistují.

Hlavní účastníci	Komunikační prostředník, Admin aplikace
Vedlejší účastníci	*nejsou*
Akce	Načíst ACL pro daného klienta, administrátora.
Hlavní scénář	1. Komunikační prostředník, Admin aplikace položí dotaz do Auth databáze. 2. Auth databáze na základě dotazu vyhledá práva pro daného klienta, administrátora a vrátí výsledek (právo existuje/neexistuje).
Možná změna	2. 1. Auth databáze vrací kompletní seznam práv, komunikační prostředník, Admin aplikace se rozhodují až na jejich základě.
Výjimky	1. a. Auth databáze není dostupná – neočekávaná chyba. 2. a. Klient, administrátor nebyl v databázi nalezen.

Případ užití	**Přidat klienta**
Požadavek	C.3 Správa uživatelů, klientů
Cíl	Přidat nového Klienta do Auth databáze.
Vstupní podmínky	Korektní údaje nového klienta.
Úspěšný výsledek	Klient založen.
Neúspěšný výsledek	Údaje nebyly správné, chyba při ukládání.
Hlavní účastníci	Admin aplikace
Vedlejší účastníci	*nejsou*
Akce	Admin aplikace přidává nového klienta.
Hlavní scénář	1. Admin aplikace vytvoří dotaz pro založení nového klienta s jeho údaji. 2. Dotaz je odeslán Auth databázi. 3. Auth databáze potvrdí vložení nového záznamu.
Možná změna	1. 1. Záznam je nejdříve založen, poté doplněn o další údaje. 3. 2. Auth databáze loguje založení záznamů.
Výjimky	2. a. Auth databáze není dostupná – neočekávaná chyba. 3. a. Klient s danými identifikačními údaji již existuje (nelze mít dvě stejná uživatelská jména). 3. b. Došlo k porušení integrity dat při zakládání klienta (zvolením nesprávného ID).

Případ užití	**Odebrat klienta**
Požadavek	C.3 Správa uživatelů, klientů
Cíl	Odebrat klienta z Auth databáze.
Vstupní podmínky	Identifikátor existujícího klienta.
Úspěšný výsledek	Klient odebrán.
Neúspěšný výsledek	Klient nebyl nalezen, neočekávaný stav.
Hlavní účastníci	Admin aplikace
Vedlejší účastníci	*nejsou*
Akce	Admin aplikace odebírá klienta.
Hlavní scénář	1. *include::Vyhledat klienta*: Admin aplikace nejdříve vyhledá v databázi klienta, kterého chce smazat. 2. Admin aplikace vytvoří dotaz pro odebrání existujícího klienta pomocí jeho unikátního identifikátoru. 3. Dotaz je odeslán Auth databázi. 4. Auth databáze potvrdí odebrání záznamu.
Možná změna	4. 1. Auth databáze loguje odebrání záznamů.

Výjimky	3. a. Auth databáze není dostupná – neočekávaná chyba. 4. a. Klient s daným identifikátorem neexistuje. 4. b. Došlo k porušení integrity dat při odebírání klienta (na klienta jsou navázány informace, která nelze odebrat).

Případ užití	**Vyhledat klienta**
Požadavek	C.3 Správa uživatelů, klientů
Cíl	Vyhledat klienta v Auth databázi.
Vstupní podmínky	Platný vyhledávací řetězec.
Úspěšný výsledek	Nalezen seznam klientů.
Neúspěšný výsledek	Nalezen žádný klient.
Hlavní účastníci	Admin aplikace
Vedlejší účastníci	*nejsou*
Akce	Admin aplikace vyhledává klienta.
Hlavní scénář	1. Admin aplikace vytvoří dotaz pro vyhledání klienta pomocí vyhledávacích kritérií. 2. Dotaz je odeslán Auth databázi. 3. Auth databáze vrátí seznam nalezených klientů.
Možná změna	3. 1. Auth databáze loguje vyhledávání záznamů.
Výjimky	2. a. Auth databáze není dostupná – neočekávaná chyba. 3. a. Pro daný dotaz nebyla nalezen žádný klient.

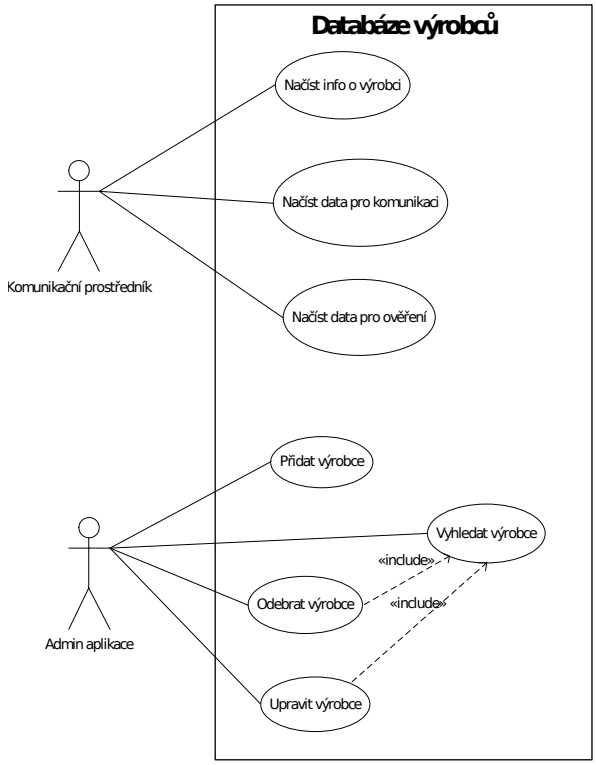
Případ užití	**Upravit klienta**
Požadavek	C.3 Správa uživatelů, klientů
Cíl	Upravit klienta v Auth databázi.
Vstupní podmínky	Identifikátor existujícího klienta.
Úspěšný výsledek	Klientovy údaje upraveny.
Neúspěšný výsledek	Klient nebyl nalezen, neočekávaný stav.
Hlavní účastníci	Admin aplikace
Vedlejší účastníci	*nejsou*
Akce	Admin aplikace upravuje existujícího klienta.
Hlavní scénář	1. *include::Vyhledat klienta*: Admin aplikace nejdříve vyhledá v databázi klienta, kterého chce upravit. 2. Admin aplikace vytvoří dotaz s upravenými údaji klienta doplněnými o jeho unikátní identifikátor. 3. Dotaz je odeslán Auth databázi. 4. Auth databáze potvrdí upravení záznamu.
Možná změna	4. 1. Auth databáze loguje úpravu záznamů.
Výjimky	3. a. Auth databáze není dostupná – neočekávaná chyba. 3. a. Klient s daným identifikátorem neexistuje. 3. b. Došlo k porušení integrity dat (např. je editováno uživatelské jméno a klient s daným uživ. jménem již existuje).

Případ užití	**Nastavit ACL**
Požadavek	C.3 Správa uživatelů, klientů
Cíl	Upravit přístupová práva klienta v Auth databázi.
Vstupní podmínky	Identifikátor existujícího klienta.
Úspěšný výsledek	Klientovi jsou upravena přístupová práva.
Neúspěšný výsledek	Klient nebyl nalezen, neočekávaný stav.
Hlavní účastníci	Admin aplikace
Vedlejší účastníci	*nejsou*

Akce	Admin aplikace nastavuje práva klienta.
Hlavní scénář	1. *include::Vyhledat klienta*: Admin aplikace nejdříve vyhledá v databázi klienta, jehož práva chce upravit. 2. Admin aplikace vytvoří dotaz pro úpravu práv existujícího klienta a přidává jeho unikátní identifikátor. 3. Dotaz je odeslán Auth databázi. 4. Auth databáze potvrdí upravení záznamu.
Možná změna	4. 1. Auth databáze loguje úpravu záznamů.
Výjimky	3. a. Auth databáze není dostupná – neočekávaná chyba. 3. a. Klient s daným identifikátorem neexistuje. 3. b. Došlo k porušení integrity dat (např. požadované právo v databázi neexistuje).

3.4.9 Databáze výrobců

Druhou částí databáze je *Databáze výrobců*. Ta obsahuje informace o výrobcích (firemní, kontaktní údaje, údaje pro komunikaci, data pro ověření obsahu tagu). V následujícím případu užití na obr. 3.13 figurují jako hlavní účastníci *Komunikační prostředník* a *Admin aplikace*, kteří systém *Databáze výrobců* používají pro svou činnost.



Obrázek 3.13: UC 8: Databáze výrobců

Případ užití	**Načíst info o výrobci**
Požadavek	B.6 Získání informací o firmě
Cíl	Načíst uložené informace o výrobci.
Vstupní podmínky	Identifikátor existujícího výrobce.

Úspěšný výsledek	Výrobce nalezen, vrácena jeho data.
Neúspěšný výsledek	Výrobce nenalezen
Hlavní účastníci	Komunikační prostředník
Vedlejší účastníci	*nejsou*
Akce	Načíst data o výrobci.
Hlavní scénář	1. Komunikační prostředník položí dotaz do Databáze výrobců. 2. Databáze výrobců na základě dotazu vyhledá výrobce a vrátí informace o něm.
Možná změna	*není*
Výjimky	1. a. Databáze výrobců není dostupná – neočekávaná chyba. 2. a. Výrobce nebyl v databázi nalezen.

Případ užití	**Načíst data pro ověření**
Požadavek	B.8 Podpora ověření dat v tagu
Cíl	Načíst údaje pro ověření platnosti dat v tagu.
Vstupní podmínky	Identifikátor existujícího výrobce.
Úspěšný výsledek	Výrobce nalezen, vráceny informace o ověřovacích datech.
Neúspěšný výsledek	Výrobce nenalezen.
Hlavní účastníci	Komunikační prostředník
Vedlejší účastníci	*nejsou*
Akce	Načíst ověřovací data pro ověření obsahu tagu.
Hlavní scénář	1. Komunikační prostředník položí dotaz do Databáze výrobců. 2. Databáze výrobců na základě dotazu vyhledá výrobce a k němu příslušná data. Návratem jsou ověřovací data.
Možná změna	*není*
Výjimky	1. a. Databáze výrobců není dostupná – neočekávaná chyba. 2. a. Výrobce nebyl v databázi nalezen. 2. b. Ověřovací data nebyla v databázi nalezena.

Případ užití	**Přidat výrobce**
Požadavek	C.4 Správa firem, výrobců
Cíl	Přidat nového výrobce do Databáze výrobců.
Vstupní podmínky	Korektní údaje nového výrobce
Úspěšný výsledek	Výrobce založen.
Neúspěšný výsledek	Údaje nebyly správné, chyba při ukládání.
Hlavní účastníci	Admin aplikace
Vedlejší účastníci	*nejsou*
Akce	Admin aplikace přidává nového výrobce.
Hlavní scénář	1. Admin aplikace vytvoří dotaz pro založení nového výrobce s jeho údaji. 2. Dotaz je odeslán Databázi výrobců. 3. Databáze výrobců potvrdí vložení nového záznamu.
Možná změna	1. 1. Záznam je nejdříve založen, poté doplněn o další údaje. 3. 2. Databáze výrobců loguje založení záznamů.
Výjimky	2. a. Databáze výrobců není dostupná – neočekávaná chyba. 3. a. Výrobce s danými údaji (IČ) již existuje. 3. b. Došlo k porušení integrity dat při zakládání výrobce (zvolením nesprávného unikátního identifikátoru).

Případ užití	**Odebrat výrobce**
Požadavek	C.4 Správa firem, výrobců
Cíl	Odebrat výrobce z Databázi výrobců.
Vstupní podmínky	Identifikátor existujícího výrobce.
Úspěšný výsledek	Výrobce odebrán.
Neúspěšný výsledek	Výrobce nebyl nalazen, neočekávaný stav.
Hlavní účastníci	Admin aplikace
Vedlejší účastníci	*nejsou*
Akce	Admin aplikace odebírá uloženého výrobce.
Hlavní scénář	1. *include::Vyhledat výrobce*: Admin aplikace nejdříve vyhledá v databázi výrobce, kterého chce smazat. 2. Admin aplikace vytvoří dotaz pro odebrání existujícího výrobce pomocí jeho unikátního identifikátoru. 3. Dotaz je odeslán Databázi výrobců. 4. Databáze výrobců potvrdí odebrání záznamu.
Možná změna	4. 1. Databáze výrobců loguje založení záznamů.
Výjimky	3. a. Databáze výrobců není dostupná – neočekávaná chyba. 4. a. Výrobce s daným identifikátorem neexistuje. 4. b. Došlo k porušení integrity dat při odebírání výrobce (na výrobce jsou navázány informace, které nelze odebrat).

Případ užití	**Vyhledat výrobce**
Požadavek	C.4 Správa firem, výrobců
Cíl	Vyhledat výrobce v Databázi výrobců.
Vstupní podmínky	Platný vyhledávací řetězec.
Úspěšný výsledek	Nalezen seznam výrobců.
Neúspěšný výsledek	Nenalezen žádný výrobce.
Hlavní účastníci	Admin aplikace
Vedlejší účastníci	*nejsou*
Akce	Admin aplikace vyhledává mezi výrobci.
Hlavní scénář	1. Admin aplikace vytvoří dotaz pro vyhledání výrobce pomocí vyhledávacích kritérií. 2. Dotaz je odeslán Databázi výrobců. 3. Databáze výrobců vrátí seznam nalezených výrobců.
Možná změna	3. 1. Databáze výrobců loguje vyhledávání záznamů.
Výjimky	2. a. Databáze výrobců není dostupná – neočekávaná chyba. 3. a. Pro daný dotaz nebyl nalezen žádný výrobce.

Případ užití	**Upravit výrobce**
Požadavek	C.4 Správa firem, výrobců
Cíl	Upravit výrobce v Databázi výrobců.
Vstupní podmínky	Identifikátor existujícího výrobce.
Úspěšný výsledek	Výrobce upraven.
Neúspěšný výsledek	Výrobce nebyl nalezen, neočekávaný stav.
Hlavní účastníci	Admin aplikace
Vedlejší účastníci	*nejsou*
Akce	Admin aplikace upravuje existujícího výrobce.

Hlavní scénář	1. *include::Vyhledat výrobce*: Admin aplikace nejdříve vyhledá v databázi výrobce, kterého chce upravit. 2. Admin aplikace vytvoří dotaz s upravenými údaji výrobce doplněnými o jeho unikátní identifikátor. 3. Dotaz je odeslán Databázi výrobců. 4. Databáze výrobců potvrdí upravení záznamu.
Možná změna	4. 1. Databáze výrobců loguje úpravu záznamu.
Výjimky	3. a. Databáze výrobců není dostupná – neočekávaná chyba. 4. a. Výrobce s daným identifikátorem neexistuje. 4. b. Došlo k porušení integrity dat (např. jsou zaslány prázdné údaje, které prázdné být nemají).

3.5 Použité technologie

Při specifikaci v části 2.5.4 byly nastíněny některé technologie, které hodláme v projektu použít. Provedeme nyní podrobnou diskuzi a návrh použití konkrétních technologií, jazyků, databází, abychom dosáhli požadovaného výsledku.

3.5.1 Klientská aplikace

Vývoj pro OS Android probíhá především v jazyce Java. Operační systém obsahuje vlastní implementaci Java Virtual Machine (JVM), která podporuje veškerou funkcionalitu Java SE (Standard Edition). K dispozici je velmi rozsáhlé API pro používání periferií v zařízení i dalších aplikací. Pro ukládání dat slouží SQLite databáze, zpřístupnit je možno interní či externí (SD karta) paměť telefonu. Pro konfiguraci a specifikaci rozvržení obrazovek se využívá popis pomocí jazyka XML. Ačkoliv existují kompilátory, které umožňují vytvářet program v jiném programovacím jazyce než Java a následně je zkompilovat pro běh nad JVM, budeme se držet jazyka primárně určeného pro tuto platformu.

Při vývoji mobilních aplikací se můžeme setkat také z řadou frameworků, které mohou ulehčit práci (např. využitím univerzálních prvků v grafickém rozhraní), popř. umí vytvořený kód zkompilovat i pro jiné platformy než pro OS Android. Jejich nevýhodou bývá nutnost používání omezené množiny funkcí, někdy i vlastního jazyka. Zároveň se můžeme setkat i s nefunkčností některých komponent na vlastním zařízení. Protože cílem práce není vytvoření širokospektré aplikace, budeme se držet striktního návrhu pro OS Android podle nejlepších technik pro to určených.

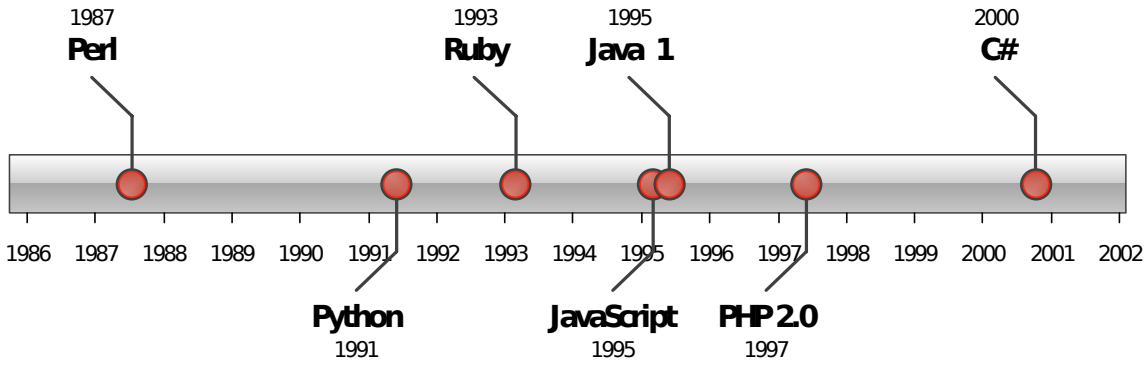
Vytvořená mobilní aplikace má navíc sloužit pouze jako prototyp, část celého fungujícího celku. Architekturu navrhneme navíc s tím, že části funkcionalit aplikace bude možné přenést do mnohem propracovanějšího programu, který již bude v budoucnu plnit ideály tohoto projektu.

3.5.2 Serverové aplikace

V rámci projektu budou vytvořeny dvě serverové aplikace: server komunikačního prostředníka a ukázka IS výrobce. Stejně jako u klientské aplikace, nabízí se zde jazyk Java, konkrétně balík standardů Java EE (Enterprise Edition) pro vytváření webových, serverových, aplikací.

Předpokládá se, že cílový IS výrobce bude v nějakém takovém jazyku vytvořen. Existující IS se většinou skládají z mnoha částí, které jsou firmám často za poplatek postupně dodávány. Každá společnost má navíc od produktu tohoto typu trochu jiná očekávání a tak dochází k personalizaci řešení, v některých případech byl pro tento případ vyvinut vlastní programovací jazyk (ABAP v SAPu).

Programovací jazyk Java má za sebou poměrně dlouhou historii a kromě mobilních zařízení se s ním můžeme setkat již i na embedded systémech, kde poskytuje poměrně snadný vývoj jakékoliv funkcionality, kterou daná JVM podporuje. Webové aplikace jsou v něm programovány poměrně často, opět díky snadnému vývoji a rozsáhlému balíku hotových a využitelných řešení. Nikdo se tak nemusí starat, jak vytvořit například připojení k databázi nebo vykreslit webový formulář. Snadno se použije již hotový framework (nebo jejich kombinace), který veškerou rutinní práci odvede za vývojáře sám. Často tak bohužel dochází k překombinování použitých knihoven a frameworků, čímž roste složitost projektu či hardwarová náročnost (ať už na samotnou aplikaci na serveru, tak klidně i na klientské skripty u uživatelů). Z tohoto pohledu není jazyk Java zcela vhodný pro velmi malé až střední aplikace, kde se očekává rychlost a jednoduchost a není třeba vysoké míry abstrakce (zajišťující lepší rozšiřitelnost). V tomto segmentu se uchytily většinou skriptovací jazyky, PHP, Perl, JavaScript, Ruby či Python.



Obrázek 3.14: Časová osa vzniku diskutovaných jazyků [42]

Kromě jazyka Java se nabízí ještě využití platformy .NET a některého z podporovaných jazyků (typicky C#). Tato varianta však nesplňuje stanovený požadavek na použití open-source řešení, ačkoliv existují volné implementace této technologie například v podobě projektu Mono. Zůstaňme však raději u univerzálnosti jazyka Java a dalších možných, většinou již skriptovacích. Zcela nepoužitelný je také jazyk C/C++, pomocí něhož bychom možná vytvořili službu komunikačního prostředníka, ale na veškeré další části bychom museli zvolit jazyk jiný.

Skriptovací jazyky jsou většinou dynamicky typované a v některých případech i typově nebezpečné. To nahrává do karet mnohým programátorům, kteří se neradi o deklarace a typy proměnných starají. Jim to poměrně usnadňuje a urychluje vývoj, může tu ale docházet k častějším chybám a nedostatkům. Asi nejpopulárnějším jazykem pro tvorbu webových aplikací je PHP, které se díky své jednoduchosti stalo oblíbeným u řady začínajících programátorů. Jazyk jako takový prošel poměrně dlouhým vývojem, až se z něho pomalu stává plně objektově orientovaná platforma, na které lze s pomocí nespočtu frameworků stavět i rozsáhlejší

systémy. Musí se ale dodržovat poměrně striktní pravidla, aby nedocházelo k uhýbání od nastaveného standardu a nevracelo se k nálepce PHP jako jazyka pro rychlé „bastlení“ a mnohdy nesystémové programování.

Jazyk Perl je nejstarší z uvedené množiny. K dispozici je velké množství modulů, které z něho dělají velmi silný nástroj především pro skriptování a tvorbu programů na desktopu i serverech. Webové aplikace se v něm příliš nevyvíjejí, chybí nějaký dominantnější framework, který by vývoj co nejvíce ulehčoval. JavaScript je na serveru poměrně nový (díky technologiím jako Node.js) a přestože jistě najdeme mnoho užitečných knihoven, je to technologie zatím příliš ne moc rozvinutá a jistě má v sobě do budoucna velký potenciál. Ruby spolu s frameworkem Ruby on Rails je již zavedeným etalonem a jeho minimalizmus i zdánlivě striktní objektovost je programátorům velmi známa. Horší je to s obecnou podporou na serverech, hostingu, proto se dostává spíše do oblasti velkých firem a aplikací (Twitter) popř. jej tlačí jeho „evangelizátoři“.

Posledním diskutovaným je jazyk Python. Ačkoliv je dynamicky typovaný, jsou za běhu typy hlídány. Před spuštěním se kód navíc překládá do objektového (byte) kódu a výkonově kritické knihovny jsou napsány přímo v jazyce C (v němž je samotný jazyk implementován). Python byl navržen tak, aby se v něm daly kromě malých aplikací psát i rozsáhlé plnohodnotné aplikace (včetně desktopových a grafických). Stručností ho lze přirovnávat k Ruby, přesto obsahuje veškerou podporu pro psaní moderních velkých aplikací. Vysoký důraz je kladen na produktivitu práce programátora. V neposlední řadě je výhodou dostupnost velké řady knihoven (balíčku) a existence dominujícího frameworku Django [12].

Pro ukázkové řešení IS výrobce využijme tedy jazyka Python spolu s frameworkem Django. Volba pro toto řešení je především ve snadné realizaci pomocí dostupných komponent a komponenty Django Administrace, která umožňuje na základě datového modelu vytvořit kompletní formuláře pro editaci obsahu aplikace. Velmi je tak urychlen vývoj ukázkového IS. Oproti Javě bude aplikace v Pythonu navíc mnohem méně hardwarově náročná. Při vývoji kompilátor pracuje v módu JIT (Just In Time: kompilace kódu probíhá v reálném čase), takže změny jsou velmi rychle viditelné. Oproti jazyku PHP si navíc Python vynucuje určitou kulturu při psaní zdrojového kódu a není třeba se bát ani rozvoje aplikace, budou-li dodrženy všechny dobré zásady.

Z již uvedených výhod jazyka Python vyplývá, že by bylo vhodné jej použít i pro tvorbu serverové části projektu. V projektu se počítá s poměrně vysokou zátěží, přičemž rozpočet pro nákup hardwaru pochopitelně není neomezený. I proto vylučujeme jazyk Java z výběru. Pro použití jazyka Python opět hovoří existence Django frameworku, snadné tvorby ORM vrstvy (objektového modelu) a dostupnost mnoha již existujících komponent usnadňujících výsledný vývoj. Předpokládáme, že při realizaci projektu nebudeme znovu řešit již vyřešené problémy, nebudeme znovu vynalézat již objevené a realizované skutečnosti. Všechny použité součásti jsou většinou dlouhodobě odladěné a komunita okolo jazyka je používá. Kromě samotné komunity má Python podporu a používá se v mnoha velkých společnostech, globálně například Google, NASA, Rackspace, lokálně pak Seznam.cz.

3.5.3 Databáze

Informační systém výrobce i komunikační prostředník potřebují k ukládání dat nějaké databázové úložiště. Ukázkový IS předpokládá pouze omezenou funkcionalitu, reálná apli-

kace bude nejspíše postavena na komerčním řešení dodaném k vlastnímu systému (typicky Oracle). Nejblíže takové relační databázi je PostgreSQL, která splňuje veškeré nároky na moderní databázové prostředí, ačkoliv se nejedná o komerční produkt. V úvahu ještě připadá velmi populární MySQL či zcela volná, nedávno vzniklá, její varianta MariaDB (nepodléhá licencování vlastníka MySQL). Nejen nejasný osud tohoto typu databáze pod konsorciem Oraclu vede k použití ještě o něco vyspělejší PostgreSQL. V Česku má navíc tato databáze širokou podporu a je dlouhodobě prověřenou variantou při ne zcela pevné důvěře v MySQL. Pro IS výrobce tedy použijeme čistě open-source řešení v podobě databáze PostgreSQL.

Na ukládání dat do serverové části potřebujeme především rychlé úložiště schopné co nejkratší reakční doby. S výhodou tak můžeme využít některé z NoSQL databází (CouchDB, MongoDB, MemcachedDB). Tento druh databází ale většinou nezaručuje konzistenci dat a nemůžeme se spoléhat, že se uložená data neztratí. S výhodou je lze využít na uložení již vypočítaných dat a snížit tak nároky na databázové řešení. Pro urychlení načítání dat můžeme taktéž využít cache, rychlé key-value storage uložené v paměti (Memcached, Redis). Ta data drží pouze po určitou dobu (do jejich expirace) popř. po dobu svého běhu (restartováním služby či systému jsou data ztracena). Na pozadí takového řešení je však umístěno nějaké většinou relační databázové úložiště, ve kterém jsou data tzv. v bezpečí. Dojde-li k vyprázdnění cache, data jsou postupně načtena z databáze, čímž na ni vznikne krátkodobě vyšší výkonový nárok. Při realizaci komunikačního prostředníka bychom mohli s výhodou některé z takových řešení použít, ideálně automatizovaně, aby se aplikace nemusela starat, z jakého zdroje data načítá. V každém okamžiku by data v paměti měla být co nejvíce aktuální, proto použijeme jejich automatickou expiraci po určitém časovém okamžiku (např. řádově minutách). Cílem je obsloužit co nejvíce požadavků za co nejkratší dobu. V případě úprav v datech je možné cache ručně (aplikačně) smazat nebo vyčkat na její vypršení.

Na pozadí celého řešení by pak měla existovat stabilní databáze, ve které budou data vždy v pořádku k dispozici. Vyloučíme komerční řešení typu Oracle (přestože aplikace by neměla být přímo na zvolené databázi závislá) a zvažujme použití již zmiňovaných MySQL či PostgreSQL. Proti MySQL opět hovoří ne zcela jasné směřování i eventuální licenční politika, volné řešení v podobě MariaDB je stále v rozvoji (a velcí hráči používající dříve MySQL na něj přecházejí pomalu). S výhodami již uvedenými, navíc v rámci neroztříštěnosti technologií, použijeme relační databázi PostgreSQL i pro komponentu komunikačního prostředníka. Databáze možná nebude tak rychlá, ale zato data v ní budou spolehlivě uložena. Při vývoji budeme také optimalizovat pokládané dotazy, aby nedocházelo ke zbytečné zátěži databázového serveru. Pro urychlení odpovědí s výhodou můžeme použít cache systému, který nabízí přímo Django framework.

3.5.4 Operační systém

Serverové pozadí bude využívat operačního systému Linux, jak již bylo uvedeno ve vlastních požadavcích. Zbývá tedy vybrat distribuci alespoň pro prezentaci v této práci. Jinak by vlastní realizace neměla být na zvoleném řešení vůbec závislá a vše závisí na specialistech, kteří budou projekt skutečně realizovat. Vybírat budeme z distribucí postavených na principech Debian a RedHat. Rozdíly mezi těmito typy distribucí nejsou velké, každá obsahuje různé verze produktů (zpravidla se ale příliš neliší). Zásadním rozdílem je tak například uspořádání doporučené adresářové struktury a balíčkovacího systému instalujícího programy.

Nebudeme v tuto chvíli také zvažovat žádné komerční distribuce jako vlastní RedHat, SuSE či jiné. Kriteriem pro výběr je dlouhá podpora od komunity, četnost vydávání nových verzí i používanost v zavedených firmách. Vyloučíme také distribuce příliš náročné na údržbu či správu, resp. vyžadující hluboké znalosti dané problematiky (Gentoo). Uživatelsky přívětivé jsou distribuce CentOS, vlastní Debian či Ubuntu Server.

Pro tuto práci se budeme rozhodovat spíše mezi distribucemi založenými na Debianu, neboť autor práce s nimi má dlouhodobé zkušenosti. Znalost prostředí je pro stavbu serverového řešení velmi důležitá, a proto budeme tomuto faktoru přikládat nejvyšší váhu. V současné době je k dispozici Debian verze 7 (pod kódovým označením Wheezy) a Ubuntu Server verze 13.10 (pod kódovým označením Saucy). Pro výběr prvně jmenovaného hovoří především jeho stabilita a prověřenost. Verze jsou vydávány pouze ve dvouletých cyklech, často jsou v nich k nalezení již starší verze softwaru, zato však otestovaného a bez zásadních chyb. Proto si distribuce získala velkou oblibu v produkčním nasazení. Ubuntu vzniklo teprve nedávno (2004, viz [34]), poměrně dobře se dokázalo prosadit i na běžné desktopy uživatelů. Serverová varianta je očištěna od uživatelských aplikací (grafického prostředí) a jednou z jejích výhod je možnost poskytnutí podpory od tvůrce, společnosti Canonical Ltd. Ve dvouletých cyklech jsou vydávány tzv. LTS verze, jimž je podpora ve formě aktualizací poskytována následujících pět let od vydání (Long Term Support). Oproti Debianu bývají v Ubuntu novější verze softwaru, ačkoliv v případě LTS verzí je tato distribuce velmi podobná s mateřským Debianem.

Z pohledu konfigurace je Ubuntu více připraveno pro cílové užívání než Debian, který je nutno od základu celý nastavit. Z tohoto důvodu vybereme pro tuto práci použití distribuce Ubuntu Server, a to v poslední LTS verzi 12.04 (verze označuje měsíc a rok vydání).

3.6 Bezpečnost

Jedním z velkých témat této práce je oblast bezpečnosti. Podle požadavků (viz 2.5.2) i dle zadání práce je třeba věnovat této oblasti zásadní pozornost. Podívejme se nyní podrobně na všechny součásti, kterých se tato otázka bude týkat.

Jedním z probíraných témat je bezpečnost samotných tagů produktů. Věnovat se musíme také komunikaci klienta s komunikačním prostředníkem, stejně jako komunikaci prostředníka s IS výrobce [13]. Vlastní část ponechme také systémové bezpečnosti, diskuzi se zabezpečením přístupu k datům, jak na úrovni administrační aplikace, tak na úrovni správy samotných serverů. Systém má také dovolovat používání privilegovanými osobami, kterým mají být poskytována rozšiřující data.

Otázku bezpečnosti řešíme především proto, abychom mohli systém považovat za důvěryhodný. Jak již bylo uvedeno v části 3.1, na důvěře v komunikačního prostředníka jako autority stavíme další komponenty systému a jejich propojení. Abychom celý systém považovali za bezpečný, musíme splnit základní podmínky bezpečnosti: důvěryhodnost, nepopiratelnost, nezaměnitelnost, autenticitu, dostupnost a auditovatelnost.

Důvěryhodný bude systém v případě, že bude zabezpečeno, aby se přenášená data nedostala do neoprávněných rukou a zároveň nebyla nikým neoprávněně změněna. To platí nejen pro přenos dat mezi komponentami systému, ale i pro data uložená v tagu produktu.

Nepopiratelný přenos dat v systému bude tehdy, jestliže zabezpečíme, aby obě strany komunikace mohly navzájem prokázat, že data pocházejí od nich, tj. aby nikdo nemohl tento přenos zpochybnit, popřít. Produkt nepopiratelného původu tak bude až po ověření jeho původu u výrobce pomocí komunikačního prostředníka a celá tato transakce nesmí být nikým, ničím zpochybněna.

Nezaměnitelná data se v systému přenášejí tehdy, když je komunikace zabezpečena proti jejich modifikaci, popř. jsou stanoveny takové mechanizmy, které změnu odhalí. Cílem této podmínky je tedy informace v tagu produktu zabezpečit tak, abychom poznali, že byla změněna, stejně jako použít takového přenosového kanálu, aby nebylo možné data změnit, popř. tuto změnu detekovat.

Autenticita dat v systému zaručuje, že data skutečně pochází z uváděného zdroje. Řešení tak musí zabezpečit, aby komunikace probíhala skutečně s tím, s kým se předpokládá, že má probíhat.

Dostupností systému z hlediska bezpečnosti rozumíme přístupnost dat pouze autentizovanému a autorizovanému účastníkovi procesu. Musíme tak zajistit, aby se do systému nedostal někdo nepovolaný a nemohl získat nějaká data či běh systému narušit.

Auditovatelný systém umožňuje rozpoznávání, zaznamenávání, ukládání a analýzu informací o událostech, které v něm proběhly a mají význam pro bezpečnost. V rámci toho bychom tak měli zajistit ukládání dat, která mohou případně pomoci k odhalení narušení některé z oblastí bezpečnosti.

3.6.1 Bezpečnost tagů

První částí bezpečnostní oblasti, kterou se musíme zabývat, jsou vlastní tagy produktů, v nichž jsou uloženy informace o daném výrobku spolu s identifikací výrobce. Potřebujeme tato data zabezpečit proti modifikaci a zároveň umět rozhodnout, zda je zapsal skutečně uvedený výrobce.

3.6.1.1 Zabezpečení dat v tagu

Nejníže postaveno celému systému je čtení tagů produktů, v nichž jsou uloženy informace o daném výrobku. Součástí této práce není sestavit zařízení, které bude data zapisovat, ani navrhnout způsob dat v tagu uložených. Tuto skutečnost mají na starosti vývojáři hardwaru, kteří podle technologických parametrů tagů stanoví, jaká data bude tag obsahovat a v jaké formě. Úkolem této práce je pak po vzájemné konzultaci navrhnout, jak ověřit, že zapsaná data nebyla nikým ani ničím modifikována.

Poohlédneme-li se trochu do historie, způsob, jakým ověřit platnost dat či chybu v nich, je starý téměř jako historie přenosu informací samotných. Na nejnižších úrovních elektrické komunikace počítáme paritní bity, kontrolní součty (CRC) či přidáváme další různá data proto, abychom ověřili, že přenesená informace nebyla poškozena. Z pokročilejších technik jmenujme různé algoritmy založené na početní operaci modulo, Luhnův algoritmus a různé kryptografické hashovací funkce (MD5, SHA). Všechny jsou založeny na analýze předložených dat a vytvoření jejich otisku, ať už unikátního či nikoliv. Algoritmus musí při své

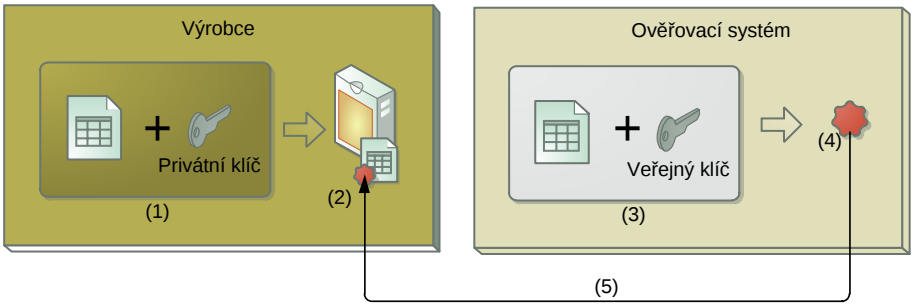
reprodukci vrátit na stejná data stejný výsledek, čehož se využije právě při kontrole, zda nedošlo k modifikaci vlastních dat. Přičemž záleží na použitém algoritmu, jak spolehlivě tuto skutečnost odhalí.

Nyní tedy víme, že spolu s daty bude do tagu uložen i jejich otisk, aby bylo jasné, že nedošlo k jejich modifikaci. Jak ale poznáme, že data nebyla pozměněna i s vlastním otiskem? Zcela jednoduše se nabízí použití určitého šifrovacího algoritmu, který data převede do nečitelné podoby. Bude-li se jednat o známý algoritmus s veřejně dostupnými vstupními daty (klíči), nemusíme data šifrovat vůbec, neboť jejich dešifrování je otázkou pouze znalosti daného algoritmu spolu s daty pro rozklíčování. Vytvořit vlastní algoritmus může být sice výhodné, ale je jenom otázkou času, kdy potenciální útočník algoritmus prolomí.

Celou touto otázkou se zabývá moderní kryptografie a přináší několik různých technik, jak data zabezpečit či kontrolovat jejich platnost. Snahou není data uložit v nečitelné podobě, nepotřebujeme je tedy šifrovat. Postačí, pokud budeme mít možnost nepopiratelně ověřit, že ten, kdo je do tagu zapsal, je tentýž subjekt, který je v zapsaných datech uveden. S výhodou tak využijme jedné z moderních metod, a to principu digitálního podpisu [16] [49].

Princip digitálního podpisu

Princip zápisu dat do tagu a jejich ověření uvedeme na typickém příkladu uživatelů Alice a Bob [53], přičemž Alice v tomto případě reprezentuje výrobce, Bob je pak systém, který ověřuje, že data do tagu skutečně zapsal výrobce a ne někdo jiný. Útočník, Eva, se pak pokusí data změnit.



Obrázek 3.15: Princip ověření dat v tagu produktu

Proces podepsání dat a jejich verifikace je znázorněn na obr. 3.15. Výrobce zapisovaná data podepíše (1) svým privátním klíčem (private key). Vytvoří tak otisk dat spolu s informací, kdo data podepsal. Otisk je připojen k datům o produktu a vložen do tagu (2), který se umístí na produkt. Ověřovací systém pak čte data z tagu a s pomocí dostupného veřejného klíče (public key) vytvoří (3) svou vlastní verzi otisku (4). Ten následně porovná s otiskem vyčteným z tagu (5). V případě, že tyto otisky jsou shodné, věříme, že data skutečně zapsal a podepsal výrobce.

Na řadu se dostává útočník Eva, který data pozměnil. V tom případě otisky nesedí, výsledek ověření selhává a došlo tedy skutečně ke snaze koncovému uživateli podvrhnout zasílaná data. V dalším případě Eva data pozměnila a zároveň je znovu podepsala. Jelikož nemá k dispozici privátní klíč výrobce, použila svůj privátní klíč. Ověřovací systém ale postupuje

opět na základě veřejného klíče výrobce (který má z důvěryhodného zdroje), vytvořený otisk tedy opět nesouhlasí.

Digitální podpis založený na asymetrické kryptografii (tedy znalosti privátního a veřejného klíče) je výkonným nástrojem moderní informační doby. V závislosti na délce, složitosti klíče, trvá nejen samotné podepisování či šifrování, ale zároveň i jeho prolomení. Ačkoliv je technologie založena na poměrně jednoduchých principech součinu prvočísel, prozatím neexistuje způsob, který by dokázal tyto šifrovací algoritmy v rozumně krátkém čase prolomit. V dalších částech se tak budeme zabývat ještě otázkou, aby Eva nepodstrčila ověřovacímu systému svůj veřejný klíč a ten se tedy nedomníval, že data jsou platně podepsána.

3.6.1.2 Ověření platnosti dat v tagu

V předchozí části byl stanoven nástroj digitálního podpisu jako způsob, jímž jsou verifikována data uložená v tagu produktu. Ověřovací systém je tedy umístěn v mobilní aplikaci. Na vyžádání uživatele bude provedeno ověření uložených dat k tomu určenou komponentou. Ta bude potřebovat mít k dispozici veřejný klíč daného výrobce.

Pro jazyk Java, v němž bude mobilní klient naprogramován, existují již hotové implementace potřebných algoritmů, bude stačit je jenom použít. Budeme však muset vytvořit způsob, jakým bude aplikaci předáván veřejný klíč výrobce a jak ta s ním bude nakládat. Pro tento případ již budeme potřebovat komunikačního prostředníka, který potřebná data poskytne. V něho máme důvěru, že má data z důvěryhodného zdroje, nebyla nikým pozměněna a komunikační kanál je dostatečně chráněný.

3.6.2 Ověření produktu u výrobce

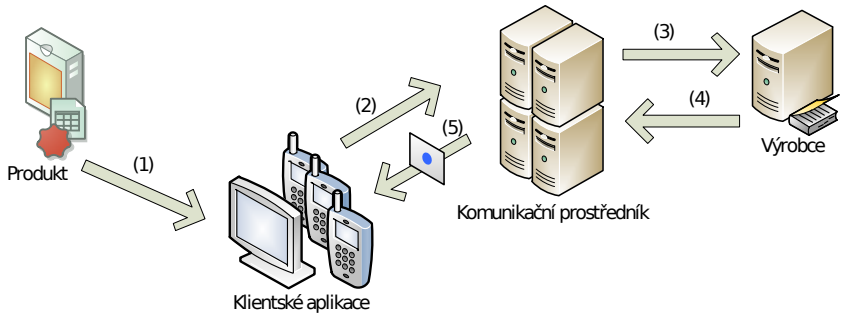
Druhým stupněm ověření původu produktu je nalezení jeho matričního záznamu. Teprve poté budeme moci prohlásit, že produkt je uvedeného původu a nebyl zaměněn s jiným či nikým zfalšován. Předpokladem pro celou akci je vedení záznamů o vyrobeném zboží v informačním systému výrobce. V této části se tak musíme zabývat otázkou zabezpečení komunikační cesty od klienta k výrobci a zaručit, že klient dostal správná data.

Již při zkoumání existujících řešení (viz 2.3) jsme narazili na službu ONS. Ta poskytuje podobnou funkcionalitu, jakou požadujeme. Jejím úkolem je směřování dotazů klienta na službu výrobce. Předpokladem je použití EPC kódu pro určení cílového výrobce, přičemž klient následně komunikuje přímo se systémem výrobce. Tato cesta je z bližšího pohledu poměrně nebezpečná, neboť klient nemá zaručeno, že byl jeho požadavek nasměrován na správnou adresu. Tímto způsobem tak nelze jednoznačně rozhodnout, jestli produkt pochází od určitého výrobce.

Při definici komunikačního prostředníka v části 3.1 byla zmíněna jeho role jakožto důvěryhodné autority. Z tohoto předpokladu budeme nyní dále vycházet. Aby měl klient jistotu, že přijímá správná data, musí existovat právě důvěryhodný prostředník, který je poskytuje. Klient v tom případě nepotřebuje znát, odkud prostředník data získal, bere je jako data nepopiratelného původu (za předpokladu bezpečné přenosové cesty).

3.6.2.1 Proces ověření u výrobce

V předchozí části jsme se zabývali bezpečnostním opatřením dat v tagu proto, abychom mohli prokázat, že produkt pochází od daného výrobce, tedy z důvěrného zdroje. Bezpečnostní prvky na cestě informace, kterými se budeme nadále zabývat, by měly zabránit jakémukoliv pokusu o podvržení dat či jejich modifikaci. Zároveň by uživatel neměl získat data, na která nemá oprávnění, a pokus toto oprávnění získat by měl být včas zastaven, tj. budeme se zabývat autentizací a autorizací uživatele.



Obrázek 3.16: Průběh ověření produktu u výrobce

Vlastní ověření existence produktu u výrobce a načtení dalších dat pak zobrazuje obr. 3.16. Data z tagu produktu jsou nejdříve načtena (1) klientskou aplikací (v tomto případě softwarem v mobilním zařízení), včetně jejich elektronického podpisu. Z vyčtených dat je vyjmut identifikátor výrobce a produktu (může se jedna o unikátní identifikátor tagu) a odeslán (2) na komunikačního prostředníka. Ten načte informace o daném výrobci a dotáže se (3) jeho informačního systému na daný produkt. Výrobce odpoví (4) zprávou, kterou komunikační prostředník zpracuje, a výsledek ověření zašle zpět (5) klientské aplikaci. Spolu se zasílanými daty může komunikační prostředník odeslat i certifikát výrobce (viz dále) v případě, že o to klient zažádal. Uvedený postup nezohledňuje problematické či chybové stavy, stejně jako vlastní způsob komunikace.

3.6.2.2 Zabezpečení kanálu komunikačního prostředníka s výrobcem

Aby bylo možné uživateli poskytnout data z bezpečného zdroje, je nutné zabezpečit jejich cestu od výrobce ke komunikačnímu prostředníkovi. Tuto cestu lze vytvořit pomocí ověřených postupů a principů, které se při propojování dvou privátních sítí používají. Při tom musíme brát i v potaz, aby komunikace nebyla příliš pomalá, popř. aby rychlost moc neovlivňovaly použité bezpečnostní mechanizmy. S tím také souvisí, na jaké síťové vrstvě je dané zabezpečení umístěno.

Obecně tedy řešíme problém spojení dvou sítí pomocí **VPN** (Virtual Private Network). Tato technologie může mít několik podob, přičemž často může být tento pojem zaměňován s aplikaci OpenVPN, která způsobem klient-server dokáže připojit klienta do vlastní privátní sítě. Komunikace je šifrována pomocí knihovny OpenSSL a vytváří další síťové rozhraní (na síťové nebo spojové vrstvě ISO/OSI modelu [35]), které používá k vlastní komunikaci. Abychom tento způsob mohli použít, musela by se infrastruktura komunikačního prostředníka připojovat k různým sítím různých výrobců, anebo by se oni připojovali k serveru umístěném

v části komunikačního prostředníka. Výhodou je celková bezproblémovost provozu i na jiných operačních systémech než Linux, nevýhodou řešení může být zpomalení při šifrování spojení.

Další formou VPN je technologie **IPSec** [54]. V tomto případě se nejedná o koncepci klient-sever, ale označujeme komunikující sítě jako strany – levou a pravou (left, right). Spojení je zabezpečeno stejně jako u OpenVPN pomocí asymetrické kryptografie, k provozu nepotřebuje vytvářet další síťové rozhraní. Po konfiguraci na obou stranách navíc dochází k automatické inicializaci spojení vyvoláním z jakékoliv strany. Další výhodou je nativní podpora v operačních systémech (včetně Windows), netřeba instalovat další podpůrný software. Nevýhodou může být trochu náročnější konfigurace a nutná znalost dalších síťových pravidel (směrování apod.) pro náročnější požadavky.

Výběr konkrétní technologie bude záležet na dohodě s výrobci, k jejichž systémům se bude komunikační prostředník připojovat. Jako preferované řešení je vytvoření IPSec tunelů díky podpoře v operačním systému a jednoduchosti navazování spojení.

Kromě vlastního komunikačního kanálu může systém výrobce vyžadovat podepisování či šifrování vlastních zpráv. To přináší další míru zabezpečení popř. alternativu k vytvoření bezpečného tunelu, klade však větší nároky na výkon i aplikaci komunikačního prostředníka samotnou. Ta by tak musela mít uložené potřebné certifikáty a k výrobcům je načítat, provádět podepisování, šifrování, ověřování zpráv. Při realizaci projektu nemůžeme ani tuto variantu vyloučit. V prototypu IS výrobce, který je součástí této práce, budeme počítat s jednoduchým rozhraním, přičemž komunikační prostředník bude zohledňovat, že každý IS výrobce může mít trochu jiné rozhraní, služby, a jako konzument by se tedy měl přizpůsobit.

Dále budeme předpokládat že komunikační kanál mezi výrobcem a komunikačním prostředníkem bude vytvořen pouze autorizovanou osobou. Běžnému uživateli bude tato funkcionalita skryta a ke kanálu by se tak mohl dostat pouze ten, kdo by se napojil na komunikační linku či měl přístup k síťovým komponentám na cestě. Obecné nepovědomí o existenci spojení nesmí snížit nároky na jeho bezpečnost, můžeme však počítat s menší pravděpodobností útoku, než když se jedná o veřejně dostupné rozhraní.

3.6.2.3 Zabezpečení komunikace klienta s komunikačním prostředníkem

Velmi důležité je zabezpečení rozhraní komunikačního prostředníka s klientem. Tato služba bude veřejně dostupná a kdokoliv tedy může k rozhraní přistoupit, pakliže bude znát jeho podobu. Musíme tak předpokládat, že budou existovat i přístupy s cílem neoprávněně získat některá data, popř. podvrhnout je klientovi tak, aby se domníval, že jsou platná.

Služba bude navržena jako webová s přístupem pomocí technologie REST. To umožňuje použít bezpečnostní mechanizmy dostupné pro komunikaci pomocí HTTP, autentizační hlavičky i podporu šifrování spojení vestavěnou ve webovém serveru. Zároveň můžeme používat moduly, které zabrání eventuálnímu útoku (omezování přístupu z IP adres) popř. budou řídit počet požadavků za danou časovou jednotku.

Z hlediska možných útoků počítejme s odchycením komunikace a získáním přístupu k zasílaným datům, podvržením komunikace, kdy se útočník bude snažit klientovi vnutit svá vlastní data, a v neposlední řadě i s možným útokem na server samotný, ať už s cílem získání dat či zahlcením požadavky (tzv. DoS popř. DDoS útok [64]).

SSL komunikace

Základním bezpečnostním prvkem pro veřejné rozhraní by mělo být použití protokolu HTTPS, který se používá pro zabezpečená spojení klientů se servery napříč celým internetem (jedná se o rozšíření standardního protokolu HTTP o SSL – Secure Sockets Layer). Odchycená šifrovaná komunikace může být sice rozkódována, ale podvrhnout ji je mnohem náročnější. Principiálně se jedná o systém asymetrické kryptografie a výměny klíčů, podobný jako u digitálního podpisu (viz část 3.6.1.1). Klient by měl kontrolovat důvěryhodnost serverového certifikátu (s veřejným klíčem) a v případě podezření informovat uživatele nebo zamítnout spojení. Pokud bude i realizovaný klient provádět zmíněnou kontrolu, můžeme s velkou pravděpodobností vyloučit možný útok podvržením dat.

Odposlechnutá data je možné rozklíčovat, s tím ale musíme počítat, pokud dáváme k dispozici veřejně dostupné rozhraní. Otázkou zůstává, jaký smysl mají pro útočníka získaná data, když je může v podstatě legálním způsobem dostat při vytvoření vlastního klienta. Navíc takto nelze získat veškerá data, která systém může poskytnout, nehledě na to, že část z nich lze získat z veřejně dostupných zdrojů (např. údaje o firmách). Zajímavá by mohla být data, která dostávají privilegovaní uživatelé (např. kontroloři). Jejich klient bude obsahovat vlastní autentizační údaje, na základě nichž budou data poskytnuta. Navíc provoz těchto klientů by měl být monitorován a v případě podezřelého chování proveden audit zjištěné situace.

SSL komunikace bude samozřejmě zpomalovat celou komunikaci klienta s komunikačním prostředníkem. Zabezpečené spojení se vytváří pouze mezi klientem a vstupním bodem infrastruktury komunikačního prostředníka, dále, interně, je komunikace již nezabezpečena. Počítá se s tím, že na vstupní bod bude použit software, který bude splňovat náročné požadavky na rychlou odezvu a zároveň nebude příliš zatěžovat server obsluhou šifrované komunikace. Takové řešení je zcela běžně používané, kdekoliv se přistupuje k nějakým datům, ke kterým nemá mít přístup kdokoliv, ale zase nelze klienta příliš omezovat při dotazování se na server. Můžeme také uvedené rozhraní použít pouze pro přístup privilegovaných klientů, pro všechny ostatní používat běžné, nezabezpečené, spojení. Prostým odchycením nezabezpečené komunikace by pak bylo možné vytvořit vlastního klienta, což obecně nejspíše nebude příliš žádoucí.

Autentizace

V rámci požadavků bylo rozhodnuto, že každý klient se bude v aplikaci autentizovat (viz požadavky v části 2.4). To platí tedy i pro neprivilegovaného klienta, který nepožaduje více než jen základní sadu dat. Protože služba je tvořena jako webová, můžeme s výhodou použít zavedené mechanizmy HTTP Autentizace [37]: Basic a Digest. Oba používají HTTP hlaviček pro odesílání autentizačních údajů, vše potřebné obstará webový server a aplikace se nemusí o přihlášení uživatele vůbec zajímat. Pro autorizaci, tj. zjištění o jakého uživatele se jedná a jaká tedy bude mít práva, tyto informace bude muset pochopitelně znát.

První možný způsob, Basic, funguje na principu zaslání uživatelského jména a hesla v téměř nezabezpečené podobě (kódování pomocí technologie Base64 nelze považovat za bezpečný přenos, jde-li jej bez problémů rozkódovat). Server přijímá zaslané údaje, rozkóduje je a prostým ověřením vůči tabulce jmen a hesel povolí či zamítne přístup (typicky

požádá o opětovné zadání údajů – záleží na nastavení). Tabulka pak je ve formě textového souboru, databáze, či jiného prostředku pro ověřování uživatelů (pam.d, LDAP, Active Directory apod.).

Druhý možný způsob, Digest, byl navržen především kvůli nebezpečnosti prvního přístupu. Cílem je přinést o něco bezpečnější způsob ověření bez odesílání hesla v „čitelné" podobě. Pro celý princip je použito hashovacích funkcí (typicky MD5), přičemž server nepotřebuje znát heslo v čitelné podobě a od klienta se odesílá pouze hash (otisk) hesla rozšířeného o další údaje (aktuální URL, serverem zaslaný časově omezený unikátní řetězec atd.). Server si vygeneruje vlastní hash a porovnává ho s přijatým. Díky tomu tak při odposlechnutí komunikace může sice dojít k ukradení aktuálního přihlášení stejně jako v prvním případě. Přihlášení může být kromě jiného vázáno na IP adresu či časově a ani se znalostí získaných dat není možné jej po ztrátě obnovit.

Z nabízených metod si pro realizaci vybereme jistě tu bezpečnější, Digest autentizaci. Při rozhodování o použitých metodách se nabízí také vytvoření vlastního autentizačního mechanizmu například za pomoci nějakého unikátního řetězce (tokenu) ve stylu Digest autentizace. Budeme-li předpokládat, že takové proprietární řešení může být mnohem více zranitelné než již existující a ověřená metoda, bude vhodnější použít již osvědčených algoritmů.

Autorizace

Ve druhé fázi, poté co se uživatel přihlásí, budeme potřebovat jeho údaje (resp. přihlašovací jméno) k tomu, abychom autorizovali jeho akce. Každý uživatel je řízen právy skupiny, do které je přiřazen. Práva mohou být stupňována, tj. skupina s vyššími právy může disponovat právy nižší skupiny, ale může mít také některá práva zcela odlišná.

Pro komunikaci klienta se službou komunikačního prostředníka budeme používat rozdělení na dvě základní skupiny uživatelů: běžní uživatelé a privilegovaní uživatelé.

Běžný uživatel disponuje nejnižšími právy, která může po přihlášení získat. Pod tímto uživatelem přistupují do aplikace všichni klienti, kteří nepotřebují nějaká zvláštní oprávnění. Poskytnuto jim je čtení dat pomocí služby, ať už dat uložených přímo v aplikaci komunikačního prostředníka nebo získané čtením ze systému výrobce. Při získávání dat z jeho IS je požadována základní sada dat, není přístup k žádným dalším položkám. Je-li rozhraní výrobce navrženo tak, že poskytuje všechna data, prostředník pak odstraňuje data, která klient nemá dostat.

Privilegovaný uživatel obsahuje kromě práv běžného uživatele přístup k dalším položkám, především k detailním informacím o produktech. Ty jsou získávány z IS výrobce a na základě jeho rozhraní je požadována rozšířená množina dat, anebo jsou již získaná data předána klientovi bez omezení. Privilegovaný uživatel se přihlašuje speciálním klientem popř. jeho přihlášení v klientovi je speciálně zabezpečeno. Jeho akce mohou být na serveru zaznamenány a monitorovány.

Při ověřování informací o produktu vycházíme z předpokladu, že klient privilegovaného uživatele dostatečně autentizoval a komunikační prostředník tak může věřit, že zaslané přihlašovací údaje autorizují povolanou osobu. Pokud by se k přihlašovacím údajům dostal

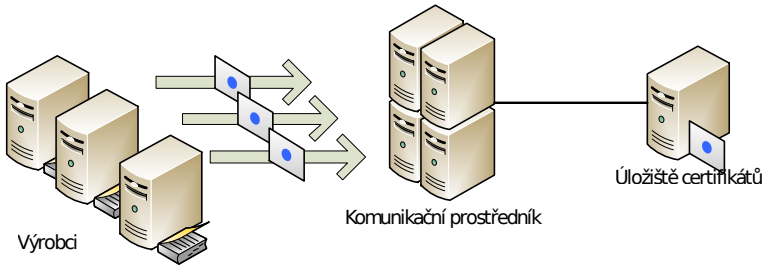
útočník, může je zneužít k vytvoření vlastního klienta a neoprávněnému získání dalších dat. Proto jsou akce privilegovaného klienta monitorovány a v případě podezřelého chování je nastalá situace blíže zkoumána (uživateli je např. zakázán přístup, prověřeno, že údaje používá skutečně ten, kdo je získal apod.).

3.6.3 Certifikát výrobce

V části 3.6.1.1 jsme se zabývali uložením dat v tagu produktu a jejich podpisem. Při tom bylo zmíněno, že ověřování se provádí pomocí veřejného klíče výrobce, který klientská mobilní aplikace získá od komunikačního prostředníka. Pakliže nedošlo k modifikaci komunikace (viz část o zabezpečení kanálu, 3.6.2.3), můžeme s důvěrou prohlásit, že máme k dispozici data potřebná pro ověření podpisu.

Certifikát má podobu souboru, který může být uložen v několika různých formátech (jak binárně tak textově). Vlastníkem certifikátu je obvykle společnost či osoba (popř. aplikace prokazující se certifikátem společnosti), obsahem pak její údaje (název, kontaktní údaje apod.), údaje o vydavateli certifikátu a veřejná část klíče sloužící právě pro podepisování či šifrování komunikace. Vydavatelem certifikátu může být pouze pověřená instituce, firma, tzv. **certifikační autorita**. Ta získala pravomoc podepisovat a vydávat certifikáty, přičemž její podpis je označen jako důvěryhodný, neboť ona sama vlastní certifikát podepsaný nadřazenou autoritou. Jedná se tedy o stromovou strukturu, kdy na počátku jsou tzv. kořenové certifikáty, které jsou jako důvěryhodné obsahem již samotných operačních systémů (internetových prohlížečů, e-mailových klientů atd.). **Podepsaný certifikát** využívá princip digitálního podpisu, kdy data uložená v certifikátu jsou označena jako pravá tím, že se za něj zaručila vyšší instance (certifikační autorita). Na základě tohoto principu stavíme moderní zabezpečovací prvky nejrůznějších systémů.

Každý podepsaný certifikát obsahující veřejný klíč může být veřejně vystaven, aby si každý mohl ověřit, že podpis dat zaslaný spolu s nimi byl skutečně podepsán tím, kdo je uveden v certifikátu (kde je také uložen veřejný klíč sloužící k onomu ověření). Pravost certifikátu samotného, jak již bylo zmíněno, pak deklaruje jeho podpis od certifikační autority. Skutečnosti, že certifikát je komukoliv dostupný, využijeme i při ověřování platnosti dat v tagu produktu.



Obrázek 3.17: Vložení certifikátů výrobce do komunikačního prostředníka

Klientská mobilní aplikace, aby mohla platnost dat ověřit, potřebuje získat certifikát výrobce od komunikačního prostředníka. Ten obsahuje úložiště certifikátů právě pro jejich snadnou distribuci, zároveň hlídá jejich platnost, vypršení či zneplatnění vydané certifikační

autoritou (tzv. CRL – Certificate Revocation List). Vložení certifikátů a strukturu zobrazuje nákres 3.17. Certifikát se pak posílá spolu s daty o firmě/produktu, pokud o to aplikace zažádá (viz obr. 3.16). Na základě získaného certifikátu aplikace provádí ověření dat zapsaných tagu a ukládá si certifikát pro pozdější použití. Kdykoliv pak může zažádat o jeho opětovné stažení či ověření (přes jeho otisk). Certifikát je tedy podkladem pro ověření nezaměnitelnosti dat v tagu, ověřovacích dat uváděných dříve.

3.6.4 Serverová platforma

Součástí realizace, aby byl systém provozuschopný, bude také možná serverová platforma. V té souvislosti se budeme zabývat bezpečnostními opatřeními této oblasti. Protože na toto téma bylo napsáno již mnoho článků a publikací (např. [60] [33]), v této části uvedeme pouze výběr těch nejdůležitějších pravidel, která v systému vyžadujeme dodržovat. Obecně bychom měli pro tuto část dodržet všechna kritéria bezpečnosti fyzické, procesní i aplikační.

Neoprávněný přístup – výběr pravidel

- Vlastní umístění serverů by mělo být zabezpečeno proti přístupu neoprávněné osoby, tedy servery umístěné v uzamčeném racku přístupné pouze držitelům přístupového bezpečnostního prvku (klíče popř. elektronického zabezpečení formou např. přístupové karty či RFID čipu).
- Hesla uživatelů jsou velmi silná (minimálně deset náhodných znaků, kromě alfanumerických i interpunkce a další), v ideálním případě je přihlášení pomocí hesel zcela zakázáno. Systém může být také nastaven tak, aby po určitém počtu neúspěšných pokusů účet zablokoval.
- Přímé přihlášení uživatele s nejvyššími právy (v Linuxu „root“) je zapovězeno. Nesmí ani existovat další účet s takovými právy.
- Vzdálené přihlašování se provádí pouze pomocí silných SSH klíčů, přičemž jejich použití je podmíněno užíváním hesla pro přístup k nim a ideálně ukládání na zabezpečené médium (kryptované disky).
- Vzdálený přístup je dále omezen na určité IP adresy, popř. je pro připojení použito zabezpečení formou OpenVPN řešení. Není možné se pro správu serverů přihlašovat z veřejně dostupného místa.
- Přístup k počítači, ze kterého přihlášení probíhá, by měl být taktéž dostatečně zabezpečen (vlastní silné heslo uživatele).
- Každé přihlášení by mělo být zaznamenáno, stejně jako každý důležitější zásah do systému.

Databáze – výběr pravidel

- Databáze obsahuje více uživatelů, kteří se řídí nastavenými právy. Každý uživatel má přístup pouze do té oblasti, do níž potřebuje zasahovat.

- Vlastníkem objektů v databázi je jiný uživatel než ten, který používá aplikace pro přístup. Aplikace tak sama nemůže ovlivnit strukturu databáze.
- Databázový server naslouchá pouze na síťovém rozhraní, pomocí něhož k němu přistupuje aplikace. Jakákoliv propagace spojení mimo infrastrukturu je zapovězena.
- Administrace serveru by měla být vyřešena tak, aby administrátor mohl zasahovat do běhu serveru, ale nemohl ovlivňovat uložená data či je číst. Lze toho docílit nastavením autentizace (modulu pam.d v Linuxu) a přístupových práv v systému.

Infrastruktura tvoří vlastní lokální síť, která je uzavřena a zvenčí přístupná pouze přes vstupní brány. Navenek musí být dostupné pouze porty a rozhraní, které jsou využívány, tj. přístup pro klientské aplikace a nutnou správu vnitřní sítě.

3.6.5 Klientská aplikace

Protože v rámci toho projektu navrhujeme i mobilního klienta, který se připojuje na komponentu komunikačního prostředníka, rozeberme krátce bezpečnostní opatření, která by měla vývoj a provoz této části provázet. Vlastní bezpečnost komunikace již byla popsána v kapitole 3.6.2.3, zaměřme se na další nutné prvky pro maximální ochranu dat i uživatele. Měli bychom také rozhodnout, co se má dít, pokud je předpokládaná bezpečnost narušena.

Rozdělení aplikace

Jak již bylo zmíněno, mobilní klient má cílit především na dvě skupiny uživatelů. V zásadě bychom tak mohli vytvořit dvě verze aplikace, přičemž jednoduchá verze obsahuje pouze základní funkcionalitu pro běžného uživatele. I kdyby byl proveden podrobný rozbor aplikace (zdrojových kódů či tzv. reverse engineering, který dokáže rozebrat již hotovou aplikaci na její součásti), nebylo by možné získat detailní informace o možné zranitelnosti řešení. Neveřejná aplikace by pak obsahovala podrobnější funkcionalitu pro privilegované uživatele (včetně jejich přihlášení a dalších možností). Tuto verzi budou mít pak dostupnou pouze uživatelé, kteří ji potřebují ke své práci. Nevýhodou celého tohoto řešení je, že velká část obou aplikací je společná.

Můžeme tak zvážit variantu, kdy by i běžnému uživateli byla dostupná plná verze aplikace, ale některé její části budou zcela skryty tak, že o nich uživatel nebude mít ani ponětí. To ale přináší určité bezpečnostní riziko právě z pohledu neoprávněného rozboru aplikace. Ideálním řešením je postavit aplikaci tak, aby jednotlivé součásti byly tvořeny moduly, pomocí nichž se sestaví výsledná aplikace.

Kontrola dodržení bezpečnosti

Bude-li klient používat všechny stanovené mechanizmy, měl by zároveň kontrolovat, zda nedochází k jejich porušení. Používá-li například SSL komunikaci, měl by kontrolovat, zda použitý certifikát je skutečně podepsán důvěryhodnou autoritou, tj. nebyl například nikým podvržen. Nutnou součástí aplikace musí být otisk tohoto certifikátu s informací o platnosti. Pokud by vypršel, je nutno vydat aktualizovanou verzi aplikace s otiskem nového certifikátu.

Při kontrole digitálního podpisu v tagu produktu se taktéž využívá certifikátů. Aplikace by tak měla zároveň kontrolovat, zda je tento certifikát stále platný a v pravidelných intervalech se o tom ujišťovat i u komunikačního prostředníka. Již stažené certifikáty musejí být také bezpečně uloženy.

Datové úložiště

Aplikace disponuje vlastním datovým úložištěm, do něhož může zasahovat pouze ona sama. To je v OS Android vyřešeno přímo prostřednictvím databáze SQLite. V návrhu můžeme počítat, že nejvíce dat zabere uložení certifikátů výrobců. Pro ty bychom měli nalézt jiný způsob uložení, ideálně na paměťovém médiu zařízení. V tu chvíli mohou mít k datům přístup i sami uživatelé a bude nutné kontrolovat, zda soubor nebyl nijak modifikován. V první fázi tak můžeme použít spíše bezpečnějšího uložení certifikátu přímo v databázi.

Při realizaci musíme počítat dále s tím, že úložiště může uživatel snadno smazat (existuje na to přímo funkce v OS). Aplikace musí počítat s tím, že žádná data nemá a bude si je muset postupně ukládat. Není možné, aby se smazáním úložiště uživatel dostal do nějaké části aplikace, do níž nemá přístup.

Úložiště slouží pouze spíše jako cache pro to, aby se omezila zátěž na komunikačního prostředníka. Nesmí být do něj ukládány žádné citlivé údaje, přihlášení apod. I v privilegované verzi aplikace musí být přihlášení uloženo pouze v paměti a pokud nebylo delší dobu použito, musí být i odtamtud odstraněno.

Podpis aplikace

Součástí vytváření distribučního balíku aplikace je její podepisování vytvořeným klíčem. To zajišťuje fakt, že by obsah aplikace neměl nikdo modifikovat a pokud by se tak stalo, neměl by operační systém dovolit její spuštění. Klíče pro podepsání pak musí být bezpečně uloženy stejně jako heslo pro jejich vytvoření.

Podepsaná aplikace obsahuje informace o jejím autorovi, tato část by měla být důsledně vyplněna. Podpis také zaručuje, že požadovaná povolení, která aplikace potřebuje k chodu, nejsou přehnaná a jsou skutečně použita ke svému účelu.

Narušení bezpečnosti

Pokud aplikace zjistí, že některý z bezpečnostních mechanizmů byl narušen, měla by o tom vhodnou formou informovat uživatele. Typicky se může stát, že vypršela platnost certifikátu, na jehož základě byla ověřována data zapsaná v tagu. V tom případě je vznesen požadavek na komunikačního prostředníka, který by měl poskytnout nový platný certifikát. V případě, že se tak nestane, aplikace informuje uživatele varovnou hláškou, že nemůže spolehlivě data v tagu ověřovat.

Další možný případ může nastat, pokud komunikační prostředník odmítne přihlášení uživatele (zablokování přístupu). Aplikace opět zobrazí varovnou hlášku, přičemž se tím ověření produktu u výrobce stává nefunkčním. Odpověď od serveru může obsahovat i dobu, po kterou je zablokován přístup, popř. další informaci vedoucí k podrobnějšímu informování uživatele.

Za narušení bezpečnosti lze považovat i případ, kdy původ produktu nebyl ověřen, resp. toto ověření bylo neúspěšné. Může se jednat již o prvotní kontrolu podepsaných dat v tagu (nesedí jejich podpis), stejně tak o online ověření pomocí serverové části. Selhala-li kontrola již v první fázi, došlo patrně ke špatnému načtení dat z tagu produktu nebo byla tato data dokonce modifikována. Opětovným načtením dat je možné vyvrátit chybu čtení, přičemž pokud se objeví znovu, znamená to, že aplikace může mít k dispozici špatný certifikát (byl ji podvržen) anebo data opravdu neodpovídají skutečnosti, nejsou původní. V takovém případě je uživatel výrazně varován, privilegovaný uživatel pak podniká příslušné (právní) kroky své kontrolní činnosti. Selže-li online kontrola, může se opakovaně provést pro vyloučení chyby spojení. Upozornění uživatele je tu opět na místě, neboť byl porušen předpoklad důvěryhodného původu zboží.

3.7 Seznam služeb

Při analýze požadavků (viz kap. 3.4) jsme rozebrali jednotlivé pohledy systémů na sebe navzájem. Z toho vychází služby, které bude systém obsahovat, ať už u komunikačního prostředníka nebo IS výrobce. Komunikační prostředník vystavuje služby klientům a konzumuje služby od IS výrobců. Klientem se v tomto případě rozumí jak klientská mobilní aplikace (cílový uživatel), tak informační systém subjektu, tedy distributora, obchodníka. Není vyloučeno, aby IS výrobce byl také klientem komunikačního prostředníka a mohl pomocí něho čerpat určitá data. Rozdělme seznam služeb na jednotlivé části podle toho, komu jsou poskytovány, popř. jakou roli zde hraje komunikační prostředník:

- služba pro komunikaci komunikačního prostředníka s klientskou mobilní aplikací (ověřování produktu),
- služba pro komunikaci komunikačního prostředníka s IS subjektu (aktualizace produktu),
- služby, které poskytuje IS výrobce komunikačnímu prostředníkovi.

3.7.1 Komunikační prostředník poskytuje

Služby komunikačního prostředníka poskytované klientům dělíme podle jejich konzumenta, tj. na služby určené pro klientskou mobilní aplikaci a rozhraní pro IS subjektu. Každé z těchto rozhraní používá odlišný přístup, odlišnou technologii.

3.7.1.1 Služba pro komunikaci s klientskou mobilní aplikací

Služba je dle požadavků založena na REST architektuře a splňuje RESTful [52] principy. Přístupná je pouze pro čtení dat, tj. povolena je pouze HTTP metoda GET. Pro všechny další metody server odpovídá hlavičkou `405 Method Not Allowed`. Data z rozhraní jsou předávána v JSON formátu.

Pro získávání dat se musí klient u serveru autentizovat pomocí HTTP hlavičky *Authorization* (viz část 3.6.2.3). Ta musí obsahovat platné přihlašovací údaje (i pro získání dat

neprivilegovaného uživatele). Pokud údaje nejsou zadány, anebo jsou zadané nesprávně, server odpovídá hlavičkou `401 Unauthorized`.

Pokud se na serveru vyskytla jakákoliv jiná chyba, server odpovídá hlavičkou `500 Internal Server Error`, popř. dalšími chybami 50x, pokud se vyskytl neočekávaný problém.

Všechna rozhraní obsahují kromě povinných parametrů další volitelné parametry uváděné za pomoci tzv. query stringu (části URL za otazníkem).

Parametr `?certificate=`

Parametr slouží pro ověření platnost certifikátu dle požadavků. Pokud neobsahuje žádnou hodnotu, nebo obsahuje číslo 1, komunikační prostředník bude v návratových datech posílat všechny platné certifikáty výrobce. Použitím číslice 0 lze toto chování zcela potlačit. Ve všech ostatních případech se počítá s tím, že hodnotou je otisk (hash) certifikátu, který má klient uložen na své straně. Tento hash je pak prověřován vůči certifikátům uloženým na serveru. Pokud sedí, obsahuje odpověď klientovi v položce *certificate* pole s jednou hodnotou *true* (nebo 1), v opačném případě nebo v případě požadavku na obsah certifikátu obsahuje položka pole certifikátů samotných. Pokud žádný platný neexistuje, hodnotou položky je $false$ (nebo 0) nebo pole o nula položkách.

Resource Product

Rozhraní poskytuje data o produktu načtená z IS výrobce. Produktem se v tomto smyslu rozumí jedna produktová položka, tj. například jedno balení léčiva, jedna krabice mléka.

> **GET** /product/*<firm>*/*<id>*/

Parametry

firm	unikátní identifikátor výrobce načtený z NFC tagu. Akceptovány jsou alfanumerické znaky šestnáctkové soustavy, tj. 0-9, A-F.
id	unikátní identifikátor položky produktu načtený z NFC tagu. Může se jednat přímo o ID tagu (zaručená unikátnost položky). Akceptovány jsou alfanumerické znaky šestnáctkové soustavy, tj. 0-9, A-F.

Návratové hodnoty

`200 OK`	Produkt byl u výrobce nalezen.
`204 No Content`	Ověření produktu u výrobce bylo úspěšné, ale nejsou již poskytnuta žádná další data o produktu (buď nejsou nebo v závislosti na oprávnění nejsou poskytována).
`401 Unauthorized`	Nebyly zaslané autentizační údaje nebo jsou neplatné.
`404 Not Found`	Výrobce nebyl nalezen.
`405 Method Not Allowed`	Server nepodporuje komunikaci pomocí zaslané metody, povolena je pouze GET metoda.
`409 Conflict`	Produkt nebyl u daného výrobce nalezen.

`408 Request Timeout`	Komunikační prostředník nedostal včas odpověď od IS výrobce nebo je jeho služba nedostupná.
`500 Internal Server Error`	Jakákoliv jiná chyba.

Tělo odpovědi

vždy existující položky	– name – název – EAN – variant – varianta (typ) – produced – datum vyrobení
volitelné položky	– expired – datum expirace – package_size – velikost balení – package_type – typ balení – batch_num – číslo šarže – destination – místo odbytu – vat_param – daňový parametr – state – stav položky produktu – certificate – pole s obsahy vyžadovaných certifikátů (popř. výsledek kontroly otisku – viz výše)

Resource Výrobce

Rozhraní poskytuje informační data o výrobci daného produktu. Pomocí tohoto rozhraní lze snadno zažádat o certifikát výrobce či jej ověřit. Odpověď komunikačního prostředníka není závislá na dalším požadavku jako v předchozím případě, proto by data měla být klientovi zaslána okamžitě po položení dotazu.

GET `/firm/<id>/`

Parametry

id	unikátní identifikátor výrobce načtený z NFC tagu. Akceptovány jsou alfanumerické znaky šestnáctkové soustavy, tj. 0-9, A-F.

Návratové hodnoty

`200 OK`	Výrobce byl nalezen, odpověď obsahuje jeho detaily.
`401 Unauthorized`	Nebyly zaslané autentizační údaje nebo jsou neplatné.
`404 Not Found`	Výrobce nebyl nalezen.
`405 Method Not Allowed`	Server nepodporuje komunikaci pomocí zaslané metody, povolena je pouze GET metoda.
`500 Internal Server Error`	Jakákoliv jiná chyba.

Tělo odpovědi

vždy existující položky	– name – IC – address – web – e-mail

volitelné položky	– contacts – certificate – pole s obsahy vyžadovaných certifikátů (popř. výsledek kontroly otisku – viz výše)

3.7.1.2 Služba pro komunikaci s IS subjektu

Pro aktualizaci stavu produktu u výrobce slouží služba komunikačního prostředníka založená na SOAP principech. Dostupný popis služby ve formátu WSDL (Web Services Description Language) [63] poskytuje dvě metody pro aktualizaci stavu: jednotlivě či pomocí určité distribuční jednotky. Pro komunikaci je třeba zasílat přihlašovací údaje stejně jako v předchozích případech.

Řešení počítá s tím, že některým IS může být vyžadováno podepisování zpráv nebo jejich šifrování, což formát WSDL umožňuje. V navrženém řešení komunikujeme pomocí HTTPS protokolu a vlastní šifrování zpráv nepoužíváme. Je však třeba počítat s tím, že takový požadavek může být vznesen.

Aktualizace stavu produktu

Rozhraní předává IS výrobce zasílaný produkt a identifikátor firmy pro to, aby výrobce mohl rozhodnout, do jakého stavu se daný produkt má dostat. Metoda slouží pro aktualizaci pouze jednoho produktu.

updateProduct(*<firm>*, *<id>*)

Parametry

firm	unikátní identifikátor výrobce produktu. Údaj může být získán z dodacího listu či načten z NFC tagu. Akceptovány jsou alfanumerické znaky šestnáctkové soustavy, tj. 0-9, A-F.
id	unikátní identifikátor samotného produktu. Identifikátor může být získán z dodacího listu či načten přímo z NFC tagu produktu. Akceptovány jsou alfanumerické znaky šestnáctkové soustavy, tj. 0-9, A-F.

Odpověď

Metoda vrací kladný výsledek (hodnotu *true*), pokud vše proběhlo v pořádku.

Vrácené výjimky

`UnauthorizedException (401)`	Nebyly zaslané autentizační údaje nebo jsou neplatné.
`NotFoundException (404)`	Výrobce nebyl nalezen.
`BadRequestException (400)`	Výrobce odpověděl chybovou zprávou (např. produkt u něj neexistuje).
`UnknownException (500)`	Neznámá chyba serveru (pro všechny ostatní případy).

Hromadná aktualizace stavu produktu

Rozhraní předává IS výrobce místo identifikátoru produktu identifikátor distribuční jednotky. Jinak je rozhraní zcela shodné s rozhraním pro aktualizaci jednoho produktu.

```
updateProductBatch(<firm>, <id>)
```

Parametry

firm	unikátní identifikátor výrobce produktů. Údaj může být získán z dodacího listu či načten z NFC tagu. Akceptovány jsou alfanumerické znaky šestnáctkové soustavy, tj. 0-9, A-F.
id	unikátní identifikátor distribuční jednotky Údaj pochází např. z dodacího listu či je vyčten z elektronicky označeného balení. Akceptován je jakýkoliv řetězec o maximální délce 255 znaků.

Odpověď

Metoda vrací kladný výsledek (hodnotu *true*), pokud vše proběhlo v pořádku.

Vrácené výjimky

`UnauthorizedException (401)`	Nebyly zaslané autentizační údaje nebo jsou neplatné.
`NotFoundException (404)`	Výrobce nebyl nalezen.
`BadRequestException (400)`	Výrobce odpověděl chybovou zprávou (např. distribuční jednotka u něj neexistuje).
`UnknownException (500)`	Neznámá chyba serveru (pro všechny ostatní případy).

3.7.2 Komunikační prostředník konzumuje

Komunikační prostředník s výrobcem komunikuje prostřednictvím služeb, které mu jsou poskytovány. Jedná se o službu typu SOAP, výrobce vystavuje WSDL popis služby. Volané metody nemusejí mít vždy název shodný s níže uvedeným – ty jsou pouze příkladem, jak mohou metody rozhraní vypadat.

Načtení dat o produktu

Metoda poskytuje načtení dat o produktu v IS výrobce. Každý výrobce může toto rozhraní mít implementované jinak, proto by se komunikační prostředník jako konzument měl umět přizpůsobit.

```
getProductDetail(<id>, [ <items> ])
```

Parametry

id	Unikátní identifikátor samotného produktu. Ten je komunikačnímu prostředníku zaslán z klientské aplikace. Akceptovány jsou alfanumerické znaky šestnáctkové soustavy, tj. 0-9, A-F.

items	Pole požadovaných položek z dat o produktu. Toto pole sestavuje komunikační prostředník na základě oprávnění klienta a může být prázdné (nebo se parametr v požadavku nemusí vůbec objevit).

Odpověď

Metoda vrací strukturovaný objekt obsahující požadovaná data o produktu. Pokud nebyla požadována žádná data, vrací služba základní informace o produktu. Pokud jsou definována data, která má služba vrátit, návratová data obsahují pouze tyto informace. Kompletní množina dat může obsahovat tyto položky:

- name – název
- EAN
- variant – varianta (typ)
- produced – datum vyrobení
- expired – datum expirace
- package_size – velikost balení
- package_type – typ balení
- batch_num – číslo šarže
- destination – místo odbytu
- vat_param – daňový parametr
- state – stav položky produktu

Vrácené výjimky

`NotFoundException (404)`	Požadovaný produkt nebyl nalezen.
`BadRequestException (400)`	Požadovaná data nejsou k dispozici (např. některé z požadovaných polí neexistuje).
`UnknownException (500)`	Neznámá chyba serveru (pro všechny ostatní případy).

Aktualizace stavu produktu

Metoda přijímá zasílaný produkt a identifikátor firmy pro to, aby výrobce mohl rozhodnout, do jakého stavu se daný produkt má dostat. Metoda slouží pro aktualizaci pouze jednoho produktu.

```
updateProduct(<IC>, <id>)
```

Parametry

IC	identifikační číslo (IČ) subjektu, který vznesl požadavek na změnu stavu produktu. IČ zasílá komunikační prostředník podle uživatele (IS subjektu), který s ním komunikuje. Jedná se o maximálně osmimístné číslo, počet míst může být doplněn nulami.
id	unikátní identifikátor samotného produktu. Ten je komunikačnímu prostředníku zaslán z IS subjektu. Akceptovány jsou alfanumerické znaky šestnáctkové soustavy, tj. 0-9, A-F.

Odpověď

Metoda vrací potvrzovací zprávu (hodnotu *true*), pokud vše proběhlo v pořádku.

Vrácené výjimky

`NotFoundException (404)`	Produkt nebyl nalezen.
`BadRequestException (400)`	Pro dané IČ neexistuje v databázi záznam o tom, do jakého se má dostat stavu.
`UnknownException (500)`	Neznámá chyba serveru (pro všechny ostatní případy).

Hromadná aktualizace stavu produktu

Metoda přijímá kromě identifikátoru firmy identifikátor distribuční jednotky. Ten byl zaslán komunikačnímu prostředníkovi, přičemž ten jej pouze přeposílá dále. Jinak je metoda zcela shodná s rozhraním pro aktualizaci jednoho produktu.

updateProductBatch(*<IC>*, *<id>*)

Parametry

IC	identifikační číslo (IČ) subjektu, který vznesl požadavek na změnu stavu produktu. IČ zasílá komunikační prostředník podle uživatele (IS subjektu), který s ním komunikuje. Jedná se o maximálně osmimístné číslo, počet míst může být doplněn nulami.
id	unikátní identifikátor distribuční jednotky. Údaj pochází např. z dodacího listu či je vyčten z elektronicky označeného balení. Akceptován je jakýkoliv řetězec o maximální délce 255 znaků.

Odpověď

Metoda vrací potvrzovací zprávu (hodnotu *true*), pokud vše proběhlo v pořádku.

Vrácené výjimky

`NotFoundException (404)`	Žádná distribuční jednotka s tímto identifikátorem nebyla nalezena.
`BadRequestException (400)`	Pro dané IČ neexistuje v databázi záznam o tom, do jakého se má dostat stavu.
`UnknownException (500)`	Neznámá chyba serveru (pro všechny ostatní případy).

Kapitola 4

Realizace

Cílem této práce je především návrh celého řešení dle zadání a požadavků. V předchozí kapitole (viz kap. 3) byl proveden rozbor požadavků ze zadání a navrženo, jak bude systém vypadat a jak by měl fungovat. Při vlastní realizaci se budeme držet navrženého řešení. Cílem realizace je vytvořit fungující celek na základě návrhu tak, aby jednotlivé komponenty fungovaly a spolu komunikovaly. Účelem není realizovat systém, který bude možné okamžitě nasadit do provozu. Budeme se snažit tohoto stavu maximálně dosáhnout, ve většině případů, i podle zadání, vytvoříme prototypová řešení daných součástí tak, jak by ve skutečnosti mohla vypadat. Například u mobilní aplikace provedeme realizaci lo-fi i hi-fi prototypu [6], nebudeme se však nadále soustředit na to, aby byla aplikace okamžitě schopna provozu a veřejné publikace (tj. optimalizace přístupnosti, finální grafické rozhraní). Součástí realizace bude také podrobný rozbor komponent jednotlivých částí a případně jejich datového modelu.

Samotnou realizaci celého řešení rozdělíme podle jednotlivých součástí systému na tři hlavní části:

1. aplikace komunikačního prostředníka,
2. ukázkový IS výrobce,
3. mobilní klient.

Pro testování rozhraní komunikačního prostředníka budeme patrně muset vytvořit i jednoduchého klienta, který bude umět aktualizovat stav k určitému produktu (viz případ užití v 3.4.5). Tato aplikace (jednoduchý skript) bude sloužit pouze pro demonstraci řešení, proto nebyla rozebrána v analýze. V reálném nasazení se bude jednat o informační systém subjektu distribučního řetězce, např. distributora či obchodníka.

4.1 Pravidla vývoje

Pro každou část realizace budou platit specifická pravidla. Obecně je můžeme rozdělit podle použité technologie, programovacího jazyka na části: aplikace v jazyce Python a mobilní klient pro OS Android v Javě. Při programování se budeme řídit pravidly a zavedenými standardy programovacích jazyků i použitých frameworků.

Jako samozřejmou bereme skutečnost, že při programování je použito výhradně anglického jazyka, jak pro názvy tříd, rozhraní, metod a proměnných, tak i pro komentování zdrojového kódu. Komentovat zdrojový kód v mateřském jazyce je sice snazší, angličtina dává komentování univerzálnost a vyhneme se problémům ve skloňováním odborných názvů. Navíc vytvořenému kódu může porozumět vývojář se znalostí angličtiny kdekoliv na světě. Zdrojový kód by měl být čitelný již sám od sebe, komentování ho pouze doplňuje, proto není třeba do komentářů sepisovat dlouhé věty či celé statě.

4.1.1 Vývoj v jazyce Python (frameworku Django)

Pro serverovou část řešení jsme v analýze vybrali jazyk Python. Jeho typová bezpečnost nedovolí některé jinak nebezpečné konstrukce, budeme ale muset poněkud přísněji dbát na výsledný kód. Tvorba bloků odsazením právě čtyřmi mezerami není úplně tím nejdokonalejším způsobem oddělování kódu, jaký byl kdy vynalezen, i když pochopitelně přispívá k úspoře místa i minimalizmu kódu. Nesprávně odsazený blok může být snadno přehlédnut a následky mohou být nedobré. Obecně bychom měli dodržovat standardy pro psaní zdrojového kódu [59], které definují styl programování i komentování kódu. Vyvíjet budeme také v prostředí, které tyto normy samo pomáhá dodržovat.

Součástí zdrojového kódu aplikace by měla být řádná dokumentace. Pro Python používáme speciální blokové komentáře [20], které poté dokáže zpracovat nástroj Docutil [19]. Dokumentovat bychom měli řádně všechny veřejné i skryté metody a proměnné.

Jazyk Python byl vybrán také proto, že je k dispozici velké množství modulů, již hotové funkcionality, kterou jenom bude třeba dát dohromady a použít. Nebudeme tak vyvíjet vlastní implementace komunikace se službami, když jsou tyto funkcionality již dostupné v podobě hotových balíčků (package). Dalším kritériem, které zvolilo jazyk Python, byla existence frameworku Django. Samotný jazyk bez tohoto frameworku by byl při výběru nezajímavý, kdyby framework neposkytl vysoký komfort při vývoji v podobě hotových balíčků, předpřipraveného prostředí, spouštěcích skriptů i generování administračního rozhraní pro správu dat uživatelem.

Framework Django drží programátora v určitých mezích a v podstatě obsahuje předepsanou strukturu umístění jednotlivých částí zdrojového kódu. Díky tomu je tak aplikace v rukou kohokoliv poměrně snadno čitelná a dodržuje tak požadavky na snadno rozšiřitelný a přehledný kód. Projekt je v Django frameworku rozdělen na aplikace. Tohoto rozdělení využijeme proto, abychom vytvořili jednotlivé části systému, postavili jeho architekturu. Oddělovat tímto způsobem tak budeme front-end a back-end.

4.1.2 Vývoj v jazyce Java

Jazyk Java nedovoluje programátorovi mnoho magičnosti či nejasných konstrukcí. Každý kód sice může být napsán více nečitelně než čitelně, ale většinou není mnoho cest, jak udělat zdrojové kódy aplikace zcela nečitelné. K tomu dopomáhá i vývojové prostředí, které programátora drží v určitých mezích, stejně jako kompilátor jazyka samotného. Pro Javu existují všeobecné standardy pro psaní zdrojových kódů [40], které definují jeho vizuální podobu. Tyto zvyklosti částečně přebírají i další programovací jazyky (např. C#, PHP) a stávají se

tak všeobecným standardem. Vývojová prostředí jsou na tzv. „coding standards“ již připravena a často umějí podle pravidel daný kód i automaticky přeformátovat. Při vývoji se tak budeme těchto pravidel striktně držet.

Neméně důležitou součástí zdrojového kódu je jeho dokumentace. Při jejím psaní používáme stylu, který akceptuje nástroj Javadoc [41]. Dokumentaci by měly obsahovat alespoň veřejné (public) metody, aby uživatel při použití daných metod pochopil, co dělají, co je jejich účelem. Budeme-li se sami nebo někdo jiný ke kódu vracet, vítány jsou i komentáře uvnitř tříd i pro zapouzdřené (private, protected) součásti. Vlastní dokumentování kódu je tak součástí standardů a musíme jej dodržovat.

Pro OS Android je velmi specifické, že obsahuje velmi rozsáhlé API a velké množství problémů je v něm již vyřešeno. Některé situace mají již předem definované scénáře, proto nemalou část vývoje strávíme čtením dokumentace k platformě a SDK [1] (Software Development Kit). Stejně tak budeme potřebovat použít již existující knihovny např. pro realizaci spojení se serverem či kontrolu bezpečnosti. V tomto případě nemá smysl vyvíjet vlastní řešení, když ho již s největší pravděpodobností někdo vytvořil a odladil.

Právě pro co největší usnadnění vývoje nebudeme do projektu neustále vkládat knihovny, ale použijeme nástroje, které tuto práci vykonají automaticky. Historicky asi nejpoužívanějším nástrojem je GNU Make [18], který se zpravidla používá pro kompilaci zdrojových souborů v C/C++. Pro jazyk Java je asi nejpoužívanější nástroj Ant [3], který je v podstatě obdobou nástroje Make s mnohými vylepšeními. Pro správu knihoven i kompletaci zkompilovaných souborů slouží nástroj Maven [4]. Pomocí něho můžeme snadno definovat, které knihovny chceme k projektu připojit, a on je automaticky stáhne z centrálních úložišť (repozitářů). Nejnovějším nástrojem podobného typu je pak Gradle [26], jehož konfigurací jsou skripty napsané v jazyce Groovy. Při realizaci projektu vyzkoušíme použít poslední jmenovaný nástroj i proto, že jej podporuje výsledně vybrané vývojové prostředí (viz dále).

Ať už píšeme zdrojový kód v jakémkoliv jazyce, měli bychom dbát na jeho dobrou čitelnost a znovupoužitelnost. Tyto požadavky jsou zadavateli softwaru často opomíjeny, my se jich budeme co nejvíce držet. Neměli bychom tak zapomenout například na používání návrhových vzorů (design patterns) [48] či testování aplikace. Návrhové vzory byly vytvořeny proto, aby řešily již známé problémy. Pokud tedy narazíme na situaci, kdy budeme řešit určitý problém, můžeme návrhový vzor použít či se jím alespoň inspirovat.

4.2 Příprava prostředí

V předchozích částech jsme si stanovili pravidla, která jsou a budou při samotném vývoji dodržena. S tím souvisí také vývojové prostředí, IDE (Integrated Development Environment), které budeme pro práci používat, popř. pak další nástroje, které jsou potřebné. Výběr IDE závisí čistě na preferencích vývojáře, následující kapitola obsahuje pouze doporučené nástroje, které se při vývoji systému osvědčily. Krátce se věnujeme také drobné evoluci, která vývoj v tomto směru provázela.

4.2.1 Instalace podpory pro vývoj

Abychom mohli vůbec začít obě aplikace, v jazyce Java i Python, vyvíjet, musíme mít nainstalované potřebné nástroje, platformu. V této části budeme předpokládat, že domov-

ským operačním systém pro vývoj je Linux. Vývoj pod Windows je samozřejmě také možný, jenom budeme používat jiné knihovny a jiné podpůrné nástroje. Cílem však je, aby vývoj nebyl na platformě závislý.

Používáme-li Linux, jazyk Python máme již pravděpodobně nainstalovaný. Pokud tomu tak není, použijeme balíčkovacího systému pro jeho instalaci, ideálně i s prostředím pro správu vlastních balíků, nástrojů `pip` či `easy_install`. Pro Windows pak na internetu nalezneme a stáhneme instalační program, který podporu pro Python přidá do systému.

Abychom mohli vyvíjet pro jazyk Java, resp. OS Android, potřebujeme nainstalovat překladač jazyka ve formě JDK (Java Development Kit). Spolu s ním se do systému nainstaluje prostředí pro běh Java aplikací, JRE (Java Runtime Environment). To až tolik nepotřebujeme, avšak pokud budeme používat některé z dále uvedených vývojových prostředí, budeme JRE potřebovat. Pro vývoj v OS Android potřebujeme Android SDK dostupný na webových stránkách platformy [1]. SDK obsahuje kromě knihoven pro vývoj i emulátor zařízení, ve kterém budeme aplikaci testovat.

4.2.2 Vývojové prostředí

Na kvalitním vývojové prostředí závisí velmi často kvalita výsledného produktu, resp. i čas, za který je produkt vyvinut. Čím je prostředí dokonalejší, tím je vývoj pro vývojáře pohodlnější a tedy rychlejší i zábavnější. Podívejme se, jaké nástroje jsme používali při vývoji v jazyce Python i Java (OS Android).

Nasnadě je použít IDE, které by mělo integrované obě technologie a nemuseli bychom tak používat více různých prostředí. Tento požadavek splňuje například platforma Eclipse [56], pro niž existují rozšíření pro obě technologie. Prostředí je to tak velmi univerzální, až ta jeho univerzálnost může být přítěží[1].

4.2.2.1 Prostředí pro Python

Na vývoj pro jazyk Python existuje několik nejrůznějších nástrojů. Velmi používaným prostředím je rozšíření do platformy Eclipse s názvem PyDev [5]. Tento doplněk přidá do IDE podporu pro Python, kompilaci, napovídání kódu včetně podpory pro vývoj v Django frameworku. První projekt byl založen právě pomocí tohoto nástroje. Bohužel však trochu selhává napovídání kódu v oblasti Django frameworku, nebo je toto napovídání příliš obecné (jako je obecné celé IDE).

Rozhodli jsme se tak při vývoji vyzkoušet ještě další nástroj, a to komerčně vyvíjený PyCharm [32]. Existuje sice jeho komunitní bezplatná varianta, která ale neobsahuje podporu Django frameworku. Pro vyzkoušení vývoje v něm postačí 30-ti denní zkušební licence. Vývoj v tomto prostředí se ukázal být mnohem snazší a rychlejší, i spolupráce s použitým frameworkem je lepší. Protože jsme se rozhodli používat open-source prostředí i nástroje, je vývoj realizován především v prostředí PyDev, ovšem použití nástroje PyCharm se zdá být také jako velmi dobrá volba.

[1] Autor této práce je zvyklý pracovat spíše na platformě NetBeans, `http://netbeans.org`

4.2.2.2 Prostředí pro Android

Pro vývoj v OS Android vzniklo na platformě Eclipse rozšíření, plug-in, které umožňuje pro tento operační systém vyvíjet: ADT (Android Development Tools) [22]. Toto prostředí bylo po mnoho let jediným možný nástrojem pro vývoj pro tuto platformu. Vlastní projekt jsme pomocí něj tedy začali také vyvíjet. Eclipse podporuje integraci technologie Maven, proto odpadla potřeba vkládat do projektu stále nové a nové knihovny – jejich definice byla vždy přidána pouze do konfiguračního prostředí, technologie se pak postarala o jejich integraci do projektu.

Na konferenci Google I/O 2013 [23] v květnu roku 2013 byl představen nový nástroj nazvaný Android Studio [21] postavený na platformě IntelliJ [31] rusko-českého původu. Jelikož se jedná o prostředí vytvořené přímo pro vývoj pro platformu Android, rozhodli jsme se jej také vyzkoušet, ačkoliv to není příliš doporučeno, viz webové stránky [1]:

> *Caution: Android Studio is currently available as an early access preview. Several features are either incomplete or not yet implemented and you may encounter bugs. If you are not comfortable using an unfinished product, you may want to instead download (or continue to use) the ADT Bundle (Eclipse with the ADT Plugin).*

Prostředí Android Studio je velmi zajímavé mimo jiné svou rychlostí oproti Eclipse a také prostředím pro kompilaci a kompletaci (build) výsledného produktu – použitou technologií Gradle [26]. Ačkoliv stále ve vývoji, prostředí je velmi vydařené a více jak polovina času vývoje mobilní aplikace probíhala právě v něm. Velkou výhodou je integrovaný grafický designer, provázání se zdrojovým kódem i téměř bezchybné napovídání kódu (intelli-sense).

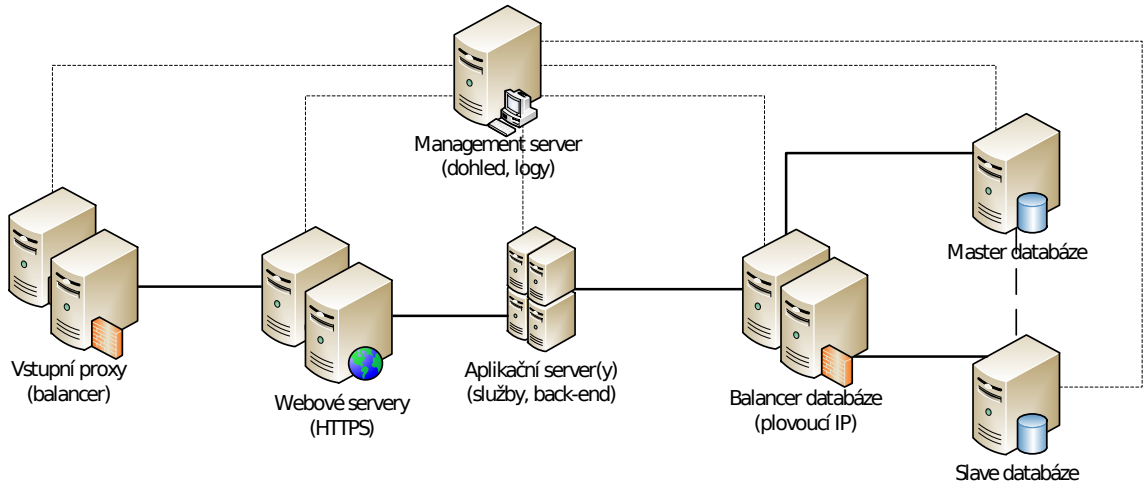
4.3 Infrastruktura komunikačního prostředníka

Pro běh aplikace komunikačního prostředníka je navrženo možné rozložení serverů. Infrastruktura splňuje náročné požadavky pro běh (viz část 2.5.3), tedy rozdělení na části podle účelu (back-end, front-end), výkonnost, dostupnost i řádný dohled a auditovatelnost (vedení záznamů o činnosti).

Vlastní prostředí zahrnuje rozložení serverů tak, aby byla zajištěna vysoká dostupnost celé služby podle požadavků. Každá z komponent by tak měla být duplikována pro pokrytí případného výpadku a zároveň rozložení zátěže. Velmi efektivně můžeme využít virtualizace, kdy naklonování jednoho stroje a vytvoření jeho duplikátu může být otázka pár hodin práce. Kromě virtualizace předpokládáme použití více fyzických serverů pro pokrytí problémů s hardwarem (výpadkem disků, pamětí apod.).

Celá infrastruktura pak ještě může běžet ve více lokalitách a být vzájemně propojena, aby bylo dosaženo vysoké dostupnosti a také rychlé odezvy klientům. Pro tento případ bude nutné vyřešit replikování dat přes všechny spuštěné instance a sdílení konfigurací i verzí samotné aplikace. Protože předpokládáme převažující čtení před zápisem dat, budeme si moct dovolit stanovit jedno zapisovací místo (které se může v průběhu času měnit), zbytek bude data pouze číst. jednotlivá místa pak můžeme propojit formou virtuálních privátních sítí (VPN) nebo síťovými tunely typu IPSec [54].

Ukázka možného rozvržení serverů je na obr. 4.1. Jak již bylo uvedeno, celé rozvržení lze zduplikovat, přičemž i každá z komponent může být taktéž obsažena vícekrát (na obr. to představuje duplicitní vyobrazení grafických prvků). Z hlediska systému či hardwaru se nemusí vždy jednat o samostatná zařízení, systémy. Jednotlivé komponenty mohou být sdruženy k dalším tak, aby byla nadále zajištěna dostupnost při výpadku kterékoliv z nich. Webový a aplikační server tak může běžet na jednom operačním systému, stejně tak balancování databáze může být fyzicky umístěno přímo na databázovém stroji.



Obrázek 4.1: Serverová infrastruktura komunikačního prostředníka

Vstupní proxy resp. balancer funguje jako firewall pro celou vnitřní síť. Není jiný způsob, jak se dostat do vnitřní sítě, než přes něj. Kromě firewallu obsahuje balancer, který rozděluje klientské požadavky webovým serverům.

Webový server je jednoduchý HTTP server (využívající protokolu HTTPS), který přijímá klientské požadavky a posílá je aplikačnímu serveru (popř. serverům). Z technologického hlediska může být vstupní proxy i webový server obsloužen stejnou aplikací, příp. na stejném serveru.

Aplikační server vystavuje služby, které mohou klientské aplikace konzumovat. Zároveň obsahuje back-end těchto služeb, který zajišťuje procesy při zpracování požadavku. Back-endy pak komunikují s databází.

Databáze je na schématu jediným zduplikovaným prvkem. Skládá se z master a slave serveru. Master slouží pouze pro zápisy, slave pak pouze pro čtení (master může obhospodařovat i určitý podíl čtecích dotazů). Předřazen databázím je **balancer databáze**, který rozděluje požadavky z back-endu a v případě výpadku jednoho serveru směřuje provoz na jiný. Sám by měl být zduplikován, aby v případě výpadku jednoho serveru zastoupil jeho funkci jiný (pomocí tzv. plovoucí IP).

Management server sbírá logy a hlídá funkce všech komponent pomocí dohledových systémů. Může také sloužit jako DHCP server přidělující IP adresy jednotlivým serverům

a s výhodou také jako interní DNS pro překlad používaných doménových jmen na cílové IP adresy.

Testovací prostředí

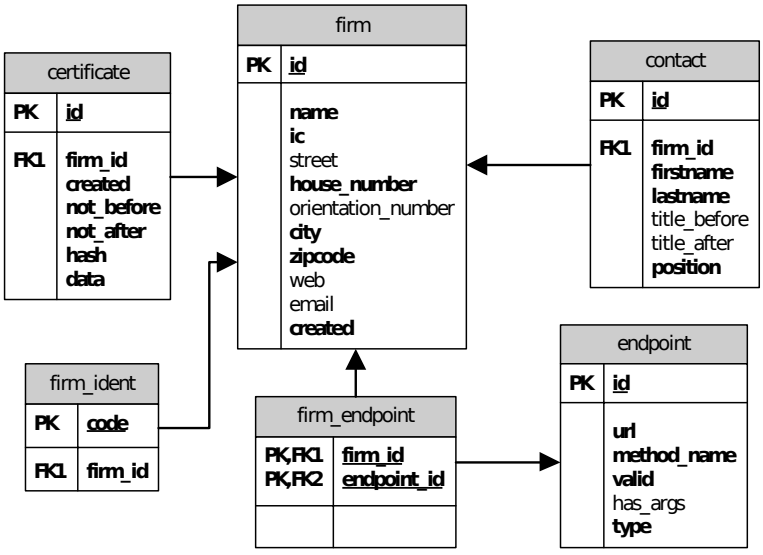
Jakýkoliv systém může obsahovat chybu. Tím je myšleno systém jak z pohledu hardwarového, tak z pohledu softwarového. V prvním případě se může jednat o výrobní vady, opotřebení nebo vliv více různých faktorů (např. nedostatečné chlazení může poškodit danou součást). Ve druhém případě se sice předpokládá použití prověřených a osvědčených nástrojů, ale ani ty nemohou být vždy zcela bezchybné. Navíc snadno může vzniknout lidská chyba ve špatné konfiguraci. I proto by veškeré zásahy do systému měly být zaznamenány a vždy nejdříve vyzkoušeny ideálně na nějakém jeho klonu. To vše hovoří pro *vytvoření testovacího prostředí.*

4.4 Aplikace komunikačního prostředníka

Klíčovou roli v celém řešení hraje aplikace komunikačního prostředníka. Běžet bude na připravené architektuře (viz 4.3). V této části probereme její podobu, tj. datový model, součásti a nahlédneme k jejím zdrojovým kódům (použitým technologiím i principům).

4.4.1 Datový model

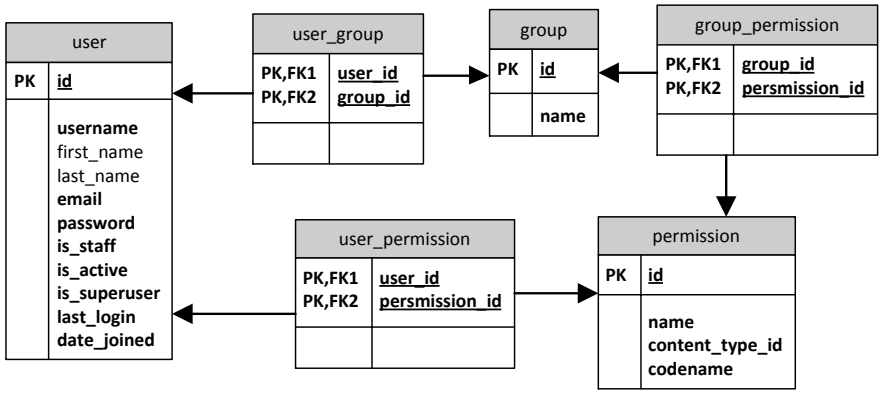
Pro uložení dat bylo zvoleno relační databázové úložiště, které je podporováno nějakým cache systémem pro urychlené vyřízení požadavků. Na pozadí jsou data strukturovaně uložena a udržují se v konzistentním stavu. Model rozdělíme na dvě částí: databázi firem, výrobců a databázi uživatelů (tak, jak to bylo konceptuálně rozděleno v analýze požadavků, viz 3.4.8).



Obrázek 4.2: Model databáze firem v komunikačním prostředníkovi

K firmě ukládáme její základní evidenční údaje (adresu, web, e-mail) spolu s kontaktní osobou. K entitě firmy jsou pak připojeny certifikáty (může jich být více) a definice rozhraní, které je použito pro připojení se při získání informací o produktu. Struktura rozhraní (endpoints) je založena na tom, že může být definováno více rozhraní pro jednu firmu, přičemž právě jedno je aktuálně aktivní. Komunikační prostředník se také přizpůsobuje tomu, že metody rozhraní mohou mít u různých IS různé názvy. Pro identifikaci navenek slouží unikátní identifikační kód výrobce, který se používá při komunikaci s klienty (atribut *id* je tak použit pouze interně). Celé schéma je znázorněno na obr. 4.2.

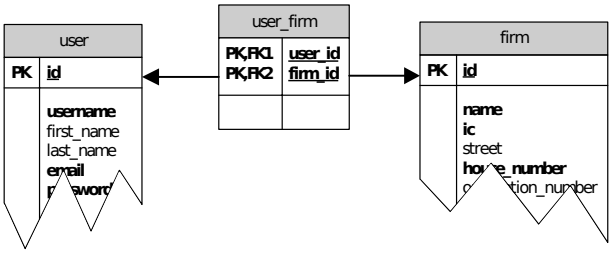
Uživatelská databáze je navržena ve standardní podobě, jaká se používá ve většině webových aplikací, které obsahují určité řízení práv. Její model je v podstatě předurčen použitým frameworkem pro vývoj. Při rozboru použitých technologií (viz 3.5.2) jsme se rozhodli použít jazyk Python a Django framework. Ten potřebnou strukturu pro řízení uživatelů a jejich oprávnění sám vygeneruje a zároveň můžeme použít již hotové modely pro práci s uživatelskými daty.



Obrázek 4.3: Model databáze uživatelů v komunikačním prostředníkovi

Obrázek 4.3 zobrazuje model uživatelské databáze umožňující řadit uživatele do skupin, těm přidávat oprávnění, ale i řídit práva přímo na úrovni uživatele. Model počítá s tím, že uživatel se autentizuje jménem a heslem. Takové přihlášení je pro základ aplikace dostačující, pro přístup do administrační aplikace je možné toto rozšířit o dvoufaktorovou autentizaci podle specifikovaných požadavků. Kromě přístupových údajů jsou u uživatele uloženy i další údaje jako je jméno, e-mail či datumové informační položky.

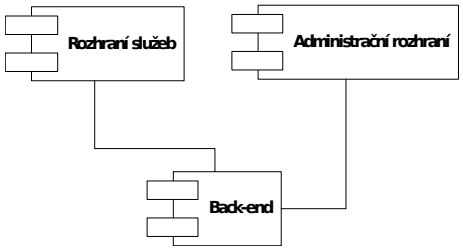
Obě dvě části databáze v podstatě nemusejí být ani svázány, pokud bychom nepotřebovali na základě přihlášeného uživatele vybírat nějaká data z databáze firem. Po podrobné analýze požadavků však zjistíme, že abychom mohli provést aktualizaci stavu produktu v IS výrobce, musíme mu odeslat informaci o tom, kdo aktualizaci provádí. To zjistíme jedině tak, že vytvoříme vazbu uživatele s firmou. Aby bylo spojení univerzální, použijeme pro to vazební tabulku a spojíme části tak, jak je zobrazeno na nákresu 4.4. Pomocí vytvořené vazby budeme moci následně také řídit oprávnění při přístupu do administrace např. formou omezení editace uživatele pouze na určitou firmu, firmy.



Obrázek 4.4: Model vazby databáze uživatelů a firem v komunikačním prostředníkovi

4.4.2 Části aplikace

Při návrhu aplikace jsme se rozhodli ji rozdělit na tři základní součásti, které jsou tvořeny balíky tříd a metod. Části jsou mezi sebou propojeny, jak je znázorněno na obr. 4.5. Komunikační prostředník kromě vystavení služeb potřebuje služby ještě konzumovat a spolupracovat při tom s back-endem, který představuje databázovou vrstvu.



Obrázek 4.5: Rozdělení aplikace komunikačního prostředníka

Back-end obsahuje především objektový model, který je promítnutý do databáze jako tabulky se sloupci (viz datový model v části 4.4.1). Vlastní model obsahuje definice atributů a jejich omezení. To dává možnost validovat vstupní hodnoty přímo na společném back-endu, nemusí to každé rozhraní dělat samo (tj. rozhraní služeb i administrační rozhraní). Integrována je tu cache vrstva pro urychlené odpovídání pro větší množství dotazů.

Rozhraní služeb tvoří pohledy (views), které představují bránu do aplikace pro připojující se služby. Zde jsou definovány cílové adresy REST API a třídy, metody, které je obsluhují. Metoda přijímající požadavek obsahuje část aplikační logiky, které za pomoci back-endu vyhledá potřebné údaje a rovnou je vrátí zpět, případně provede dotaz na server výrobce a teprve na jeho základě zašle odpověď klientovi.

Administrační rozhraní nazvané v případech užití též jako Admin aplikace spravuje vložená data v uživatelsky přijatelné formě (není třeba zásahu do databáze). Jeho vývoj není primárním cílem tohoto projektu, a tak si vystačíme s použitím a úpravami již hotové aplikace Django Admin, kterou nabízí použitý Django Framework. Díky tomu se můžeme podrobně zabývat přípravou dalších částí aplikace a nemusíme velkou část času věnovat tvorbě administračních funkcí, bez kterých by se celá serverová část stěží provozovala.

4.4.3 Vlastní aplikace

Podle návrhu je aplikace komunikačního prostředníka napsána v jazyce Python s podporou Django frameworku. Rozdělena je konceptuálně na části uvedené v kap. 4.4.2 s tím, že definice administračního rozhraní je podle pravidel Django umístěna uvnitř aplikace *backend*. Kromě konfigurace, spouštění serveru a nastavení URL (routování) je dále vytvořena Django aplikace *service* obsahující vlastní rozhraní služeb a k tomu podpůrné moduly.

Při realizaci použijeme s výhodou již hotové balíčky, moduly, které umožní aplikaci poměrně snadno zprovoznit. Nemusíme se tak zabývat např. převodem dat do JSON formátu ani definicí popisu služby ve WSDL.

4.4.3.1 Použité technologie

Jazyk Python byl vybrán také z toho důvodu, že pro něj existuje mnoho knihoven, které řeší některou požadovanou funkčnost. Protože ne všechny knihovny jsou již dostupné pro verzi Python 3, budeme nadále používat interpret verze 2.7. Jinak by vytvořený kód měl být kompatibilní i se třetí verzí.

První částí realizace je vytvoření REST služby. Pro to máme k dispozici mnoho různých knihoven, výsledné řešení používá Django REST Framework [10], který přímo spolupracuje s použitým Django frameworkem. Jednoduše umí zpřístupnit modely pro jejich snadné prohlížení i aktualizaci. Pro aplikaci komunikačního prostředníka potřebujeme pouze metodu GET, i tak technologie usnadní práci (podporuje autentizaci a autorizaci). Pomocí definování URL určíme metodu, která má danou část obsluhovat. Tato metoda pak vrátí výsledek v podobě objektu `Response` ať už obsahujícím data či stavové kódy, např.:

```
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status

class ProductDetail(APIView):
    def get(self, request):
        return Resource(status=status.HTTP_204_NO_CONTENT)
```

Pro připojení na aplikaci IS výrobce musíme podporovat konzumaci služeb popisovaných WSDL souborem. Tuto funkcionalitu umožňuje framework suds [25]. Použita je upravená verze původních autorů, která vylepšuje některé vlastnosti a opravuje chyby. Jeho použití je velmi jednoduché, stačí definovat umístění WSDL souboru a následně je možné volat v něm uvedené metody, např.:

```
from suds.client import Client

deep_thought = Client('http://deep-thought.space/api/?wsdl')
result = deep_thought.getAnswerToUltimateQuestion()
print(result)    # prints number 42
```

Zároveň je potřeba vytvořit rozhraní služeb pro aktualizaci stavu produktu. Podle požadavků se jedná o SOAP službu, tedy vytvoříme rozhraní služby, které popisuje WSDL soubor. K tomu je s výhodou použito frameworku spyne [7], s jehož pomocí jsou vytvořeny metody rozhraní služby. Definováním Python anotací (decorators) lze z definované metody vytvořit metodu publikovanou v rámci služby, např.:

```
from spyne.service import ServiceBase
from spyne.decorator import rpc
from spyne.model.primitive import Integer

class DeepThoughtService(ServiceBase):
    @rpc(_returns=Integer, _in_message_name='getAnswerToUltimateQuestion')
    def get_answer_to_ultimate_question(self):
        for year in range(0, 7500000): pass
        return 42
```

Poslední součástí je jednoduchá práce s certifikáty. Ty si komunikační prostředník sám ukládá, tj. potřebuje kontrolovat jejich platnost a při komunikaci s klientem také ověřuje jeho existenci popř. jej klientovi zasílá. K tomu slouží knihovna pyOpenSSL [9], ze které použijeme modul `crypto`.

4.4.3.2 Práce s certifikáty

Certifikát je v databázi aplikace uložen ve své textové podobě. K tomu se ukládá i rozsah jeho platnosti (datum od-do) a jeho číselný otisk. Tyto údaje jsou automaticky vyčteny z certifikátu pomocí knihovních funkcí, uživatel do nich nijak nezasahuje. Certifikát je možné vložit v administračním rozhraní, při ukládání na back-endu se provádí kontrola, zda je certifikát platný.

Podle parametru `?certificate` (viz 3.7.1.1) se určuje, zda klient požaduje zaslání certifikátu nebo ověření jeho platnosti. To je prováděno při zpracování REST požadavku na server. Počítá se s tím, že otisk certifikátu je zasílán v hexadecimálním formátu, který se převádí na číslo uložené v databázi. Podle něj se pak dohledává platný certifikát a zpracuje odpověď klientovi.

4.4.3.3 Serializace

O převod Python objektů do formátu JSON se starají třídy serializace. Pro načtení údajů o firmě s výhodou použijeme předpřipraveného `ModelSerializer`, jemuž pouze definujeme položky, které chceme klientovi zasílat. Pro odeslání informací o produktu musíme položky nadefinovat kompletně sami, protože data pocházejí z rozhraní služby IS výrobce.

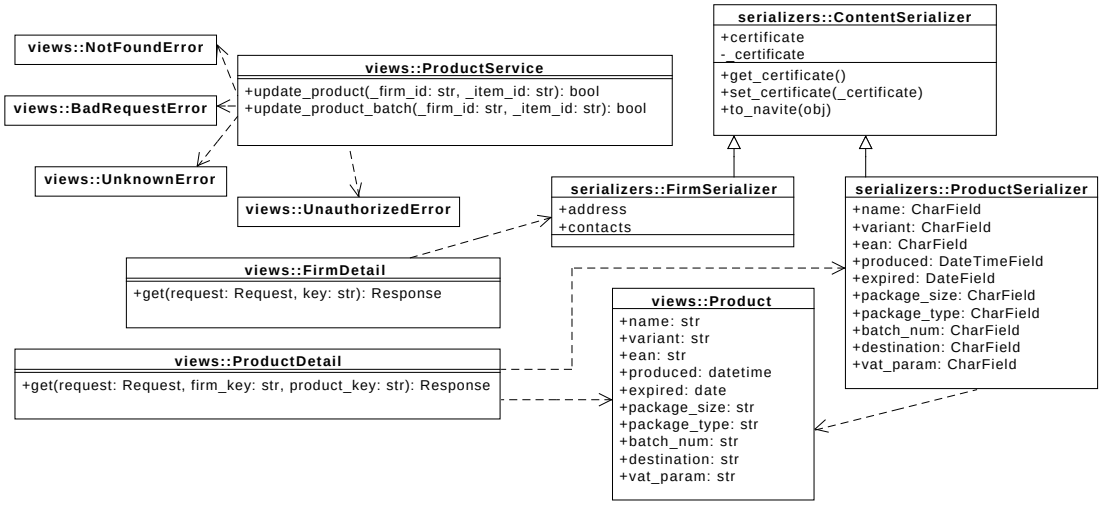
Je tak vytvořen vlastní objekt serializace `ContentSerializer`, který kromě převodu na JSON umožňuje vložení certifikátu. Tato třída navíc z výsledných dat odstraní atributy, které mají hodnotu *None*.

4.4.3.4 Diagram tříd

Kromě datového modelu (viz 4.4.1), který je převeden do objektů v části back-endu a doplnění tříd pro operaci nad nimi (managers), projekt obsahuje třídy pro zpracování požadavku, komunikaci s IS výrobce a sestavení odpovědi. Přehled metod a atributů zobrazuje digram tříd na obr. 4.6.

V části *views* jsou umístěny třídy pro klientská rozhraní, ať už REST či SOAP, které sestavují odpověď pro klienta. Tyto objekty pak komunikují s back-endem, kde je umístěn

vlastní datový model aplikace. V rozhraní REST se používá již zmíněná serializace pro sestavení dat ve výsledném formátu JSON. V případě rozhraní REST se chybové stavy realizují kódy HTTP odpovědí, pro SOAP to jsou definované výjimky.



Obrázek 4.6: Diagram tříd komunikačního prostředníka

4.5 Informační systém výrobce

Další částí realizace je implementace prototypu informačního systému výrobce. Tato aplikace slouží pro demonstraci celé komunikace a vůči tomu ji budeme také věnovat svou pozornost. Ve skutečnosti tyto aplikace již existují, ale neobsahují požadovanou funkcionalitu. Cíl tedy není vytvoření dalšího z řady informačních systémů, vytváříme jej pouze jako exemplární příklad proto, aby fungovala kompletní komunikace a splnili jsme tak stanovený cíl této práce.

Pro demonstraci řešení tak bude postačovat, pokud bude fungovat rozhraní služby přijímající zprávy od komunikačního prostředníka a grafická administrace systému, pomocí níž budeme moci nahlížet do uložených dat a spravovat je.

Serverová architektura

Z pohledu serverového je architektura IS výrobce teoreticky totožná s rozložením serverů pro komunikačního prostředníka. Vytvářený prototyp nebude disponovat tak rozsáhlým zázemím a nejspíše se spokojíme s jedním až dvěma servery, které sjednotí veškerou funkcionalitu dohromady. V reálné praxi je celý IS často dodáván jako ucelený produkt včetně hardwaru popř. jsou poskytnuty konzultace s jeho výběrem.

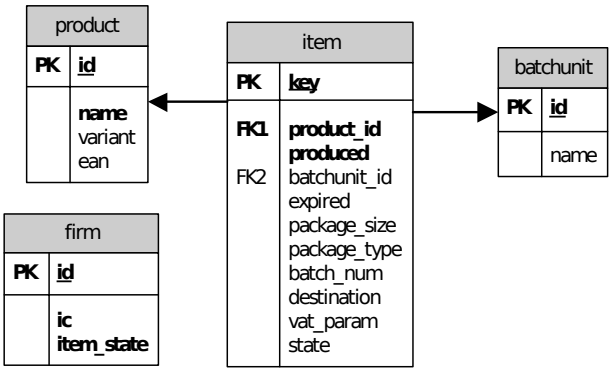
4.5.1 Datový model

Ve skutečném informačním systému bude datový model několikanásobně větší. Pro tento případ stačí pouze čtyři tabulky zobrazené na obr. 4.7. Obsahuje pouze výřez toho, co je nutné pro fungování vytvořeného prototypu.

Návrh počítá s tím, že existuje jeden produkt daného typu, ke kterému jsou vyrobeny jednotlivé položky. V celém textu objevující se označení produkt, výrobek či zboží tak náleží vlastně jedné položce daného typu. Tato položka má vlastní identifikátor (ideálně shodný s unikátním identifikátorem v tagu) a různé parametry, jako:

- datum výroby,
- datum spotřeby,
- velikost balení,
- typ balení,
- číslo šarže,
- místo odbytu,
- daňový parametr,
- stav položky v distribučním řetězci.

Tyto položky jsou závazné pro komunikaci s komunikačním prostředníkem. Model může obsahovat ještě další atributy, stejně jako může být některý z atributů přesunut spíše do entity produktu (product). Ta je v tomto smyslu označením spíše pro daný typ produktu, druh.



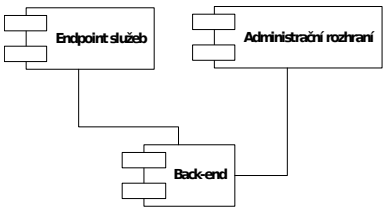
Obrázek 4.7: Datový model databáze IS výrobce

Každá položka obsahuje informaci o stavu. Tento atribut může či nemusí být využíván, v demonstračním řešení je umístěn pro ukázku možné aktualizace stavu produktu na základě jeho pohybu v distribučním řetězci. Aby bylo možné stav správně vyhodnotit, obsahuje model ještě databázi firem. V uvedeném případě se omezujeme pouze na uložení IČ (Identifikační číslo podnikatelského subjektu v Česku) a definici, do jakého stavu se má položka nastavit,

byl-li zaslán požadavek o aktualizaci stavu právě s tímto IČ (to aplikaci zasílá komunikační prostředník). Kromě toho databáze obsahuje evidenci distribučních jednotek, v nichž může být produktová položka zahrnuta. To slouží pro hromadnou aktualizaci produktů.

4.5.2 Části IS výrobce

Aplikaci IS výrobce jsme se stejně jako aplikaci komunikačního prostředníka rozhodli rozdělit na tři základní součásti, které následně obsahují jednotlivé balíky tříd a metod. Části jsou mezi sebou propojeny, jak je znázorněno na obr. 4.8. Cílem je minimalistická fungující aplikace pro ukázku, jak může rozhraní pro poskytování dat vypadat.



Obrázek 4.8: Předpokládaná architektura IS výrobce

Back-end je jádrem celého systému. Obsahuje datový (objektový) model aplikace, základní aplikační logiku.

Administrační rozhraní je vytvořeno pouze pro testování celého systému. Obsahuje pouze základní funkcionalitu tak, aby bylo možné přistupovat k uloženým datům a přijatelným způsobem je spravovat. Některá v něm obsažená funkcionalita (např. vytvoření produktu) je v reálném IS určitě zautomatizována a napojena například přímo na výrobní linku.

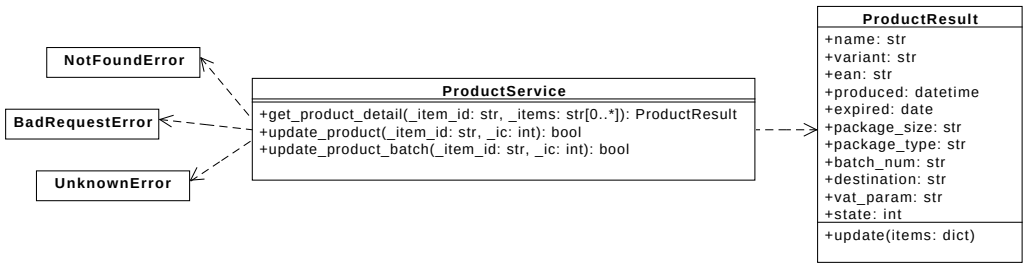
Endpoint služeb představuje rozhraní vystavených služeb pro komunikaci s okolním světem, v realizovaném případě především s komunikačním prostředníkem. Každý IS může mít toto rozhraní trochu jiné a je na komunikačním prostředníkovi, aby jej dokázal používat.

4.5.3 Vlastní aplikace

Aplikace je vytvořena v jazyku Python s pomocí Django frameworku, jak bylo definováno na základě analýzy. Projekt je rozdělen na Django aplikace podle návrhu částí z předchozího bodu. Definice administrace je podle Django umístěna uvnitř aplikace *backend*. Nastavení aplikace, definování URL (routování) a spuštění serveru je umístěno uvnitř základní aplikace *manufacturer*. Aplikace *service* pak obsahuje definici zobrazení (views) pro vytvoření vlastního rozhraní služby (endpoint).

Pro vytvoření rozhraní služby je použito, stejně jako v případě komunikačního prostředníka (viz 4.4.3.1), frameworku spyne [7]. Ten umožňuje vytvořit třídu, jejíž metody se za pomocí Python anotací (decorators) převedou na rozhraní služby, které vytvoří výsledný WSDL soubor popisující službu jako takovou. Diagram tříd je tak velmi jednoduchý, zobrazen je na obr. 4.9. Každá metoda obsluhuje jednu ze tří metod rozhraní, zpět vrací objekt

(ve WSDL definováno jako complex type) `ProductResult` nebo *bool* hodnotu, případně vyvolává definované výjimky v popisu služby z části 3.7.2.



Obrázek 4.9: Diagram tříd IS výrobce

4.6 Mobilní klient

Podle zadání i požadavků máme vytvořit mobilní aplikaci, která bude číst informace uložené v tagu produktu a ověřovat tak vlastní původ zboží. V této části se budeme zabývat podobou této aplikace, vytvoříme lo-fi a hi-fi prototyp [6], připravíme ukázkový projekt jako možnou realizaci celého systému. Mobilní klient má být cílovým rozhraním, pomocí něhož uživatelé budou službu používat. Neočekává se, že by aplikace byla zcela dokonalá, má sloužit především jako ukázka možného využití služby komunikačního prostředníka. V závislosti na tom bude navržena také její architektura, rozhraní a chování.

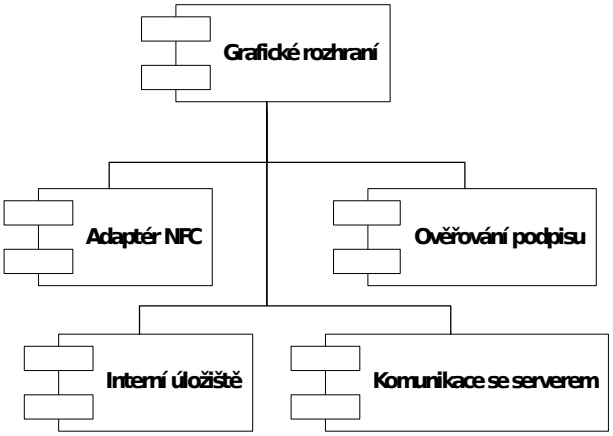
4.6.1 Součásti klientské aplikace

Aby byla aplikace znovupoužitelná, rozdělíme ji na jednotlivé části. Tím hlavním je rozdělení na **knihovnu** (library) a **vlastní projekt**. Do knihovny umístíme veškerou potřebnou funkcionalitu aplikační (business) logiky, tedy části, které může vyžít i další implementace projektu, další aplikace. V této části se budeme věnovat předpokládanému rozvržení aplikace z hlediska její struktury, kdy definujeme spojení obou částí dohromady.

Na obr. 4.10 je vyobrazena architektura mobilního klienta, jak si ji lze představit na základě analýzy požadavků. Jednotlivé komponenty spojuje grafické rozhraní, které využívá jejich funkcionalitu. Vše je koncipováno tak, aby komponenty bylo možné použití i v jiné aplikaci, tj. aby byly součástí knihovny.

Grafické rozhraní spojuje veškeré komponenty do jednoho celku a poskytuje uživateli prostředek, jak s aplikací komunikovat. Rozhraní má být navrženo dle pravidel dobrého návrhu pro OS Android.

Adaptér NFC slouží ke čtení dat z tagu produktů. Navržen je jako samostatná jednotka s rozhraním, které může být uvnitř implementováno různými způsoby (např. NFC lze vyměnit za čtení QR kódů). Předpokládáme, že při vývoji budeme potřebovat toto rozhraní simulovat, protože prozatím nemáme dostupnou veškerou technologii (zařízení s NFC, tagy s nahranými daty).



Obrázek 4.10: Konceptuální součásti mobilního klienta

Interní úložiště s výhodou využijeme pro ukládání již vyčtených dat, která nepodléhají tak častým změnám (např. některé údaje o výrobci, jejich certifikáty apod.). Forma tohoto úložiště může být různá, nejsnáze využijeme SQLite databázi integrovanou přímo v operačním systému. V případě potřeby můžeme využít i paměťového média v zařízení.

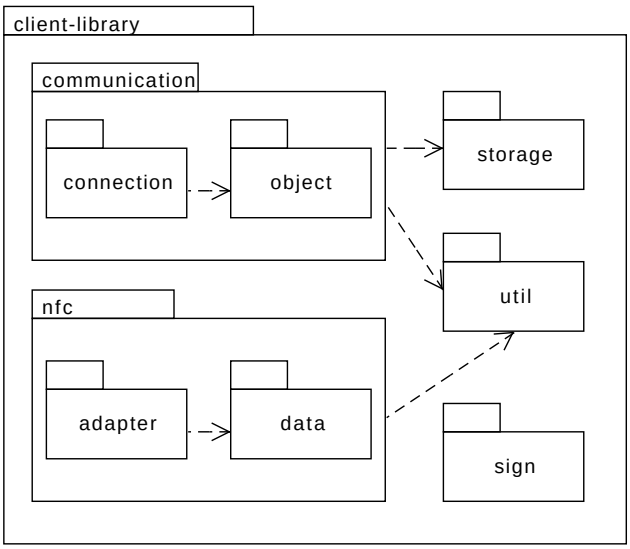
Ověřování podpisu probíhá na základě dat vyčtených s tagu adaptérem NFC. Komponenta také potřebuje pro svou funkci certifikát výrobce uložený v interním úložišti nebo načtený ze serveru. Navenek komponenta vystupuje pod rozhraním a uvnitř tedy může být implementována různými způsoby (např. pomocí různých knihoven či vlastní implementací ověření).

Komunikace se serverem je velmi důležitá pro chod celé aplikace. Komponenta komunikující se serverem konzumuje služby serveru a poskytuje data ze serveru vrácená dalším částem. Tato data mohou být pak ukládána do interního úložiště.

4.6.2 Projekt knihovny

Na základě rozdělení na součásti v předchozí kapitole (viz 4.6.1) je vytvořena knihovna (Android Library), kterou je možné přidat ke kterémukoliv projektu, aplikaci, a poskytnout tak potřebnou funkcionalitu pro čtení tagů, ukládání do interního úložiště a ověřování podpisů i původu produktu pomocí serveru komunikačního prostředníka.

Podle částí je vytvořen diagram balíků uvedený na obr. 4.11. Naznačeny jsou také jednotlivé vazby mezi vlastními balíky, resp. jejich použití. Většina tříd je rozmístěna právě pomocí balíků, některé jsou postaveny obecněji mimo ně. Pojďme se krátce podívat na jejich funkce. Kompletní diagramy tříd (class-diagram) jsou uvedeny v příloze D této práce. V této části budeme uvádět zjednodušené diagramy tříd bez jejich obsahu, čistě pro pochopení vazeb mezi nimi.

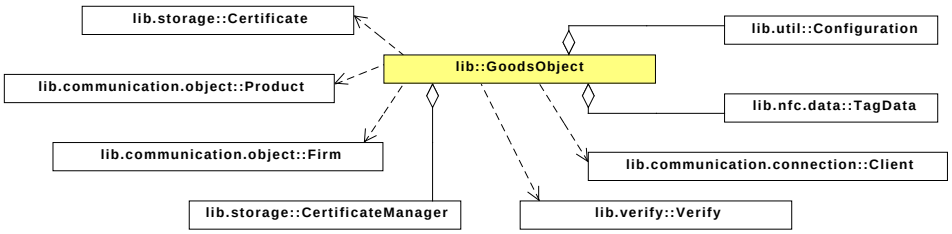


Obrázek 4.11: Diagram balíků pro klientskou knihovnu

4.6.2.1 Vše v jednom objektu

Pro snazší používání celého balíku byla vytvořena třída `GoodsObject`, která přijímá objekt přečtených dat z tagu (`TagData`) a implementuje veškerou funkcionalitu celého balíku tříd. K dispozici je tak okamžitě ověření podpisu v tagu či ověření online a načtení dalších dat k produktu.

Vazbu tohoto objektu na další třídy z knihovny zobrazuje diagram 4.12. Objekt používá dostupných tříd v balíku, aby spojil dohromady veškerou potřebnou funkcionalitu a uživateli knihovny tak dal nástroj, pomocí něhož může snadno knihovnu začít ihned používat. Ukrytou funkcionalitu je však možné zcela nahradit vlastní implementací a tento objekt sloužící především pro demonstraci spojení jednotlivých tříd.

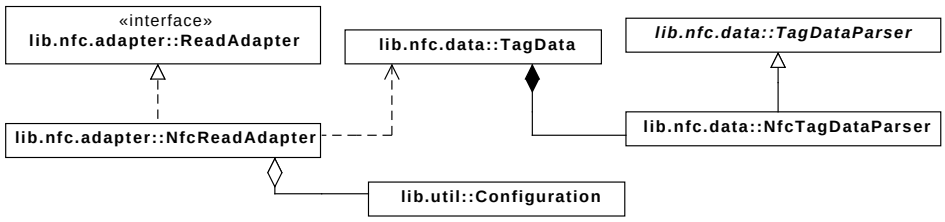


Obrázek 4.12: Vazby mezi třídami knihovny a objektem `GoodsObject`

4.6.2.2 Adaptér NFC

Čtení tagů zajišťuje NFC adaptér [11]. Ten i tak obsahuje rozhraní, které umožní použít jiného čtecího adaptéru a parseru načtených dat. Definice použitých adaptérů je uložena ve třídě `Configuration` pro snadnou změnu parametrů na jednom místě.

Na obr. 4.13 je uveden zjednodušený diagram tříd pro čtecí adaptér. Základem je rozhraní `ReadAdapter`, které definuje metody, které čtecí třída musí implementovat, aby mohla data přečíst a uložit je do objektu `TagData`. Tento objekt pak v sobě pomocí konfigurace obsahuje třídu pro parsování daných dat, která je odvozena od abstraktní třídy `TagDataParser`. Parser dokáže v předložených datech najít příslušnou hodnotu, o kterou jej objekt žádá, a vrátit ji v požadované podobě.



Obrázek 4.13: Zjednodušený diagram tříd NFC adaptéru

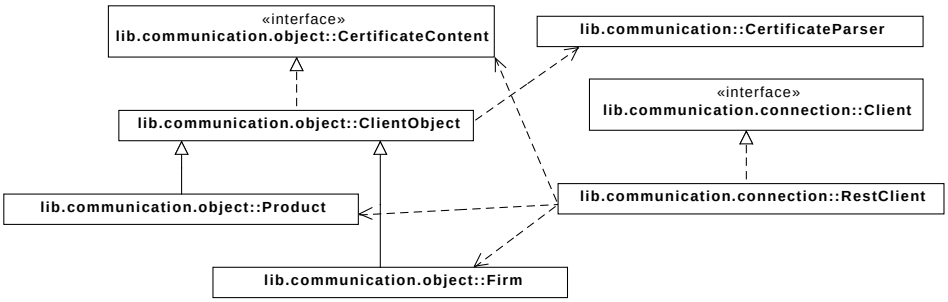
4.6.2.3 Komunikace se serverem

Součást, která komunikuje se serverem, poskytuje navenek rozhraní, v němž vrací objekty, které byly ze serveru získány. Přitom také probíhá verifikace předloženého certifikátu. Balík umožňuje použít jiný adaptér pro komunikaci než REST klienta, popř. využít jiné knihovny pro komunikaci se serverem. V realizované knihovně se jedná o knihovnu Spring for Android [24].

Jak vidíme na zjednodušeném diagramu tříd 4.14, třída `RestClient` implementuje rozhraní `Client`. Tím je definována možnost použití jiné implementace komunikace se serverem při zachování shodných vlastností. Rozhraní definuje sadu výjimek, které musí být ošetřeny, pokud by komunikace selhala. Rozhraní i jeho implementace vrací již hotové objekty, `Firm` a `Product`, které obsahují stažená data. Tyto třídy dědí od společného `ClientObject`, který definuje práci s certifikáty ze serveru staženými. K jejich převodu na objekt slouží třída `CertificateParser`. Ta přijímá několik různých vstupů, na jejichž základě provede pokus o převedení do objektu certifikátu, třídy `X509Certificate`. Pro případ neúspěchu parsování certifikátu je třeba ošetřit sadu různých výjimek (ty diagram neuvádí). Rozhraní `CertificateContent` slouží pouze pro definici metod pro práci s certifikáty, jak je vyžaduje jeho ověření v objektu `RestClient`. To je z toho důvodu, že třída `ClientObject` není vedena jako veřejně dostupná pro jiný balíček (package).

4.6.2.4 Interní úložiště

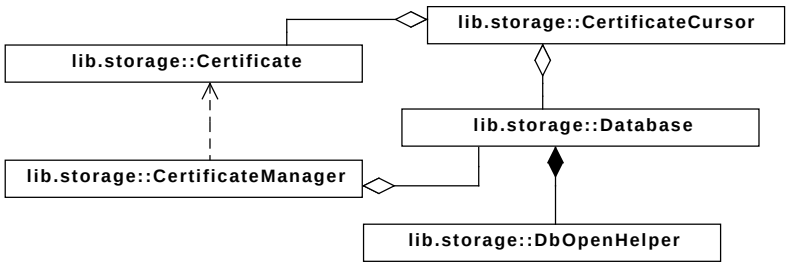
Pro ukládání certifikátů výrobců stažených ze serveru komunikačního prostředníka byla stanovena komponenta interního úložiště. S výhodou použijeme pro tuto funkcionalitu databázi SQLite integrovanou přímo v OS Android. Úložiště certifikátů, jak bychom tuto komponentu mohli také nazvat, tak poskytuje rozhraní pro vkládání a aktualizaci stažených certifikátů pro jejich pozdější použití. Tímto způsobem je možné ověřovat podpis dat v tagu



Obrázek 4.14: Zjednodušený diagram tříd balíku komunikace se serverem

i bez internetového připojení, offline. Ačkoliv tak není původ produktu zcela zaručen, je toto ověření prvním stupněm pro kontrolu pravosti původu produktu.

Na obr. 4.15 je zobrazen zjednodušený digaram tříd této komponenty. Navenek jsou exportovány pouze dvě třídy: `Certificate` a `CertificateManager`. Ostatní třídy jsou ukryté uvnitř balíku. Objekt `Certificate` slouží jako kontejner pro data uložená o certifikátu v databázi. Obsahuje jeho atributy i samotnou instanci objektu certifikátu typu `X509Certificate`. Zároveň obsahuje statické metody, které umožní vytvářet nový objekt ze stažených dat nebo dat z databáze. Do databáze je možné tak přistupovat pomocí třídy `CertificateManager`, jež slouží pro načítání, vkládání a aktualizaci objektů certifikátů. Kromě vlastní aktualizace umožňuje uložení nového časového razítka pro případ, že byl certifikát znovu validován vůči serveru. Pro procházení jednotlivých certifikátů slouží třída `CertificateCursor`, která implementuje rozhraní `Iterator`. Pomocí něho je tak možné jednotlivé vrácené záznamy načítat. Podporu všem objektům dělají třídy `Database` a `DbOpenHelper`, které obsahují vlastní instanci spojení do databáze popř. otevírají databázi pro aktuálně potřebnou funkcionalitu – čtení či zápis.



Obrázek 4.15: Zjednodušený diagram tříd úložiště certifikátů

4.6.3 Vlastní aplikace – grafické rozhraní

V první fázi realizace mobilní aplikace se věnujeme hrubému návrhu jejího uživatelského rozhraní, tj. low-fidelity prototypu. Ve druhé fázi je pak vytvořena funkční aplikace, high-fidelity prototyp, který bude simulovat kompletní funkčnost aplikace na smyšlených datech.

K tomu není třeba ani podpory v hardwaru (NFC) ani spojení se serverem. V poslední fázi vývoje pak vyměníme simulační třídy za reálné, aby komunikace probíhala tak, jak bylo navrženo.

4.6.3.1 Lo-fi prototyp

V rámci prvního prototypu aplikace splníme požadavky na zobrazení informací o produktu uložených v tagu a načtení dalších dat ze serveru. Při tom také probíhá ověření podpisu v tagu uloženém, resp. samotný původ produktu u konkrétního výrobce.

Zohledňujeme také chybové stavy, tj. pokud komunikace se serverem selhala a nebylo možné data načíst, popř. došlo k chybě při ověření, tedy server vrátil chybový stav. Další upozornění pro uživatele již mají podobnou podobu. V nákresech prototypu také zobrazujeme, jak by mohlo vypadat přihlášení pro privilegovaného uživatele, viz obr. B.3. V další implementaci již s touto modifikací aplikace nepočítáme.

Všechny nákresy prototypu jsou uvedeny v příloze B této práce.

4.6.3.2 Hi-fi prototyp

Druhou části prototypu aplikace je realizace grafického rozhraní na koncové platformě, OS Android. Oproti reálné verzi budeme simulovat chování při komunikaci se serverem i čtení dat z tagu. K tomuto je v knihovně aplikace připraveno kompletní prostředí, tzv. Fake adaptéry. Pro jejich použití existuje simulační konfigurace, třída `FakeConfiguration`.

Adaptér pro simulaci chování obsahuje z části statická data, z části generovaná, jinak se chová zcela stejně, jako kdyby komunikace probíhala přímo se severem, resp. přímo s NFC adaptérem. Při realizaci této části prototypu jsme ověřili, že navržené uživatelské rozhraní funguje v pořádku i na samotné platformě s jejími grafickými prvky.

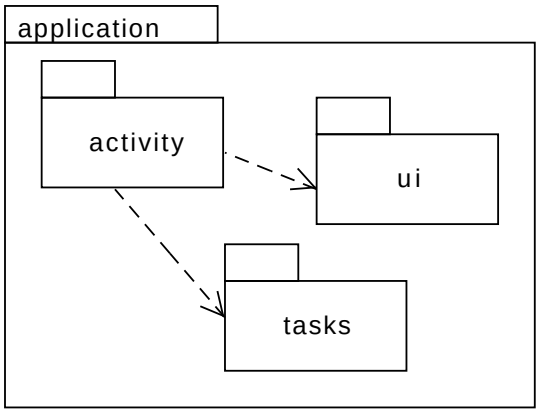
Nasnímané obrazovky prototypu jsou uvedeny v příloze C této práce.

4.6.3.3 Realizace mobilní aplikace

Protože většina funkcionality je umístěna v knihovně, aplikace samotná obsahuje pouze grafické rozhraní a jeho obsluhu. Při tom používáme standardních grafických prvků, které platforma nabízí.

Pro použití prvků z novějších verzí platformy je použito podpůrného frameworku [2], díky němuž můžeme grafické elementy známé z novějších verzí OS Android použít i ve starších verzích. Touto formou je tak vytvořeno navigační menu a použito tzv. fragmentů.

Pro zpracování vzdálených volání se serverem používá aplikace řešení v podobě třídy `AsyncTask`. Ta umožňuje naplánovat vykonávání úlohy na pozadí spolu se zobrazením dialogu informujícího načítání dat (třída `ProgressDialog`). Na obr. 4.16 je znázorněn jednoduchý diagram balíků, jak jsou jednotlivé třídy rozčleněny. Většina funkcionality je jinak uchována ve třídách `MainActivity` a `DetailActivity`, které představují jednotlivé obrazovky (activity) aplikace. Využita je veškerá funkcionalita z knihovny tak, aby bylo nutné se v aplikaci věnovat především grafické části a logiku fungování nebylo potřeba dopodrobna řešit.



Obrázek 4.16: Diagram balíků mobilní aplikace

Aplikace využívá jako základ objektu `GoodsObject` z realizované knihovny. Pomocí něho provádí verifikaci podepsaných dat v tagu i načtení online dat. Kontrolovány jsou výjimky, které se mohou při dotazování na server objevit. Ty jsou odchyceny a převedeny do srozumitelné podoby uživateli.

Realizovaná aplikace obsahuje ještě některé grafické nedostatky, které můžeme do budoucna vyladit. Hlavním cílem bylo použití vytvořené knihovny aplikací, která ukazuje její možnou funkcionalitu. Ačkoliv jsme provedli zjednodušený návrh grafického rozhraní pomocí prototypování, není předmětem této části se nějak hlouběji zabývat použitelností a přístupností aplikace. Tuto část můžeme realizovat a zkoumat v budoucnu.

Kapitola 5

Závěr

Cílem této práce bylo především navrhnout a realizovat systém pro ověření původu zboží, který bude důvěryhodný a dostatečně zabezpečený. Výsledné řešení formou fungujícího prototypu dokazuje splnění stanovených cílů, komplexní popis návrhu a realizace daný systém dokumentuje. Zahrnuta jsou bezpečnostní kritéria nutná k tomu, aby systém mohl být všeobecně považován za důvěryhodný.

V současné době neexistuje mechanizmus, jímž jednoznačně a nepopiratelně určit původ produktu. Prvky používané pro označení zboží nenesou dostatek informací proto, aby bylo možné identifikovat výrobce a tím pádem odhalit nelegální kopii daného výrobku. Navržené řešení zavádí elektronickou identifikaci zboží spolu s vložením informace, která slouží k ověření jeho původu. Tato data jsou digitálně podepsána, aby bylo možné určit, že údaje nebyly modifikovány. První stupeň ověření tedy probíhá při čtení údajů z tagů produktů.

Pro spolehlivé ověření původu je zavedena druhá část kontroly pomocí systému komunikačního prostředníka, jemuž je vznesen ověřovací dotaz, který dále putuje k systému výrobce. Ten byl stanoven na základě vyčtených údajů z tagu produktu. Systém počítá s uložením veškerých údajů o produktech v systému výrobce, vlastní data neukládá. Zajištěna je tak důvěra výrobců, že poskytované údaje nemohou být zneužity např. v konkurenčním boji.

Protože některé typy produktů mohou vyžadovat evidenci stavu, je vytvořen způsob, jímž je možné stav produktu předávat od subjektu (distributora nebo obchodníka) k výrobci za pomoci důvěryhodné autority – komunikačního prostředníka. Počítá se s tím, že klientem takové služby je určitý informační systém, jenž s okolím komunikuje pomocí služeb založených na principu SOAP. Rozhraní komunikačního prostředníka tak poskytuje služby tohoto protokolu.

Součástí této práce bylo vytvoření ukázky informačního systému výrobce, který slouží k demonstraci uložení údajů o produktech. Obsahuje rozhraní, jenž komunikační prostředník konzumuje. Předpokládá se, že takový systém v praxi již existuje, pouze neobsahuje požadovanou funkcionalitu. Proto je komunikace založena na službách, jejichž rozhraní popisuje WSDL soubor. Vlastní část je věnována zabezpečení takové komunikace.

Mobilní klientská aplikace demonstruje klienta serverového řešení. Na službu komunikačního prostředníka se dotazuje pomocí REST rozhraní. Zabezpečení tohoto veřejně dostupného rozhraní je diskutováno v rámci bezpečnostní analýzy. Aplikace slouží k ověřování původu zboží zákazníkem popř. kontrolorem, který se o původ zboží zajímá při výkonu své

pracovní činnosti. Řešení umožňuje tomuto uživateli přidělit vyšší privilegia a zpřístupnit mu tak další potřebná data. Grafické rozhraní aplikace bylo navrženo jako prototyp nad knihovnou, která obsahuje implementaci všech potřebných součástí pro fungování klientské aplikace.

V celé práci je kladen velký důraz na bezpečnost navrhovaného a realizovaného řešení. Komunikační prostředník se totiž má stát důvěryhodnou autoritou, podobně jako certifikační autority. Aby mu všichni důvěřovali, musí ve všech oblastech splňovat bezpečnostní kritéria důvěryhodnosti, nezaměnitelnosti, nepopiratelnosti i autenticity. V rámci analýzy jsou tato kritéria rozebírána na úrovni samotných tagů produktů, aplikací a komunikace mezi nimi. Realizované řešení pak tato pravidla implementuje.

Přínosem této práce je vytvoření systému pro ověření původu zboží, který používá nejmodernějšího způsobu označování výrobků pomocí elektronických tagů k tomu, aby dokázal jednoznačně a nepopiratelně odlišit pravý a originální produkt výrobce od jeho padělku, kopie. Zákazník by tímto neměl být uveden v omyl, že kupovaný značkový produkt je ve skutečnosti méně či více zdařilým padělkem. Státní orgány pak mají možnost snáze tyto nepravé výrobky odhalit a dle příslušné legislativy dotčené subjekty postihovat.

Literatura

[1] Android Open Source Project. *Android Developers* [online]. [cit. 19. 11. 2013]. Dostupné z: http://developer.android.com/.

[2] Android Open Source Project. *Support Library – Android Developers* [online]. [cit. 2. 12. 2013]. Dostupné z: http://developer.android.com/tools/support-library/index.html.

[3] Apache Software Foundation. *Apache ANT* [online]. [cit. 27. 11. 2013]. Dostupné z: http://ant.apache.org/.

[4] Apache Software Foundation. *Apache Maven Project* [online]. [cit. 19. 11. 2013]. Dostupné z: http://maven.apache.org/.

[5] Appcelerator, Inc. *PyDev* [online]. [cit. 26. 11. 2013]. Dostupné z: http://www.pydev.org.

[6] Arnowitz, J. – Arent, M. – Berger, N. *Effective Prototyping for Software Makers.* Elsevier Science, 2010. ISBN 978-0-12-088568-8.

[7] Arskom Ltd. *Spyne* [online]. [cit. 4. 12. 2013]. Dostupné z: http://spyne.io/.

[8] Auto-ID Labs. *Auto-ID Labs website* [online]. [cit. 1. 10. 2013]. Dostupné z: http://www.autoidlabs.org/.

[9] Calderone, J.-P. *pyOpenSSL* [online]. [cit. 4. 12. 2013]. Dostupné z: https://launchpad.net/pyopenssl.

[10] Christie, T. *Django REST Framework* [online]. [cit. 4. 12. 2013]. Dostupné z: http://django-rest-framework.org/.

[11] Coskun, V. – Ok, K. – Ozdenizci, B. *Professional NFC Application Development for Android.* Wrox, 2013. ISBN 978-1118380093.

[12] Django Software Foundation. *Django Česká republika* [online]. [cit. 6. 10. 2013]. Dostupné z: http://www.djangoproject.cz.

[13] Dostálek, L. *Velký průvodce protokoly [TCP/IP]: bezpečnost.* Computer Press, 2003. ISBN 9788072268498.

[14] ECMA International. *JSON* [online]. [cit. 5. 10. 2013]. Dostupné z: http://www.json.org.

[15] Federální shromáždění. *Zákon č. 634/1992 Sb., o ochraně spotřebitele* [online]. Dostupné z: http://portal.gov.cz/app/zakony/zakonPar.jsp?idBiblio=40431.

[16] Feghhi, J. – Feghhi, J. – Williams, P. *Digital Certificates: Applied Internet Security.* Addison-Wesley, 1999. ISBN 978-0201309805.

[17] Fielding, R. T. *Architectural Styles and the Design of Network-based Software Architectures – CHAPTER 5: Representational State Transfer (REST).* PhD thesis, University of California, Irvine, 2000. Dostupné z: http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

[18] Free Software Foundation, Inc. *Make – GNU Project* [online]. [cit. 19. 11. 2013]. Dostupné z: http://www.gnu.org/software/make/.

[19] Goodger, D. *Docutils: Documentation Utilities* [online]. [cit. 25. 11. 2013]. Dostupné z: http://docutils.sourceforge.net/.

[20] Goodger, D. – Rossum, G. *PEP 257 – Docstring Conventions* [online]. [cit. 25. 11. 2013]. Dostupné z: http://www.python.org/dev/peps/pep-0257/.

[21] Google, Inc. *Android Tools Project Site* [online]. [cit. 26. 11. 2013]. Dostupné z: http://tools.android.com/.

[22] Google, Inc. *ADT Plugin* [online]. [cit. 26. 11. 2013]. Dostupné z: http://developer.android.com/tools/sdk/eclipse-adt.html.

[23] Google, Inc. *Google I/O 2013* [online]. [cit. 26. 11. 2013]. Dostupné z: https://developers.google.com/events/io/.

[24] GoPivotal, Inc. *Spring for Android* [online]. [cit. 2. 12. 2013]. Dostupné z: http://projects.spring.io/spring-android/.

[25] Gospodnetić, J. *jurko / suds* [online]. [cit. 4. 12. 2013]. Dostupné z: https://bitbucket.org/jurko/suds/.

[26] Gradleware, Inc. *Gradle – Build Automation Evolved* [online]. [cit. 19. 11. 2013]. Dostupné z: http://www.gradle.org/.

[27] GS1 AISBL. *GS1 – EPCglobal* [online]. [cit. 1. 10. 2013]. Dostupné z: http://www.gs1.org/epcglobal.

[28] GS1 AISBL. *GS1 – The global language of business* [online]. [cit. 1. 10. 2013]. Dostupné z: http://gs1.com/.

[29] GS1 AISBL. *GS1 Object Name Service (ONS) – Ratified Standard, Version 2.0.1* [online]. [cit. 1. 10. 2013]. Dostupné z: http://www.gs1.org/gsmp/kc/epcglobal/ons/ons_2_0_1-standard-20130131.pdf.

[30] Hönigsberg & Düvel Datentechnik Czech s.r.o. *Certifikace Lihovin – mobilní aplikace* [online]. [cit. 23. 9. 2013]. Dostupné z: https://play.google.com/store/apps/details?id=cz.hud.certifikace.lihovin.

[31] JetBrains, Inc. *IntelliJIDEA* [online]. [cit. 26. 11. 2013]. Dostupné z: http://www.jetbrains.com/idea/.

[32] JetBrains, Inc. *PyCharm* [online]. [cit. 26. 11. 2013]. Dostupné z: http://www.jetbrains.com/pycharm/.

[33] Kar, D. C. – Syed, M. R. *Network Security, Administration and Management: Advancing Technology and Practice.* IGI Global, 2011. ISBN 978-1609607777.

[34] Lundqvist, A. *GNU/Linux Distribution Timeline 12.10* [online]. [cit. 16. 10. 2013]. Dostupné z: http://futurist.se/gldt/.

[35] Microsoft, Inc. *The OSI Model's Seven Layers Defined and Functions Explained* [online]. [cit. 5. 11. 2013]. Dostupné z: http://support.microsoft.com/kb/103884.

[36] nearfield.cz. *Úplně všechno o NFC* [online]. [cit. 20. 9. 2013]. Dostupné z: http://nearfield.cz/.

[37] Network Working Group. *HTTP Authentication: Basic and Digest Access Authentication* [online]. [cit. 5. 10. 2013]. Dostupné z: http://www.ietf.org/rfc/rfc2617.txt.

[38] Nielsen, J. *Usability Engineering.* Morgan Kaufmann, 1993. ISBN 978-0125184069.

[39] Oat Systems & MIT Auto-ID Center. *The Object Name Service – Technical Manual*, v0.5 beta edition, 2001. Dostupné z: http://www.autoidlabs.org/single-view/dir/article/6/113/page.html.

[40] Oracle Corporation. *Code Conventions for the Java Programming Language* [online]. [cit. 18. 11. 2013]. Dostupné z: http://www.oracle.com/technetwork/java/codeconv-138413.html.

[41] Oracle Corporation. *Javadoc Tool* [online]. [cit. 18. 11. 2013]. Dostupné z: http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html.

[42] O'Reilly Media, Inc. *History of Programming Languages* [online]. [cit. 16. 10. 2013]. Dostupné z: http://oreilly.com/pub/a/oreilly/news/languageposter_0504.html.

[43] Parlament České republiky. *Zákon č. 102/2001 Sb., o obecné bezpečnosti výrobků a o změně některých zákonů (zákon o obecné bezpečnosti výrobků)* [online]. [cit. 21. 9. 2013]. Dostupné z: http://portal.gov.cz/app/zakony/zakonPar.jsp?idBiblio=51148.

[44] Parlament České republiky. *Zákon č. 22/1997 Sb., o technických požadavcích na výrobky a o změně a doplnění některých zákonů* [online]. [cit. 21. 9. 2013]. Dostupné z: http://portal.gov.cz/app/zakony/zakonPar.jsp?idBiblio=44944.

[45] Parlament České republiky. *Zákon č. 307/2013 Sb., o povinném značení lihu* [online]. [cit. 29. 11. 2013]. Dostupné z: http://portal.gov.cz/app/zakony/zakonInfo.jsp?idBiblio=80607.

[46] Parlament České republiky. *Zákon č. 378/2007 Sb., o léčivech a o změnách některých souvisejících zákonů (zákon o léčivech)* [online]. [cit. 21. 9. 2013]. Dostupné z: http://portal.gov.cz/app/zakony/zakonPar.jsp?idBiblio=65289.

[47] Parlament České republiky. *Zákon č. 676/2004 Sb., o povinném značení lihu a o změně zákona č. 586/1992 Sb., o daních z příjmů, ve znění pozdějších předpisů* [online]. [cit. 21. 9. 2013]. Dostupné z: http://portal.gov.cz/app/zakony/zakonPar.jsp?idBiblio=58698.

[48] Pecinovský, R. *Návrhové vzory – 33 vzorových postupů pro objektové programování.* Computer Press, 2007. ISBN 9788025115824.

[49] Peterka, J. *Báječný svět elektronického podpisu.* CZ.NIC, z.s.p.o., 2011. Dostupné z: http://bajecnysvet.cz. ISBN 978-80-904248-3-8.

[50] Piedad, F. – Hawkins, M. W. *High Availability: Design, Techniques and Processes.* Prentice Hall, 2000. ISBN 978-0130962881.

[51] Redaktoři MAFRA, a.s. *iDNES.cz – Rubrika Metanol* [online]. [cit. 21. 9. 2013]. Dostupné z: http://zpravy.idnes.cz/metanol-c51-/archiv.aspx?klic=481068.

[52] Richardson, L. – Ruby, S. *RESTful Web Services.* O'Reilly Media, 2007. ISBN 978-0-596-52926-0.

[53] Schneider, B. *Applied Cryptography: Protocols, Algorithms and Source Code in C.* John Wiley & Sons, 2nd Edition, 1995. ISBN 978-0471117094.

[54] Security-Portal.cz. *Jak funguje IPSec?* [online]. [cit. 15. 10. 2013]. Dostupné z: http://www.security-portal.cz/clanky/jak-funguje-ipsec.

[55] Stock Plzeň – Božkov s.r.o. *Zkontroluj si láhev – Pij bezpečně* [online]. [cit. 23. 9. 2013]. Dostupné z: http://www.pijbezpecne.cz/.

[56] The Eclipse Foundation. *Eclipse* [online]. [cit. 26. 11. 2013]. Dostupné z: http://eclipse.org/.

[57] The RFID Network. *What is RFID? Radio Frequency Identification* [online]. [cit. 20. 9. 2013]. Dostupné z: http://rfid.net/basics.

[58] Unie výrobců a dovozců lihovin. *CertifikaceLihovin.cz* [online]. [cit. 23. 9. 2013]. Dostupné z: http://www.certifikacelihovin.cz/.

[59] Rossum, G. – Warsaw, B. – Coghlan, N. *PEP 8 – Style Guide for Python Code* [online]. [cit. 25. 11. 2013]. Dostupné z: http://www.python.org/dev/peps/pep-0008/.

[60] Vincoli, J. W. *Basic Guide to System Safety.* Wiley-Interscience, 2nd edition, 2006. ISBN 978-0471722410.

[61] Vláda České republiky. *Nařízení vlády č. 317/2012* [online]. [cit. 21. 9. 2013]. Dostupné z: `http://www.vlada.cz/assets/media-centrum/aktualne/Narizeni-vlady-26092012---formular-puvodu-lihu.pdf`.

[62] W3C. *SOAP Specification* [online]. [cit. 5. 10. 2013]. Dostupné z: `http://www.w3.org/TR/soap12/`.

[63] W3C. *Web Services Description Language (WSDL) 1.1* [online]. [cit. 12. 11. 2013]. Dostupné z: `http://www.w3.org/TR/wsdl/`.

[64] Čmelík, M. *Seznamte se – DoS a DDoS útoky* [online]. [cit. 5. 11. 2013]. Dostupné z: `http://www.security-portal.cz/clanky/seznamte-se--dos-ddos-útoky`.

Příloha A

Seznam použitých zkratek

ACL Access Control List (seznam pro řízení přístupu).

ADT Android Development Tools (vývojové nástroje pro Android).

API Application Programming Interface (rozhraní pro programování aplikací).

CRC Cyclic Redundancy Check (cyklický redundantní součet).

CRL Certificate Revocation List (seznam revokovaných certifikátů).

DDoS Distributed Denial of Service (distribuovaný DoS).

DHCP Dynamic Host Configuration Protocol (protokol pro automatickou konfiguraci počítačů připojených do počítačové sítě).

DNS Domain Name System (systém doménových jmen).

DoS Denial of Service (odmítnutí služby).

EAN European Article Number (evropský zbožový kód).

EPC Eletronic Product Code (elektronický kód produktu).

GNU GNU's Not Unix! (GNU Není Unix!).

HTTP Hypertext Transfer Protocol (internetový protokol určený pro výměnu hypertextových dokumentů).

HTTPS Hypertext Transfer Protocol Secure (zabezpečený internetový protokol určený pro výměnu hypertextových dokumentů).

IDE Integrated Development Environment (vývojové prostředí).

IP Internet Protocol (protokol síťové vrstvy).

IS Information system (informační systém).

IČ Identifikační číslo podnikatelského subjektu v Česku.

JDK Java Development Kit (vývojový kit pro jazyk Java).

JIT Just In Time (právě včas).

JRE Java Runtime Environment (běhové prostředí pro jazyk Java).

JSON JavaScript Object Notation (objektový zápis jazyka JavaScript).

JVM Java Virtual Machine (virtuální stroj pro spouštění Java aplikací).

LDAP Lightweight Directory Access Protocol (protokol pro ukládání a přístup k datům na adresářovém serveru).

LTS Long Term Support (dlouhodobá podpora).

MD5 Message-Digest v.5 (hašovací funkce).

MIT Massachusetts Institute of Technology (Massachusettský technologický institut).

NFC Near Field Communication (komunikace na krátkou vzdálenost).

ONS Object Name Service (služba pro pojmenování objektů).

ORM Object-Relational Mapping (objektově relační mapování).

OS Operating System (operační systém).

PIN Personal Identification Number (osobní identifikační číslo).

QR Quick Response (rychlá odpověď).

REST Representational State Transfer (architektonický styl pro definici a adresaci zdrojů pomocí protokolu HTTP).

RFID Radio Frequency Identification (identifikace pomocí radiové frekvence).

RMI Remote Method Invocation (vzdálené spouštění metod).

RPC Remote Procedure Call (vzdálené volání procedur).

SD Secure Digital (paměťová karta).

SDK Software Development Kit (kit pro vývoj softwaru).

SHA Secure Hash Algorithm (zabezpečný hašovací algoritmus).

SOAP Simple Object Access Protocol (jednoduchý protokol pro přístup k objektům).

SSH Secure Shell (zabezpečený komunikační protokol v počítačové síti).

SSL Secure Sockets Layer (vrstva bezpečných socketů).

TCP/IP Transmission Control Protocol/Internet Protocol (primární přenosový protokol, protokol síťové vrstvy).

URL Uniform Resource Locator (jednotný lokátor zdrojů).

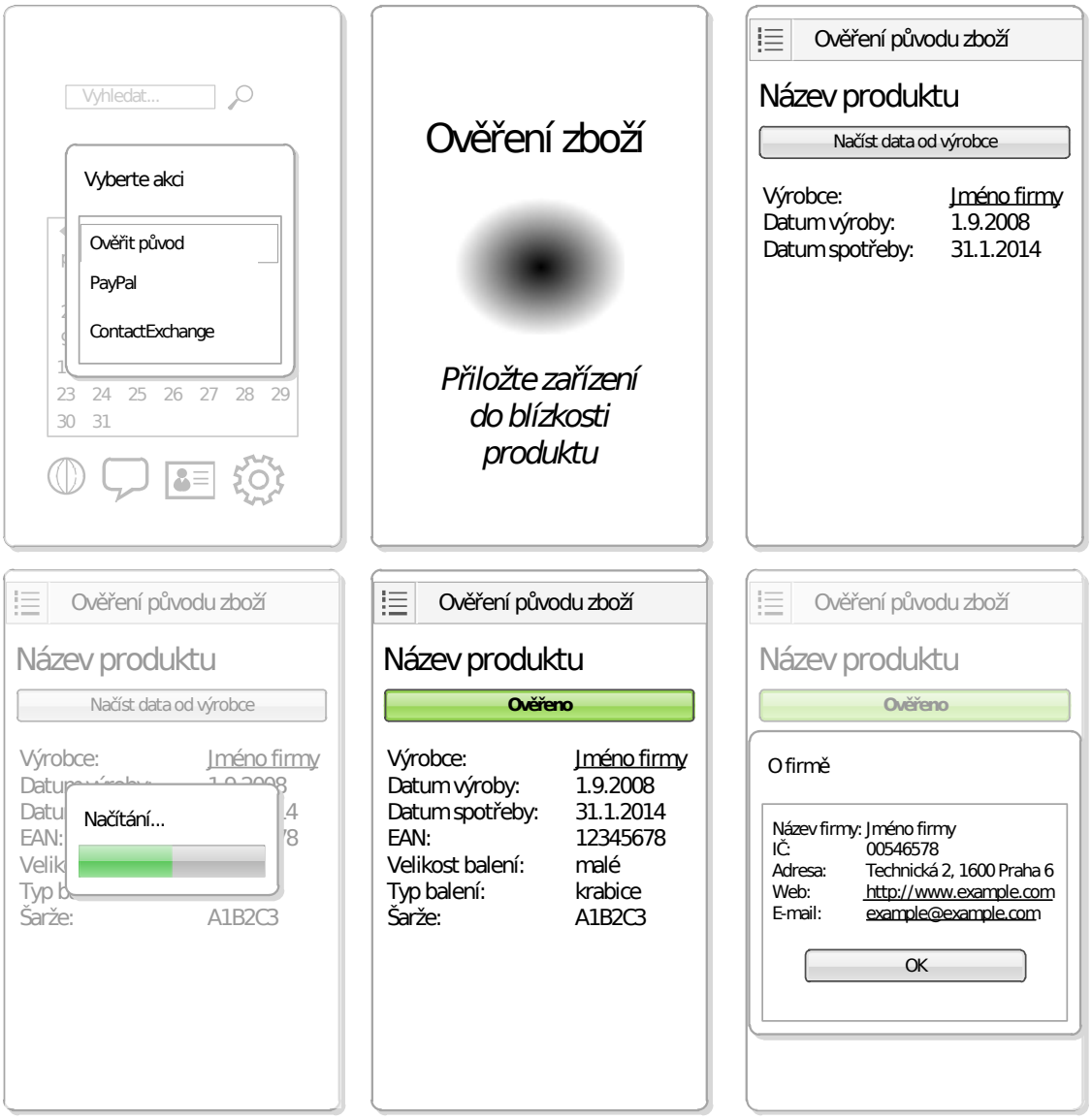
VPN Virtual Private Network (virtuální privátní síť).

WSDL Web Services Description Language (jazyk pro popis webové služby).

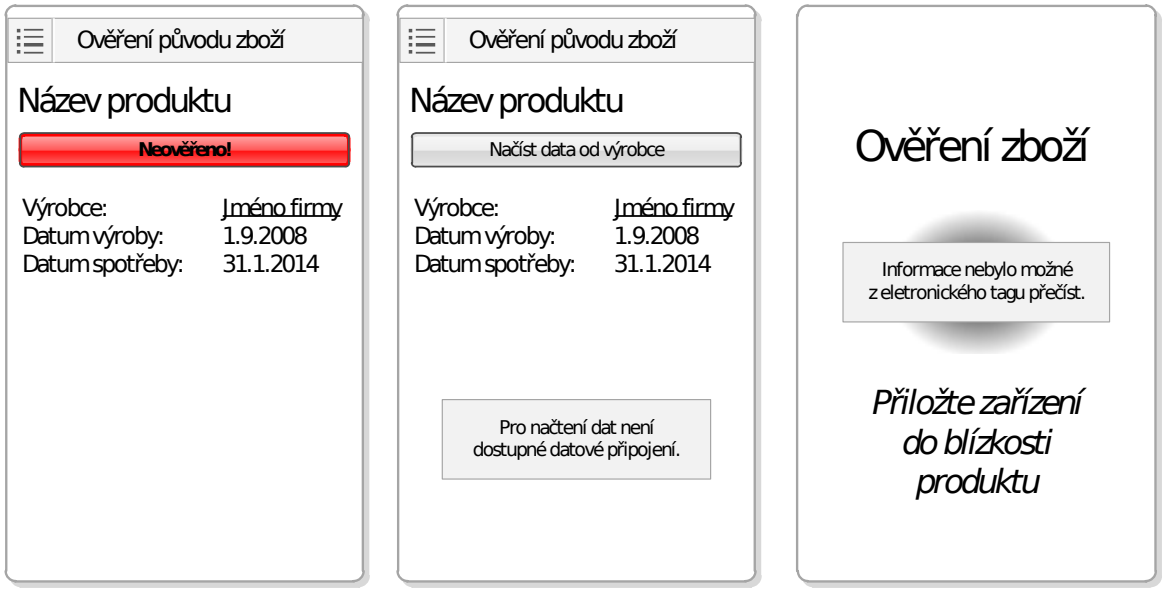
XML Extensible Markup Language (rozšiřitelný značkovací jazyk).

Příloha B

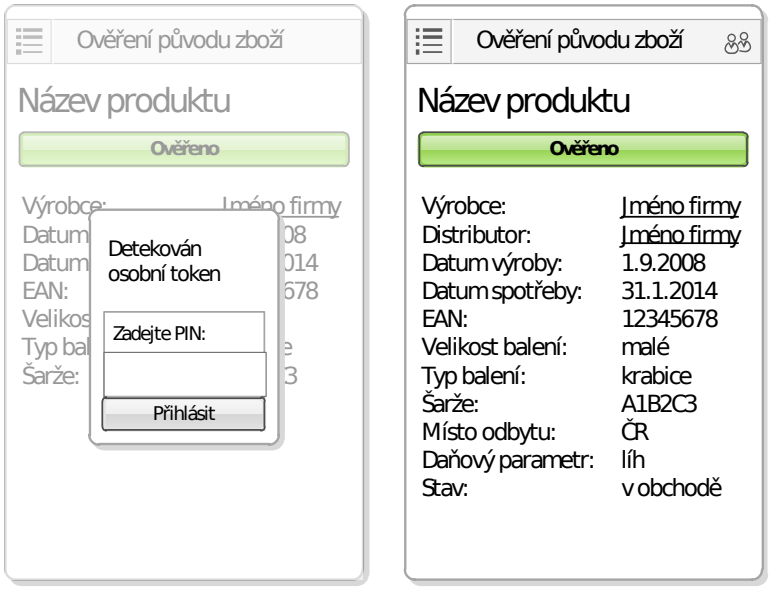
Lo-fi prototyp mobilní aplikace



Obrázek B.1: Wireframes pro low-fidelity prototyp mobilního klienta



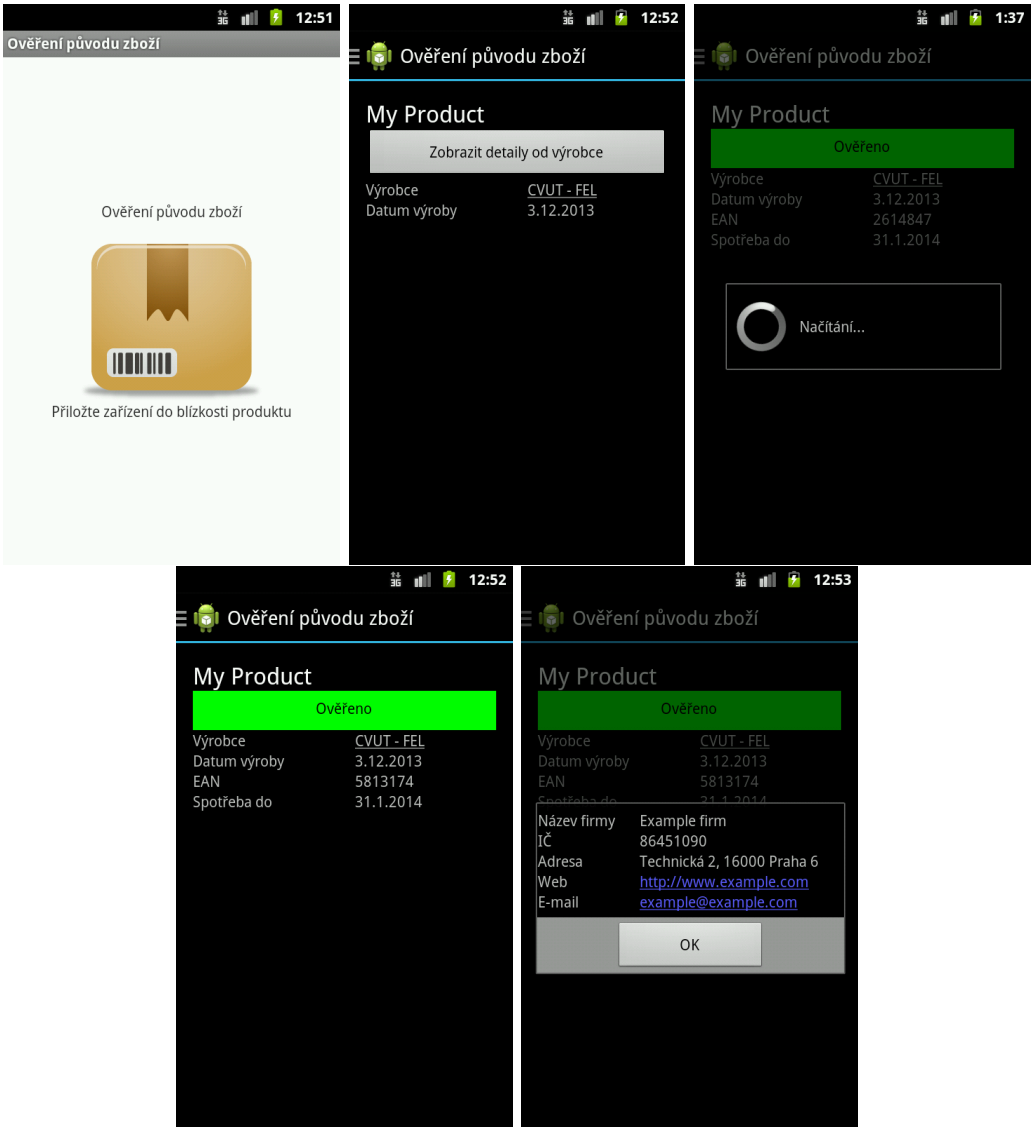
Obrázek B.2: Wireframes pro low-fidelity prototyp mobilního klienta – nezdařené operace



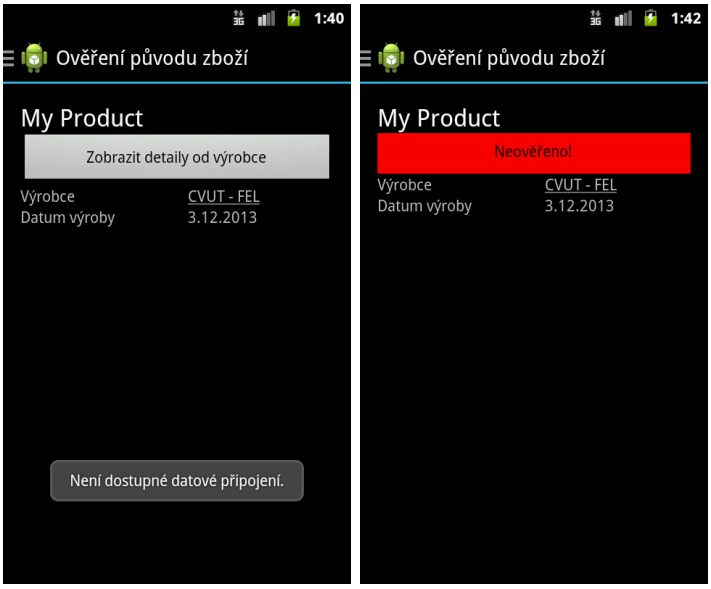
Obrázek B.3: Wireframes pro low-fidelity prototyp mobilního klienta – privilegovaný uživatel

Příloha C

Hi-fi prototyp mobilní aplikace



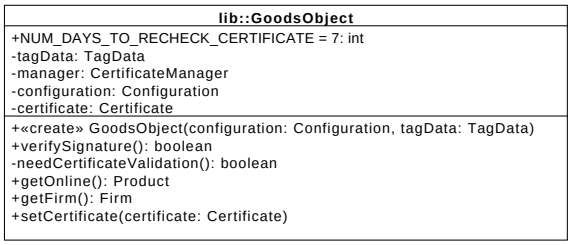
Obrázek C.1: Obrazovky z high-fidelity prototypu mobilního klienta



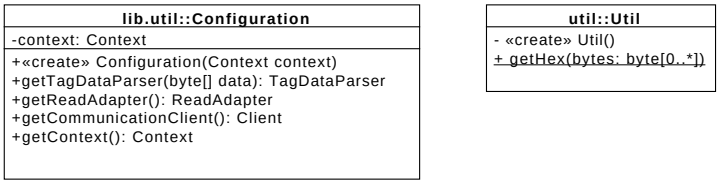
Obrázek C.2: Obrazovky z high-fidelity prototypu mobilního klienta – nezdařené operace

Příloha D

Diagramy tříd mobilního klienta

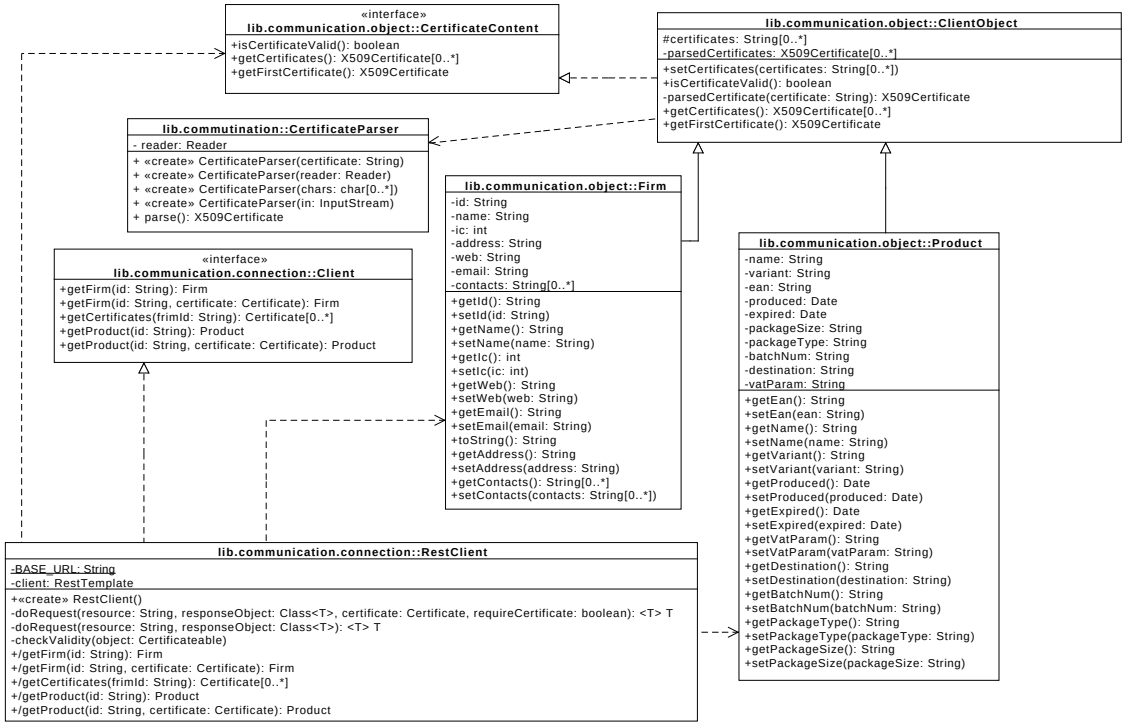


Obrázek D.1: Diagram obecné třídy v knihovně mobilního klienta

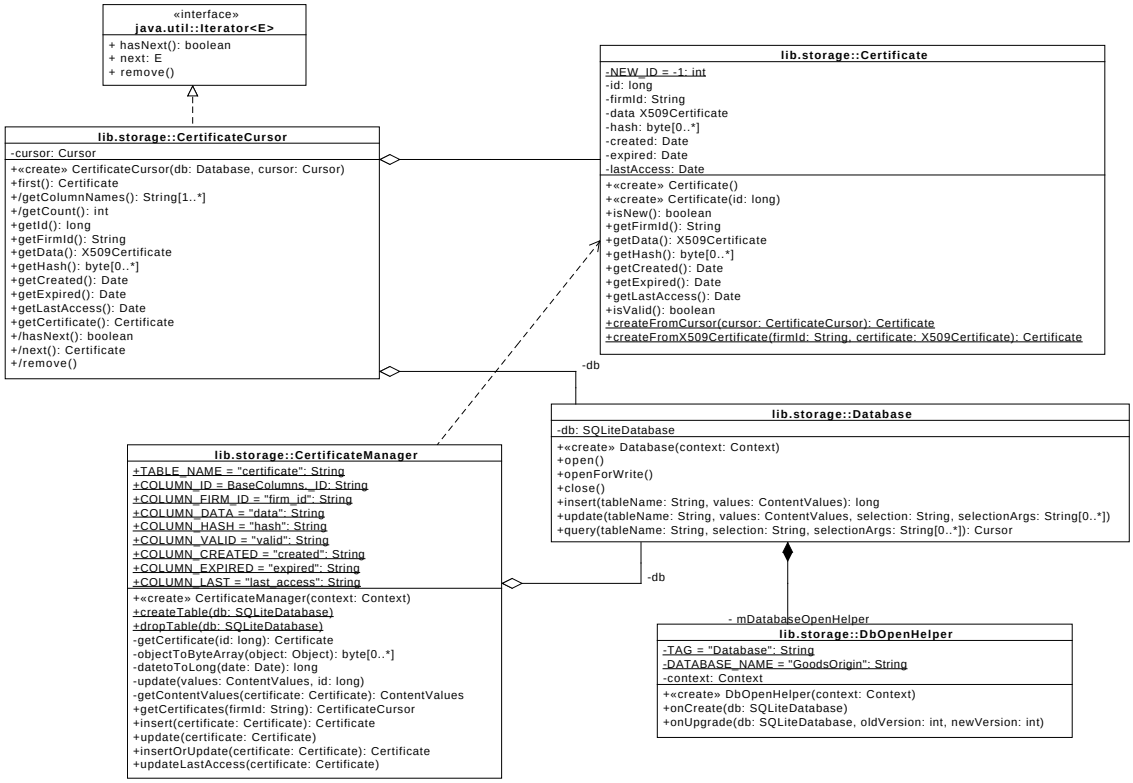


Obrázek D.2: Diagram podpůrných tříd v knihovně mobilního klienta

Obrázek D.3: Diagram třídy pro ověření podpisu v knihovně mobilního klienta



Obrázek D.4: Diagram tříd pro komunikaci se serverem v knihovně mobilního klienta



Obrázek D.5: Diagram tříd pro uložení certifikátů v knihovně mobilního klienta

Příloha E

Obsah přiloženého CD

Nedílnou součástí této práce je i přiložené CD se zdrojovými kódy všech aplikací v systému obsažených. Součástí zdrojových kódů jsou také nastavení projektů pro vývojové prostředí Android Studio a nakonec také uvedený PyCharm (viz 4.2.2), obojí pro okamžité otevření projektu a pokračování v práci.

data zdrojové soubory obrázků a nákresů uvedených v práci a realizovaných v softwarech Microsoft Visio 2010, Umlet a Inkscape

doc dokumentace k projektu

- **sql** vygenerovaná struktura databáze a ukázková data použitá při návrhu a testování prototypu systému
- **src** vygenerovaná dokumentace pro všechny tři součásti systému, jak z Java projektu, tak z aplikací v jazyce Python

source zdrojové kódy všech tří aplikací vytvořených v rámci realizace projektu

- **client** mobilní klientská aplikace pro OS Android
- **dwarf** aplikace komunikačního prostředníka
- **manufacturer** ukázkový informační systém výrobce
- **subject** jednoduchý skript pro testování aktualizace produktu

text zdrojový text této práce v jazyce LaTeX (včetně vlastních maker k tomu vytvořených)