FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Centrální správa počítačů v Kiosk módu
Student:	Jan Klička
Vedoucí:	Ing. Jiří Mlejnek
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce je analyzovat požadavky, navrhnout a implementovat systém hromadné správy Raspberry PI zařízení (kiosek), která budou zobrazovat webové stránky. Systém umožní vzdáleně kiosky nastavovat (která webová stránka má být zobrazena) a spravovat (například zpřístupnit ssh).

Proveďte implementaci centrálního serveru umožňující provádět hromadnou správu zařízení a implementaci aplikace pro kiosek, která umožní zaregistrování zařízení na centrální server a následné hromadné nastavování a správu.
Výsledné řešení důkladně otestujte a připravte pro produkční nasazení.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 15. prosince 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Centrální správa počítačů v Kiosk módu

Jan Klička

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Mlejnek

13. května 2021

Poděkování

Rád bych poděkoval mému vedoucímu práce, Ing. Jiřímu Mlejnkovi, za jeho čas a rady při jejím vypracování. Dále bych také rád poděkoval Jakubovi Petrovi za poskytnutí zadání práce a podporu při jejím vypracování. V neposlední řadě bych chtěl poděkovat rodině a přátelům za jejich podporu.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. května 2021 . . . . . .

Odkaz na tuto práci

Klička, Jan. *Centrální správa počítačů v Kiosk módu.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato práce se zabývá vývojem serveru pro správu kiosků, což jsou Raspberry Pi zařízení zobrazující webovou stránku. Dalším cílem je vytvořit proces efektivní instalace těchto kiosků. Server je napsán v programovacím jazyce Java s použitím frameworku Spring, pomocí kterého poskytuje REST API. Raspberry Pi používá Raspberry Pi OS, který je pro potřeby práce upraven během jeho instalace na SD kartu. Výstupem práce je systém připravený pro produkční nasazení.

Klíčová slova kiosek, REST API, Java, Spring, Raspberry Pi

Abstract

This thesis deals with development of a server for managing kiosks, which are Raspberry Pi devices showing a webpage. Another goal is to create an effective installation process of these kiosks. The server is written in Java programming language using framework Spring to expose REST API. Raspberry Pi runs on Raspberry Pi OS, which is modified during its installation on SD card. The output of this thesis is a system ready for production deployment.

Keywords kiosk, REST API, Java, Spring, Raspberry Pi

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Efektivita práce v továrnách je vždy aktuálním tématem. Díky zvýšené efektivitě se zvedá produkce, snižují se náklady a podnik více prosperuje. Jeden způsob jak zvýšit efektivitu práce jsou výrobní obrazovky zobrazující například kapacitu výroby, upozornění, nebo z nich lze v informačním systému dokončit rozpracované úkoly a výrobky, nahlásit neshody ve výrobě atp. Tyto obrazovky umožňují komunikaci s informačním systémem přímo z pracoviště u stroje bez nutnosti odbíhat a také ulehčí komunikaci mezi pracovníky, kteří mají informace živě a nemusí si je mezi sebou předávat ústně či papírově.

Současná řešení jsou buď drahá - průmyslové all-in-one počítače - nebo levná a nespolehlivá - Pipo počítače s nekvalitním dotykovým displejem s obtížnou manipulací. Alternativní cenově dostupné řešení je Raspberry Pi, levný a spolehlivý mini počítač, spárovaný s dotykovou obrazovkou. Tento kiosek lze využít pro potřeby výrobních obrazovek. Cílem práce je vytvořit proces efektivní přípravy těchto kiosků, analýza, návrh, implementace a testování serveru pro jejich správu. Výstup této práce bude prakticky využitelný ve výrobním prostředí, kde zmenší náklady na hardware, což jsou hlavní důvody, proč jsem si zvolil toto téma.

Práci řeším ve spolupráci s firmou Factorify, jejíž hlavním produktem je stejnojmenný informační systém na řízení výroby. Díky této spolupráci budeme schopni výstup mé bakalářské práce otestovat v praxi, iterovat na něm a dostat ho do podoby, aby výsledný přínos byl co největší.

V první kapitole se věnuji cílům práce. Ve druhé kapitole popisuji současná řešení. Ve třetí kapitole se věnuji analýze požadavků. Ve čtvrté kapitole se zabývám návrhem aplikace, které technologie a architekturu jsem použil. V páté kapitole popisuji implementaci zajímavých částí řešení. V šesté kapitole se věnuji testování. V sedmé kapitole popisuji možnosti dalšího vývoje. V závěru hodnotím míru splnění cílů.

Kapitola 1

Cíl práce

Cílem této bakalářské práce je vytvořit systém pro správu tzv. kiosků, což jsou Raspberry Pi zařízení, jejichž účelem je zobrazovat danou webovou stránku. Tento systém obsahuje server dostupný přes REST API a bude napsán v jazyce Java s použitím Spring knihovny.

Systém dále obsahuje proces pro rychlou a jednoduchou přípravu těchto kiosků. Kiosek běží na linuxové distribuci Raspberry Pi OS, což je upravená distribuce Debian pro Raspberry Pi.

Tyto kiosky budou umístěny v továrně, aby zobrazovaly potřebné informace pro výrobu a umožňovaly komunikaci s informačním systémem továrny.

Mezi dílčí cíle této práce patří analýza požadavků, rešerše existujících řešení, návrh, implementace a testování navrhovaného řešení a také příprava pro produkční nasazení.

Kapitola 2

Rešerše existujících řešení

S kiosky se potkáváme roky na místech, jakými jsou například nemocnice, školy, obchodní centra, turistické atrakce atp. Proto se v této kapitole budu zabývat nabízenými řešeními na trhu.

Tato řešení mohou být pouze softwarové nebo mohou zahrnovat kiosky jako takové. Hlavní kritéria pro hodnocení jsou cena, možnost spravovat více zařízení, podpora pro Linux a ARM architekturu, případně jestli řešení poskytuje vlastní hardware.

2.1 Autocont IPC a.s.

Autocont vyrábí a dodává průmyslové počítače, kiosky a další průmyslové systémy. Kiosky nabízejí v různých provedeních: bez/s klávesnicí, s různými čtečkami a nebo s dotykovými obrazovkami [1]. Konkrétní ceny hardwarových kiosků bez registrace nelze zjistit, ale ze zkušenosti od klientů ve Factorify víme, že ceny kiosků od různých výrobců se pohybují v částkách zhruba 50 000 Kč za kus [2], což je jejich hlavní nevýhodou.

Ke kioskům dodávají řešení od Ki-Wi Digital umožňující vzdálenou správu, přehrávání videí, plánování obsahu, analýzu využití a další. [3]

2.2 Mobile Device Manager

ManageEngine se zaměřuje na řešení problémů z IT oblastí, např. bezpečnost, správa zařízení, datová analýza a mnoho dalších. [4]

Mobile Device Manager Plus je jedním z jejich produktů sloužících pro spravování Windows a mobilních zařízení. Umožňuje dálkově instalovat aplikace, zabezpečit zařízení a data, sdílet data mezi spravovanými zařízeními, pustit zařízení v kiosk módu a další. Podporované platformy jsou iOS, Android, Windows a ChromeOS [5].

Ačkoliv tento produkt poskytuje hodně funkcionalit, chybějící podpora pro linuxová zařízení je jeho hlavní nevýhodou.

2.3 KioWare Kiosk Management

KioWare je společnost zaměřující se na vývoj softwaru KioWare Kiosk Management pro správu kiosků. Mezi hlavní vlastnosti patří správa zařízení (aktualizace, přenos souborů, restart atp.), statistika využití a monitoring zařízení (využití CPU, logování). Podporují pouze Android a Windows [6].

Stejně jako u Mobile Device Manageru, hlavní nevýhodou zůstává absence podpory pro Linux.

2.4 Porteus Kiosk

Porteus Kiosk je výkonostně nenáročný bezplatný linuxový operační systém omezený pouze na používání webového prohlížeče. K dispozici je také server pro správu zařízení. Tento server je koncipován také jako operační systém, tudíž je potřeba extra počítače na jeho běh nebo využití virtualizace. Porteus nepodoporuje architekturu ARM, na které běží Raspberry Pi. [7]

2.5 Pi Kiosk

Pi kiosk je malý a jednoduchý open source projekt, který pomocí nástroje Ansible spravuje Raspberry Pi v kiosk módu. Ansible slouží k automatizaci nasazení aplikace, správy zařízení a dalších IT potřeb [8]. Projekt sestává z několika Ansible Playbooks, což jsou soubory definující, co má Ansible provést. Díky Ansible je proces jednoduchý a opakovatelný, a na Raspberry Pi není třeba žádného klienta.

Projekt není připraven pro hromadnou správu, jelikož seznam kiosků je udržován v textovém souboru společně s jejich IP adresami, což je hlavní nevýhoda tohoto projektu. Tento přístup vyžaduje statické IP adresy pro jednotlivá zařízení, což přidává nárok na infrastrukturu [9].

2.6 Zhodnocení

Podle zmiňovaných kritérií by ideální řešení bylo bezplatné, připravené hromadně spravovat více zařízení a podporovalo by Linux na ARM architektuře. Bohužel, ani jedno z porovnávaných zařízení nesplňuje všechny požadavky. Nejvíce se těmto požadavkům blíží Pi Kiosk, u kterého správa více zařízení není vhodná. V následující tabulce jsou řešení srovnána.

	Bezplatné	Správa více zařízení	Linux / ARM	Poskytuje HW
Autocont	✗	✓		✓
Mobile Device Manager	✗	✓	✗	✗
KioWare Kiosk	✗	✓	✗	✗
Porteus Kiosk	✓	✓	✓ / ✗	✗
Pi Kiosk	✓	?	✓ / ✓	✗

Tabulka 2.1: Srovnání analyzovaných řešení

Kapitola 3

Analýza

V této kapitole se zabývám tím, co by systém měl obsahovat. Kapitola obsahuje doménový model, případy užití a nefunkční požadavky. Kapitola neobsahuje funkční požadavky, jelikož jsou popsány případy užití.

3.1 Doménový model

Doménový model je zachycen na diagramu 3.1. Administrátor spravuje kiosky prostřednictvím serveru, který udržuje spojení s kiosky. Pracovníci interagují s webovou stránkou, kterou kiosek zobrazuje. Pokud pracovníci mají problém s kioskem, ozvou se administrátorovi, aby kiosek opravil. Kiosek se pravidelně připojuje k serveru, díky čemuž je server schopný rozlišit aktivní a neaktivní kiosky.

3.2 Nefunkční Požadavky

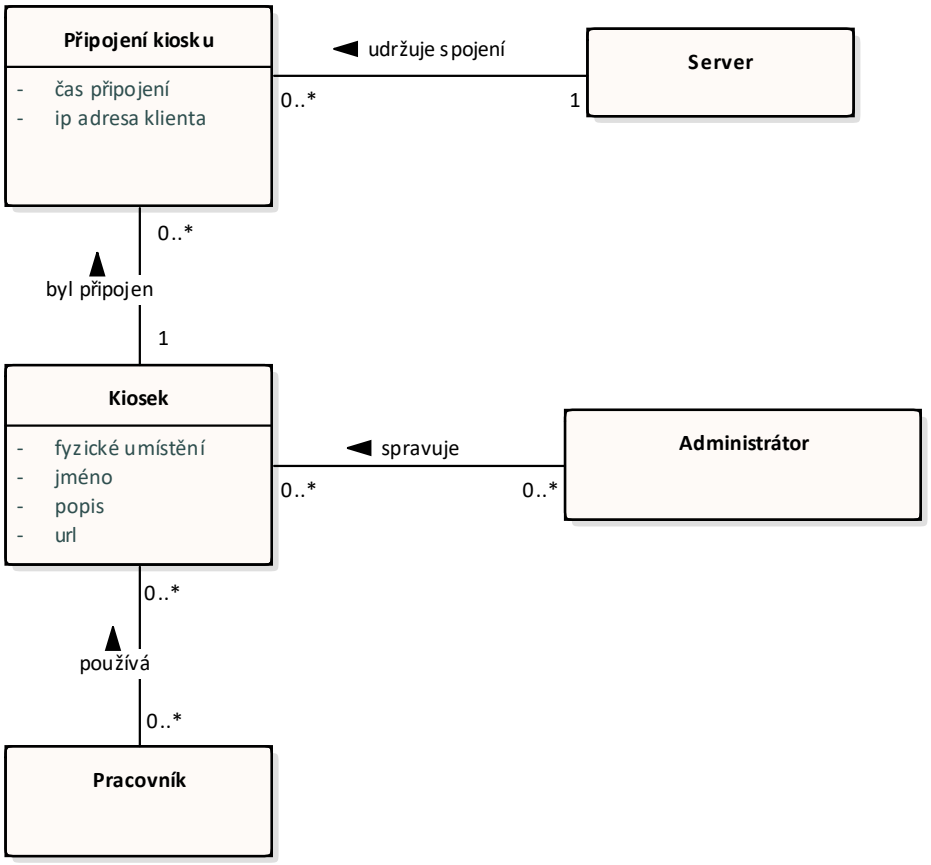
„Nefunkční požadavky popisují vlastnosti systému jako např. výkon nebo zabezpečení. Dále také mohou klást nároky na design nebo rozhraní systému s vnějším světem.“ [10]

N1: Programovací jazyk Server bude napsán v jazyce Java.

N2: Rozhraní a technologie Server bude využívat framework Spring pro poskytnutí REST API. Kiosek bude využívat toto rozhraní ke splnění požadovaných funkcionalit.

N3: Hardware Kiosek bude Raspberry Pi zařízení.

N4: Zabezpečení Práce neřeší autentizaci kiosků nebo uživatelů, protože tento systém je koncipován pro vnitřní sítě, do kterých se nelze dostat z internetu.



Obrázek 3.1: Doménový model

3.3 Případy užití

Případy užití (UC) se používají pro určení požadovaného chování systému ve spolupráci s aktérem (uživatelem systému nebo jiným systémem). Případ užití specifikuje jednu užitečnou funkcionalitu systému, která je nabízena aktérům [11].

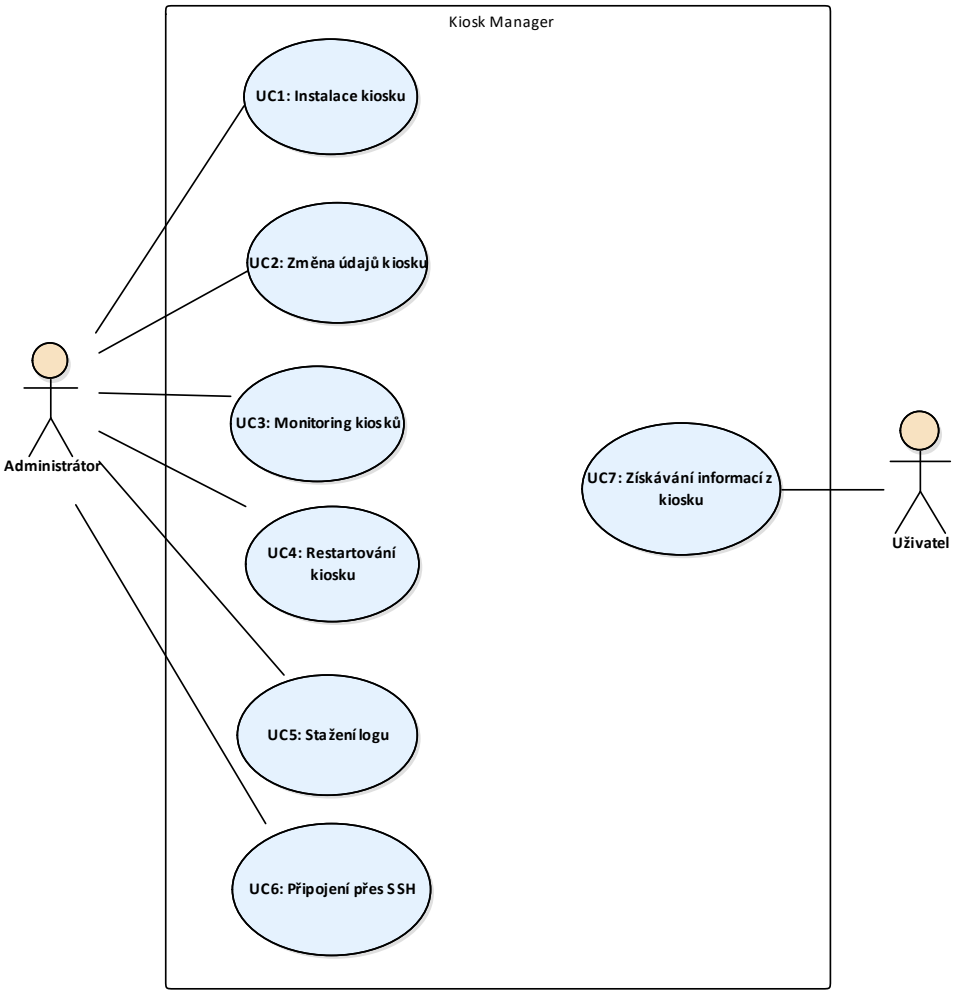
Aktéři v tomto systému jsou:

administrátor, který se zabývá správou a instalací kiosků,

pracovník, který interaguje s webovou stránkou zobrazovanou kioskem.

Případy užití lze vidět na diagramu 3.2 a textově jsou popsány níže.

UC1: Instalace kiosku Administrátor musí připravit samotné zařízení a zapojit kiosek na místě určení. Zařízení by se následně mělo samo zaregis-



Obrázek 3.2: Diagram případů užití

trovat na server, kde je zapotřebí nastavit informace o zařízení, primárně jaká webová stránka se má zobrazit. Kiosek se rozběhne do 15 minut. Během instalace lze zavést SSH přístup na zařízení.

Pokud to bude možné, webová stránka by měla jít nastavit už během přípravy zařízení, aby administrátor nemusel navštívit systém po prvním spuštěním kiosku.

Hlavní scénář je následující:

1. Administrátor fyzicky připraví zařízení, kterému určí jméno a volitelně webovou stránku pro kiosek.
2. Administrátor spustí kiosek.
3. Kiosek se zaregistruje na server pod zvoleným jménem.
   - Pokud administrátor nezvolil webovou stránku, musí po tomto kroku na serveru zvolit webovou stránku.
   - Pokud administrátor zvolil webovou stránku, uloží se na server během registrace.
4. Kiosek zobrazí zvolenou stránku.

Scénář je také popsán diagramem aktivit 3.3.

UC2: Změna údajů kiosku Administrátor změní informace v systému o kiosku. Pokud změní URL kiosku, při dalším spojení kiosku a serveru si kiosek změní zobrazovanou stránku.

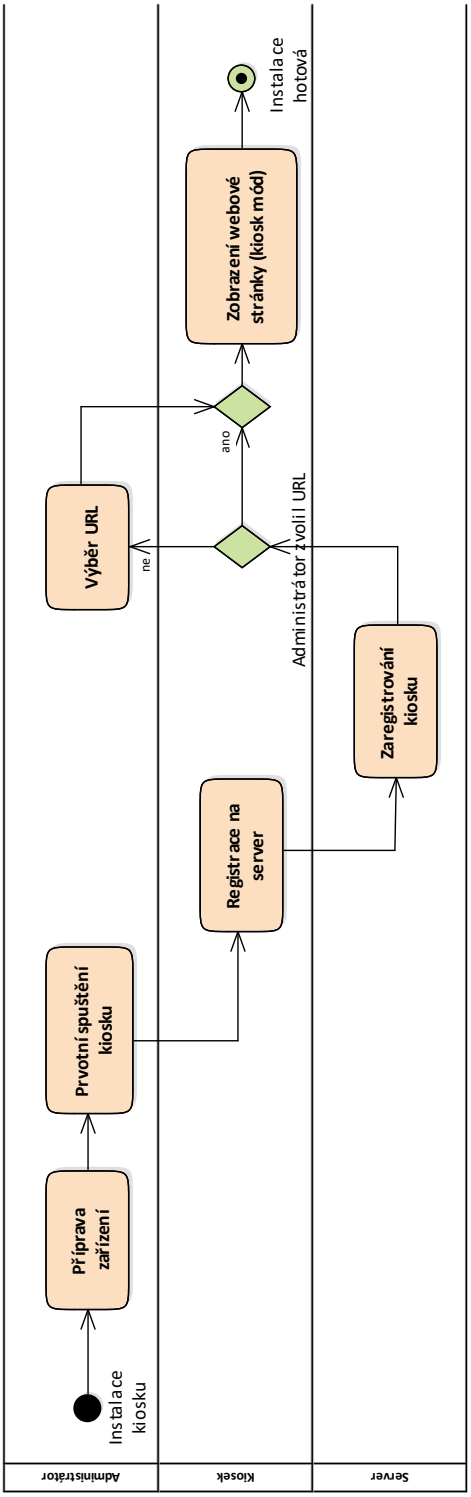
UC3: Monitoring kiosků Systém udržuje informace o tom, které kiosky se k němu připojily. Systém také s nimi pravidelně udržuje spojení a udržuje informace jakými jsou např. čas posledního spojení a poslední IP adresa, ze které se kiosek připojil. Díky tomu lze rozlišit mezi aktivními a neaktivními kiosky.

UC4: Restartování kiosku Administrátor na serveru vyžádá restartování kiosku. Po dalším spojení kiosku a serveru se kiosek restartuje.

UC5: Stažení logu Administrátor na serveru vyžádá stažení webového nebo systémového logu. Po dalším spojení kiosku a serveru kiosek nahraje zvolený log na server. Administrátor následně si stáhne obsah logu ze serveru.

UC6: Připojení přes SSH Administrátor si ze serveru zjistí poslední IP adresu kiosku, pomocí které se připojí na zařízení přes SSH.

UC7: Získávání informací z kiosku Kiosek zobrazuje zvolenou webovou stránku. Pracovník využije kiosek, když potřebuje interagovat s webovou stránku.



Obrázek 3.3: Instalace kiosku

Kapitola 4

Návrh

V této kapitole se zabývám návrhem systému, jeho fyzickou strukturou a vybranými technologiemi.

4.1 Architektura Java aplikace

Dle [12] se pro servery často používá tzv. třívrstvá architektura:

Prezentační vrstva obstarává komunikaci mezi uživatelem a aplikací.

Business vrstva má na starost logickou/business část aplikace, kde se vykonává hlavní logika.

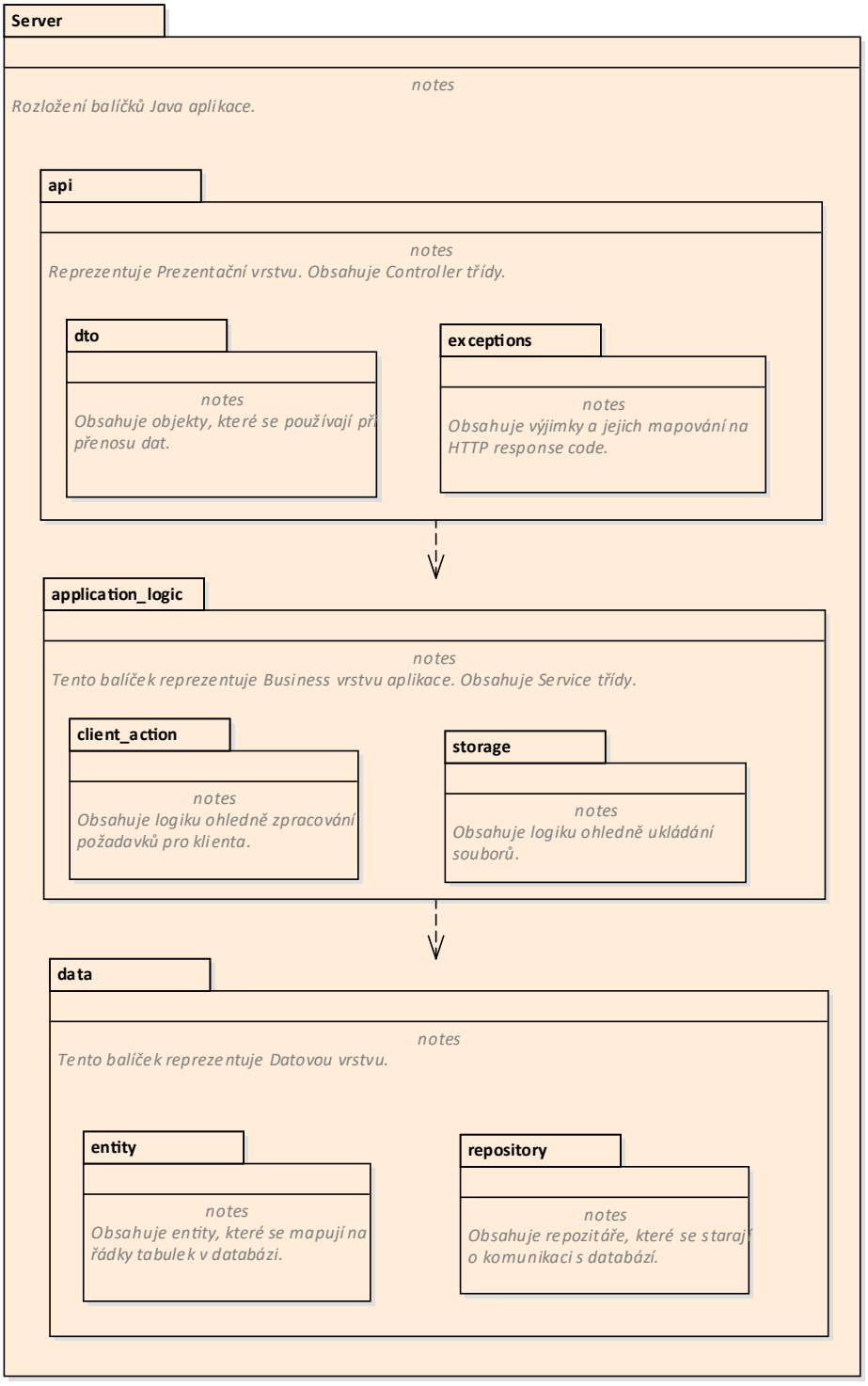
Datová vrstva slouží ke komunikaci s jinými systémy, primárně obstarává perzistenci dat v databázi.

Vrstvení systému má řadu výhod, jednou z nich je rozdělení složitého systému na menší části, které postupně na sobě staví. Tzn. že aplikační vrstva je závislá na datové, ale datová vrstva neví nic o aplikační. Díky tomu jsou odpovědnosti vrstev jasně definované a lze pochopit funkčnost jedné vrstvy bez znalosti ostatních [12].

Rozdělení balíčků implementace je popsáno textově v následujících kapitolách a nebo diagramem 4.1.

4.1.1 Prezentační vrstva

Do prezentační vrstvy patří REST rozhraní aplikace, protože se stará o předání zpráv uživatelů. Tato vrstva je reprezentována balíčkem api, ve kterém se nachází Controller třídy. Dále také obsahuje balíček dto, ve kterém se nacházejí objekty používané k transportu dat, a balíček exceptions, ve kterém jsou speciální výjimky, které při vyvolání Spring namapuje na zvolený HTTP response code. Klient do této vrstvy nepatří, protože se dá považovat za jinou aplikaci, která využívá toto rozhraní.



Obrázek 4.1: Model balíčků aplikace

4.1.2 Business vrstva

V této vrstvě se nachází aplikační logika, validace dat, ukládání souborů atd. Primárně se jedná o třídy anotované @Service. Tyto třídy se nachází v balíčku application_logic, který je rozdělen na 2 podbalíčky - client_action, který obsahuje logiku ohledně zpracování požadavků pro klienta, a storage, který se stará o ukládání souborů na disk.

Složitější služby, které neslouží jako proxy pro repozitáře, implementují Interface, aby výměna implementace nevyžadovala změnu kódu v místě použití.

4.1.3 Datová vrstva

Tato vrstva obsahuje databázi a třídy potřebné pro komunikaci s ní.

K ukládání dat se běžně používají SQL databáze, kde se data ukládají do tabulek a vazby mezi objekty se modelují pomocí odkazů na řádek jiné tabulky. Jednou z takových databází je PostgreSQL - open-source databáze známá svou spolehlivostí, výkonem a funkcionalitami [13].

Object Relation Mapping (ORM) slouží k synchronizaci mezi 2 rozdílnými reprezentacemi dat, jedním v relační databázi a druhým v paměti [14].

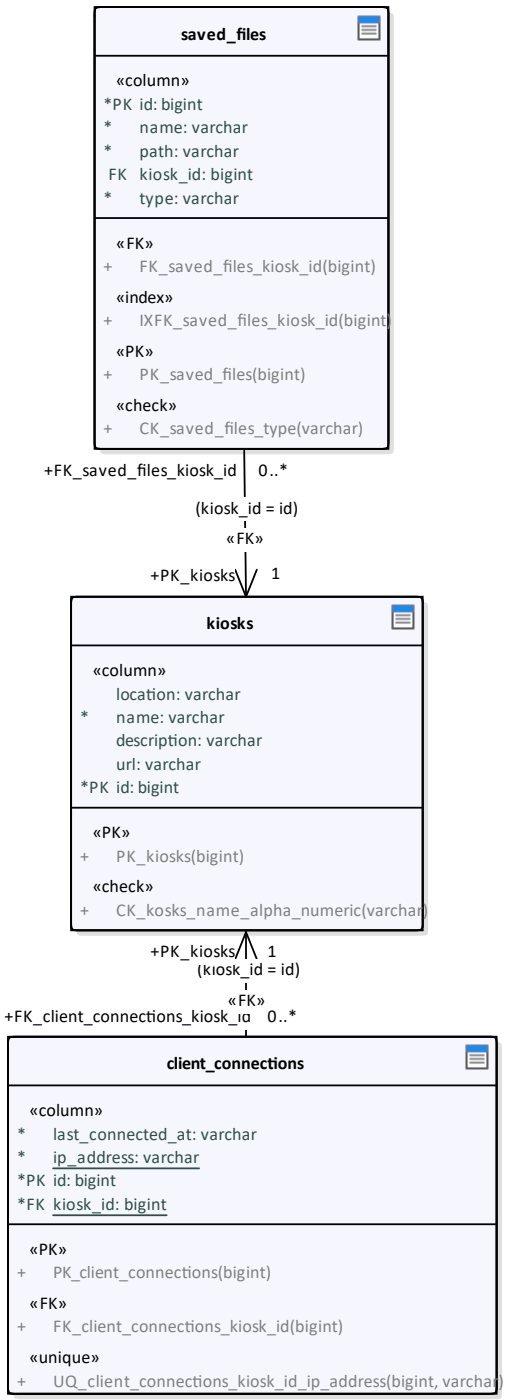
Ebean ORM podporuje řadu databází a je flexibilní, protože umožňuje různé úrovně abstrakce pro dotazy nad databází, mezi které patří typově bezpečný způsob pro psaní dotazů bez použití SQL [15].

Tato vrstva je v aplikaci uložena v balíčku data, jenž obsahuje další 2 balíčky. Repozitáře mají vlastní balíček repository. Tyto repozitáře používají ORM ke CRUD operacím nad entitami, které jsou uloženy v balíčku entity. Repozitáře jsou vždy odděleny přes Interface, aby výměna ORM či databáze nezasáhla zbytek aplikace.

Databázový model je popsán diagramem 4.2. Hlavní tabulka kiosks obsahuje informace o kiosku - zobrazované URL, jméno, popisek a fyzickou lokaci. Jméno kiosku je omezeno na alfanumerické znaky bez diakritiky přes check CK_kosks_name_alpha_numeric. Tabulka saved_files obsahuje informace o ukládaných souborech, které v této době jsou pouze systémové logy. kiosk_connections obsahuje informace o připojení kiosků k serveru. Na této tabulce je unique constraint přes sloupečky ip_address a kiosk_id. Tímto se zamezí vytváření řádků, které by se lišily pouze časem připojením a tím pádem z této tabulky lze vyčíst IP adresy, ze kterých se kiosek připojil, a čas posledního připojení z této adresy. Detailnější popis lze najít ve vygenerované dokumentaci v balíčku data.entity.

4.2 Programovací jazyk

Programovací jazyk Java je určen nefunkčními požadavky. Tato práce vzniká ve spolupráci s firmou Factorify, jejíž informační systém je psán v Javě ve



Obrázek 4.2: Databázový model

frameworku Spring, tudíž bylo rozhodnuto, aby práce byla napsána ve stejné technologii.

Dle [16] Java patří mezi populární jazyky pro psaní backend serverů. Dle [17] mezi její hlavní výhody patří:

- jednoduchost,
- výkonnost,
- je objektově orientovaná,
- kód je nezávislý na platformě, kompiluje se do Java VM bytecode a lze spustit na jakémkoliv Java VM.

4.3 Spring

Ze stejných důvodů jako v předchozím bodě je volba Springu je určena nefunkčními požadavky. Spring patří mezi velice populární open-source frameworky a v základu poskytuje vývojářům Inversion of Control a Dependency Injection, ale má také mnoho jiných modulů, které lze použít, např. webový server [18].

4.3.1 Inversion of Control

Inversion of Control (IoC) slouží k přesunutí odpovědnosti za volání kódu z programu na framework. Jinými slovy, místo toho, aby kód aplikace volal funkce frameworku, framework volá kód aplikace. Díky tomu lze oddělit vykonání úkolu od jeho implementace a jednodušeji měnit implementaci [19].

Dependency Injection (DI) je jeden ze způsobů, jak lze implementovat IoC. DI spočívá v definování závislostí objektů přes konstruktor, setter nebo jiný způsob, který IoC container použije při instanciování objektu, aby dodal požadovanou závislost za programátora [20]. V této práci bude použit DI přes konstruktor.

4.3.2 Komponenty Springu pro DI

Dle [20] Spring skenuje třídy aplikace a rozhoduje, které třídy má použít k DI. Jedním způsobem, jak označit třídy, je pomocí těchto anotací:

@Component je základní označení třídy bez speciálního významu,

@Service slouží pro označení tříd v aplikační vrstvě,

@Repository slouží k označení tříd pro perzistenci dat,

@Controller slouží k označení tříd v prezentační vrstvě a typicky má zvolené URL k mapování požadavků webového serveru.

4.4 Sestavení aplikace

Gradle je nástroj pro automatické sestavení aplikace, udržování závislostí a spouštění testů aplikace [21].

Docker je nástor pro vývoj, sestavení a nasazení aplikací. Aplikace běží v prostředí zvaném container. Pomocí nástroje docker-compose lze spustit více kontejnerů zároveň, ve kterých běží například databáze pro aplikaci. Díky tomu lze jednodušeji a rychleji testovat a nasazovat aplikace [22].

4.5 Kiosek

U kiosku se věnuji hardware, použitému operačnímu systému, způsobu jeho instalace a aplikaci klienta.

4.5.1 Raspberry PI

V České republice je oficiálním prodejcem Raspberry Pi zařízení RPishop. Nejnovější Raspberry Pi je verze 4B, která má 4 varianty lišící se ve velikosti RAM paměti - 1/2/4/8 GB. Pro potřeby kiosku je dostačující 4GB verze, kterou RPishop prodává v sadě[1] společně s SD kartou, krabičkou, napájením, HDMI kabelem a chladičem za 2 638 Kč.

4.5.2 Operační systém

Kiosky využijí oficiálně podporovaný Raspberry Pi OS, dříve známý jako Raspbian. Raspberry Pi OS staví na linuxové distribuci Debian, která je upravena pro optimální běh na hardware Raspberry Pi [23].

Raspberry Pi OS je k dispozici[2] ve 3 variantách:

Lite nejmenší z nich, obsahuje pouze příkazovou řádku se základními linuxovými utilitami.

Desktop obsahuje grafické rozhraní společně se základními aplikacemi jako VLC či Chromium.

Desktop and recommended software obsahuje totéž, co Desktop a navíc různé vývojové prostředí, kancelářské aplikace atp.

Pro potřeby této práce je ideální verze Desktop, do které se po instalaci na kartu nahraje klient. Lze také upravit Lite verzi, nahrát na ni potřebné aplikace (grafické rozhraní, webový prohlížeč) a připravit si tím vlastní minimální image. Tento image by byl menší ve velikosti, což urychlí instalaci, ale v zájmu jednoduchosti volím Desktop variantu.

[1] `https://rpishop.cz/4gb-ram/3572-zonepi-sada-s-raspberry-pi-4-4gb-ram-32gb-karta-oficialni-krabicka-cerna.html`

[2] `https://www.raspberrypi.org/software/operating-systems`

4.5.3 Instalace systému

Při instalaci systému je potřeba vyřešit 2 problémy - jak identifikovat zařízení a jak nainstalovat aplikaci klienta. Každé Raspberry Pi má unikátní výrobní číslo, které lze zjistit pouze za běhu a které není napsáno na zařízení. Toto číslo se lehce plete, a proto je vhodné při instalaci nechat administrátora zvolit identifikátor, pomocí kterého na serveru snadno určí konkrétní Raspberry Pi. Výhody a nevýhody zvažovaných přístupů jsou popsány v následujících kapitolách.

4.5.3.1 Network boot

Načítání systému přes síť umožňuje udržovat jednu verzi systému na serveru, což značně zjednodušuje konfiguraci systému a instalaci klienta. Instalaci nové verze lze snadno provést restartováním zařízení a není potřeba fyzické SD karty, která může časem selhat.

Nevýhodou je ale identifikace zařízení - pokud by administrátor instaloval více zařízení najednou, je potřeba rozlišit tato zařízení jinak než přes jejich sériové číslo. Pokud se zařízení připojí do switche, lze pomocí IP adresy přiřadit zařízením jejich jména.

Pro povolení network boot je potřeba Raspberry Pi alespoň jednou spustit s kartou a povolit ho v BIOS, což s předpřipravenou kartou není problém.

4.5.3.2 Nahrání systému a aplikace na SD kartu

Další přístup je na kartu nahrát image systému a poté do filesystému nahrát vše potřebné, např. aplikaci klienta, identifikátor karty nebo IP adresu serveru.

Výhodou tohoto přístupu je jeho jednoduchost - jde o připojení karty, spuštění instalačního skriptu a zapojení zařízení s nainstalovanou kartou.

Aktualizace systému je s tímto přístupem o něco obtížnější, protože se musí řešit buď ručně na všech zařízeních, nebo se o to musí postarat aplikace klienta.

Další nevýhodou je, že tento přístup vyžaduje zápis na ext4 filesystem, který Windows nativně nepodporuje. Lze využít softwaru třetí strany[3], který je ale pro komerční účely placený. Dalším možným řešením je využít automatizační software (např. Ansible), pomocí kterého se do zařízení po jeho spuštění nahrají potřebné soubory přes síť. Zde je ale potřeba spustit zařízení a znát jeho IP adresu (např. přes zapojení do switche).

4.5.3.3 Výběr způsobu instalace

Protože se v této práci klade důraz na jednoduchost a rychlost instalace kiosku, byl zvolen druhý přístup - instalace na SD kartu pomocí linuxového zařízení. Po stáhnutí image Raspberry Pi Os stačí připojit kartu k počítači,

[3] `https://www.paragon-software.com/business/extfs-for-windows/`

spustit skript, který na kartu nainstaluje systém, nahraje na ni potřebné soubory a poté spustit Raspberry Pi, které se připojí k serveru, na kterém administrátor nastaví URL na zobrazení. Z hlediska administrátora se vyžaduje malý počet rychlých akcí a lze připravit více karet zároveň (a dopředu) bez zbytečných komplikací.

Implementování ostatních možností instalace může proběhnout v rámci dalšího vývoje po odevzdání této práce, pokud o to administrátoři budou mít zájem.

4.5.4 Aplikace klienta

Klient se stará o komunikaci se serverem, ovládání zařízení (kiosk mód, restartování, atp.).

Aplikaci klienta ovlivňuje způsob a směr komunikace mezi serverem a zařízením. Aplikace by nebylo potřeba v případě, kdy se server pomocí SSH připojí na zařízení a vykoná potřebné příkazy. Tento přístup vyžaduje přístup k SSH ze serveru a změny se promítnou okamžitě na zařízení. Pokud se každému zařízení nepřidělí statická IP adresa, pořád je potřeba aplikace klienta, která se připojí k serveru a sdělí mu svoji aktuální IP adresu.

Opačný přístup je ten, kdy se zařízení periodicky připojuje k serveru a zjišťuje si změny. Ačkoliv chvíli trvá, než se promítnou změny ze serveru, tento přístup neklade požadavky na SSH a statické IP adresy, což je důvod, proč jsem ho zvolil.

Vzhledem k systémové povaze úloh, které se vyžadují od klienta, se vybízí použít BASH, který se běžně používá na administraci systému. Klient bude napsán formou BASH skriptů, které se v pravidelných intervalech připojí na server, ze kterého si stáhnou požadované akce - URL zobrazované stránky, nahrání logů na server, restart zařízení a uložení SSH klíče.

Také je nutné vyřešit, jakým způsobem periodicky spustit aplikaci klienta. Na Debianu, ze kterého Raspberry Pi OS vychází, se periodické pouštění programů dá zařídit 2 způsoby - systemd timer a cron. Mezi hlavní výhody systemd timer oproti cron patří oddělení spuštění služby od jejího načasování, což usnadňuje hledání chyb, lze tvořit závislosti mezi službami (např. spuštění služby až po připojení k síti) nebo logování do journalctl. Více se lze dočíst zde [24] nebo zde [25].

4.5.5 Správa zařízení

Klient se stará o správu zařízení - nastavení kiosk módu, nahrání logů, restartování zařízení a uložení SSH klíče. Během každého spuštění si ze serveru stáhne akce, které následně vykoná.

K zobrazování webových stránek je použito Chromium spustitelné přes LXSession, které „automaticky spouští aplikace a připravuje pracovní desktopové prostředí“ [26]. LXSession je součástí Raspberry Pi OS a není nutné

doinstalovávat. Pro nahrání logů server vystaví endpoint, na který kiosek zvolený log nahraje.

Součástí zadání je zpřístupnit SSH na kiosky. Toto lze docílit 2 způsoby - během instalace a dodatečně přes server. Během instalace se na kartu nakopíruje authorized_keys soubor, ve kterém jsou uloženy veřejné části klíčů, které se mohou připojit na zařízení bez použití hesla. Server také povolí nahrát klíč, který si pak kiosky stáhnou a uloží.

4.6 Testování a Continuous Integration

Aplikace bude testována 2 způsoby - jednotkové testy pro Java kód a End-to-End (E2E) test na konci vývoje, během kterého zkontroluji funkcionalitu aplikace.

Standardem pro jednotkové testování Java aplikací je framework JUnit. Jednotkové testy se skládají z funkcí, které testují jednu malou část aplikace. Testy probíhají při každém sestavení aplikace a pokud test selže, sestavení aplikace selže také. Jednotkově testována bude servisní a prezentační vrstva. Při testování budou závislosti každé třídy nahrazeny Mock objekty, u kterých lze předem definovat, jaké výsledky mají vracet. Tím lze otestovat kód dané třídy v izolaci.

Continuous Integration (CI) umožňuje pustit skript pokaždé, kdy se do Git repozitáře dostanou změny. Tento skript může sestavit aplikaci, otestovat ji a zvalidovat změny v kódu předtím, než se dostanou do hlavní větve [27]. V této práci použiji fakultní Gitlab, ve kterém se budou automaticky spouštět jednotkové testy aplikace.

End-to-end test (E2E) testuje aplikaci jako celek, od začátku do konce. Na konci vývoje provedu manuální test, ve kterém nainstaluji a spustím několik zařízení zároveň, abych odladil systém jako celek a ověřil, že je splněno zadání.

Kapitola 5

Implementace

V této kapitole popisuji vývoj aplikace, jak fungují nejdůležitější části aplikace a jak spolu tyto části komunikují.

5.1 Verzovací systém

Dle [28], verzovací systém (zkr. VCS z anglického version control system) je systém pro zaznamenávání změn nad soubory, aby šlo dohledat specifické verze z minulosti. Mezi hlavní možnosti verzovacích systémů patří:

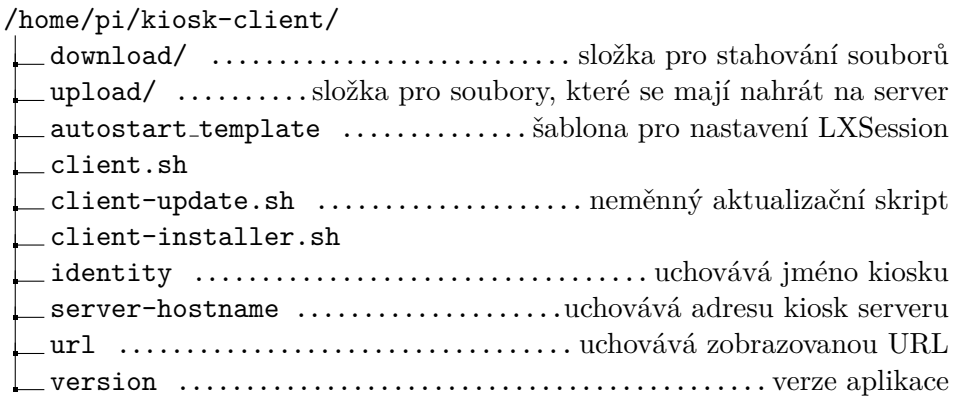
- zahodit změny a vrátit se k předchozím verzím,
- porovnat verze,
- zjistit, kdo upravil jakou část souboru,
- zjistit, v jaké verzi vznikla chyba,
- předejít ztrátě změn.

Pro verzování zdrojových kódů je dnes asi nejčastěji používaným systémem Git, který je jednoduchý, rychlý a umožňuje pracovat více lidem na stejném projektu. V této práci používám Git společně s fakultním Gitlab repozitářem[4], na kterém se při každé změně kódu spustí jednotkové testy. Zdrojové kódy jsou také dostupné v nefakultním Gitlab repozitáři[5], kde bude aktuální verze i po skončení mého studia na FIT ČVUT.

5.2 Aplikace klienta

Aplikace klienta jsou skripty v jazyce BASH, které jsou spouštěny přes systemd service a timer. Očekává se, že se klient bude postupem času měnit, tudíž

[4] `https://gitlab.fit.cvut.cz/klickjan/bc-kiosk-manager`
[5] `https://gitlab.com/klicka-jan/kiosk-manager`



Obrázek 5.1: Struktura souborů klienta

je potřeba zajistit jeho aktualizaci. Jak je nastíněno zde [29], interpret může načítat skript do paměti po kouscích, tudíž při aktualizaci skriptu (přepsání .sh souboru) může dojít k nečekanému chování. Proto je vhodné mít neměnný aktualizační skript, který stáhne a spustí proměnlivou část aplikace.

5.2.1 Úložiště souborů

Soubory jsou uloženy ve složce kiosk-client, která se nachází v domovské složce defaultního uživatele pi. Do složky download/ se stahují soubory, které se těsně před použitím přesunou na své místo. Obdobně funguje složka upload/, do které se ukládají soubory před nahráním na server. Funkce hlavních souborů jsou popsány v následujících kapitolách. Struktura je popsána diagramem 5.1.

5.2.2 client-update.sh

Tento jednoduchý aktualizační skript, který se nakopíruje na kartu při její instalaci, kontroluje, zda lokální verze klienta odpovídá verzi na serveru, a pokud je na serveru nová verze, stáhne a spustí client-installer.sh, který provede instalaci.

Pro nastavení systemd služby je potřeba 2 souborů - .service a .timer. .service soubor definuje, co se má vykonat, a .timer soubor definuje, jak často a kdy se to má vykonat. Následující kód demonstruje nastavení služby, která co hodinu pouští client-update.sh. Služba má závislost na multi-user.target, což znamená, že se spustí jako součástí normálního startu systému.

Listing 5.1: kiosk-client-update.service

```
[Unit]
Description=Updating service for kiosk client
After=multi-user.target
```

```
[Service]
Type=simple
Restart=on-failure
RestartSec=30
StartLimitBurst=3
StartLimitIntervalSec=180
ExecStart=bash /home/pi/kiosk-client/client-update.sh

[Install]
WantedBy=multi-user.target
```

Listing 5.2: kiosk-client-update.timer

```
[Unit]
Description=Timer kiosk-client-update

[Timer]
OnUnitActiveSec=1800

[Install]
WantedBy=timers.target
```

5.2.3 client-installer.sh

Tento skript se stará o registraci kiosku na server a o samotnou instalaci klienta. Jedná se o jednorázová nastavení pro běh klienta. Proces instalace se dá rozdělit do těchto kroků:

1. Registrace klienta na serveru.
2. Stažení souborů potřebných pro běh klienta - autostart_template, konfigurační soubory systemd služby klienta, client.sh.
3. Zastavení služby klienta.
4. Přesunutí souborů klienta na jejich místo.
5. Nastavení a spuštění systemd služby klienta.

5.2.4 client.sh

Tento skript má na starosti zpracování požadavků ze serveru. Pouští se stejně jako client-update.sh s častějším intervalem 5 minut. Během každého spuštění si ze serveru stáhne a vykoná požadované akce. Tyto akce jsou:

- aktualizace webové stránky kiosku,

- nahrání logů na server,
- restart zařízení,
- uložení SSH klíče.

5.2.5 Kiosk mód a zobrazování webové stránky

Kiosk mód je zajištěn přes Chromium a LXSession. V konfiguračním souboru LXSession autostart se definují příkazy, které se mají provést při spuštění prostředí. Každý příkaz je na nové řádce, která začíná zavináčem. Jako webový prohlížeč je použito Chromium, které se spouští následovně:

```
@chromium-browser --incognito \
  --noerrdialogs --kiosk {webUrl}
```

Při změně webové stránky je nutné změnit nastavení LXSession a poté restartovat LXSession službu. Nastavení je předpřipraveno v autostart_template, ve které se pomocí nástroje sed změní {webUrl} za URL ze serveru.

Chromium běží v incognito módu, aby se neukládala historie prohlížených webových stránek a cookies. Nevýhodou incognito módu pro tento projekt je ta, že incognito mód neukládá logy webových stránek, tudíž nelze splnit část UC5: Stažení logů, které vyžaduje stažení logu webového prohlížeče.

5.2.6 Instalace

Instalace je koncipována tak, aby po administrátorovi vyžadovala co nejméně kroků. Proto je připraven BASH skript, který na kartu nainstaluje a upraví systém, a díky tomu je tento proces pro administrátora krátký - stačí připojit kartu k počítači, zjistit pod jakým zařízení se připojila (např. pomocí `sudo fdisk -l`), spustit skript a po jeho doběhnutí zapojit kartu do zařízení.

Raspberry Pi OS využívá ext4 filesystem, který je během instalace potřeba připojit k počítači a vzhledem k tomu, že Windows nepodporuje nativně čtení a zápis ext4, instalace je připravena pro linuxové zařízení.

Skript prepare-rpi.sh má na vstupu tyto údaje:

- image Raspberry Pi OS,
- údaje k připojení na WiFi,
- hostname / IP adresu serveru,
- alfanumerické jméno (identitu) kiosku,
- složky do kterých může skript připojit filesystem karty,
- URL webové stránky na zobrazení (volitelné),

- veřejná část SSH klíče, pomocí kterého se lze připojit na zařízení (volitelné).

Instalační skript provede tyto kroky:

1. Nahrát image na kartu.
2. Vytvořit složky do kterých se v následujícím kroku připojí boot a system partition karty. Tyto složky se vytvoří v /mnt/kiosk-manager/{millis}/, kde {millis} znamená počet milisekund od půlnoci 1.1.1970. Díky tomu lze skript spustit paralelně.
3. Připojit boot a system partition karty k počítači.
4. Na boot partition nahrát údaje k připojení WiFi a povolit SSH službu vytvořením ssh souboru.
5. Do pracovní složky klienta na system partition nahrát client-update.sh, server_hostname, identity.

   Pokud administrátor na vstupu zadá URL, uloží se do souboru url v pracovní složce klienta.

   Pokud je k dispozici authorized_keys, uloží se do /home/pi/.ssh/.
6. Povolit systemd službu kiosk-client-update.

   Na běžícím systému povolení služby probíhá přes příkazovou řádku, která v tento moment není k dispozici, proto se musí vytvořit následující symbolické linky, které službu povolí.

   - Nakopírovat kiosk-client-update.timer a kiosk-client-update.service do /etc/systemd/system/.
   - Ze složky /etc/systemd/system/network-online.target.wants/ vytvořit link na kiosk-client-update.service.
   - Ze složky /etc/systemd/system/timers.target.wants/ vytvořit link na kiosk-client-update.timer.
7. Vypnout uspávání zařízení a šetřič obrazovky nakonfigurováním X11 Window System.
8. Odpojit kartu od systému.
9. Smazat vytvořené složky.

5.3 Server

Server je naimplementován podle struktury popsané v návrhové kapitole. Spring Controllers se nachází v balíčku api, služby se nachází v balíčku application_logic, entity a repozitáře se nachází v balíčku domain.

5.3.1 REST API

Rozhraní aplikace se dá rozdělit na 5 částí:

- ClientActionController,
- LinuxClientController,
- KioskController,
- StorageController,
- soubory klienta.

Detailně se budu věnovat pouze ClientActionController, LinuxClientController a souborům klienta, jelikož toto jsou nejdůležitější části. KioskController poskytuje rozhraní pro CRUD operace nad entitou Kiosk, StorageController poskyuje rozhraní pro nahrávání a stahování souborů, primárně logů z kiosku. Více se dočtete o těchto třídách ve vygenerované dokumentaci.

Server používá ke komunikaci formát dat JSON až na výjimky u endpointů pro klienta, které používají „text/plain“, aby na straně klienta nemuselo probíhat JSON parsování.

5.3.2 LinuxClientController

Tato třída má za úkol poskytnout rozhraní pro komunikaci s klientem, která je obstarána přes 2 endpointy s prefixem client/linux/:

PUT /register Provede registraci klienta po jeho prvním spuštění. Akceptuje také URL klienta, kterou si uloží. Pokud se klient již jednou zaregistroval, vícenásobné volání tohoto endpointu nic neprovede.

POST /action/{kioskName} Vrátí požadované akce po klientovi. Pokud server eviduje URL pro kiosek, vždy se pošle akce s URL stránky pro zobrazení. Také ukládá informace o připojení klienta na server včetně jeho IP adresy. Poslané akce klientovi se na serveru přestanou evidovat.

5.3.3 ClientActionController

Tato třída má za úkol poskytnout rozhraní pro akce klienta z pohledu administrátora, primárně aby mohl spravovat kiosek. Změna URL se provede pomocí úpravy entity Kiosk přes KioskController. Všechny tyto endpointy mají prefix client/. Mezi hlavní endpointy pro administrátora patří:

GET /action/{kioskName} Vrátí požadované akce od specifikovaného kiosku. Na rozdíl od této varianty endpointu pro klienta neobsahuje akci změny URL webové stránky a nemaže požadované akce.

POST /log/system/{kioskName} Vytvoří požadavek na stažení system logu z kiosku.

POST /restart/{kioskName} Vytvoří požadavek na restartování kiosku.

POST /ssh Vytvoří požadavek, aby si klienti specifikovaní v těle požadavku uložili SSH klíč.

5.3.4 Soubory klienta

Soubory klienta jsou poskytovány jako statické soubory přes Spring pod URL client/linux/{fileName}. Seznam těchto souborů je:

- version,
- autostart_template,
- client.sh,
- client-installer.sh,
- kiosk-client.service,
- kiosk-client.timer.

Poskytování souborů klienta přes statické soubory Springu není ideální řešení - jejich výměna vyžaduje úpravu nebo vygenerování .jar souboru. Lepší řešení je tyto soubory poskytnout přes webový server (např. Nginx), který požadavky na soubory zpracuje sám a ostatní požadavky nechá zpracovat aplikací.

5.4 Interakce mezi klientem a serverem

V této sekci popisuji, jak klient využívá rozhraní serveru během instalace a normálního používání. Vedle textového popisu jsou i sekvenční diagramy.

5.4.1 Interakce během instalace

Níže je popsáno, jak klient volá server během instalace a aktualizace klienta. Tento postup je také popsán diagramem C.1.

1. systemd po načtení sítě a pak každou hodinu spustí client-update.sh.
2. client-update.sh pomocí **GET /client/linux/version** zjistí aktuální verzi klienta.
3. Pokud se verze na serveru neshoduje s lokální verzí, tak stáhne a spustí client-installer.sh přes **GET /client/linux/client-installer.sh**.

4. client-installer.sh se přes **POST /client/linux/register** zaregistruje na serveru. V rámci tohoto požadavku pošle svoje jméno a také URL, pokud ji má lokálně uloženou.

5. Pokud server ještě neregistroval kiosek s daným jménem, uloží si ho do databáze případně i s jeho URL. Pokud se kiosek registroval již předtím, server nic neprovede.

6. Dále skript stáhne soubory klienta ze zdroje **GET /client/linux/**, seznam těchto souborů je popsán v kapitole Soubory klienta. Klienta zastaví, nainstaluje a spustí.

5.4.2 Interakce během normálního běhu klienta

Níže je popsáno, jak klient volá server během svého běhu. Tento postup je také popsán diagramem C.2.

1. systemd po načtení sítě a pak každých 5 minut spustí client.sh.

2. client.sh pomocí **POST /client/linux/action/{kioskName}** získá požadované akce.

3. Server si uloží aktuální IP adresu klienta a vrátí klientovi požadované akce.

4. Pro každou akci klient provede odpovídající úkon:

   URL změní nastavení LXSession, pokud se nová URL liší od lokální,

   uploadSyslog uloží system log z posledních 2 dnů a nahraje ho na server přes **POST storage/log/system/{kioskName}** a server jej následně uloží na disk,

   restart restartuje zařízení.

   SSH přidá specifikovaný klíč do lokálního authorized_keys.

5.5 Docker

Backend aplikace sestává ze 2 částí (databáze a server), jež se spouští ve vlastním Docker container.

Server aplikace má vlastní Dockerfile, ve kterém se definuje, jak se má vytvořit image, který se spustí v Docker container. Jako základ je použit openjdk:12-jdk image. Během sestavení se na tento image nahraje kód aplikace a pomocí gradle wrapper se sestaví jar soubor, který je použit jako vstupní bod image. Tento upravený image je k dispozici na Dockerhub pod tagem klickjan/kiosk-manager:latest[6].

[6] `https://hub.docker.com/repository/docker/klickjan/kiosk-manager`

Listing 5.3: Dockerfile

```
FROM openjdk:12-jdk
ENV GRADLE_OPTS -Dorg.gradle.daemon=false
COPY . /kiosk-manager
WORKDIR /kiosk-manager
RUN chmod +x gradlew
RUN ./gradlew build
RUN ./gradlew test

ENTRYPOINT ["java", "-jar",
  "/kiosk-manager/build/libs/kiosk-manager-1.0.0.jar"]
```

V souboru docker-compose.yml je popsána infrastruktura aplikace. Pro databázi je využit postgres:12 image a pro aplikaci je využit image popsaný v předchozím odstavci. Databáze má svazek postgres_data, ve kterém se perzistují data. Při vytváření toho svazku se spustí 01_create_dm.sql, který vytvoří tabulky požadované aplikací.

Listing 5.4: docker-compose.yml

```
volumes:
  postgres_data:

services:
  app:
    image: klickjan/kiosk-manager:latest

    ports:
      - "8080:8080"

  db:
    image: postgres:12
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_USER=kiosk_manager
      - POSTGRES_PASSWORD=kiosk_manager
      - POSTGRES_DB=kiosk_manager
    volumes:
      - ./src/main/resources/db/01_create_dm.sql:
          /docker-entrypoint-initdb.d/init.sql
      - postgres_data:/var/lib/postgresql/data
```

Kapitola 6

Testování

V této kapitole se věnuji testování aplikace. V rámci testování jsem použil 2 přístupy - jednotkové testování a manuální E2E test.

6.1 Jednotkové testování

V projektu se nachází 4 třídy na kontrolu REST rozhraní. V těchto třídách se testuje, že server má požadované chování (response code, tělo požadavků atp.) jak v normálních, tak v hraničních podmínkách.

- ClientActionControllerTest,
- KioskControllerTest,
- LinuxClientControllerTest,
- StorageControllerTest.

Ze servisní třídy je otestována pouze třída ClientInMemoryActionService. Ostatní třídy slouží buď jako proxy pro repozitáře nebo je nelze testovat jednotkově např. LocalFileStorage. Dále lze najít jednotkové testy pro menší kusy logiky.

6.2 E2E test

E2E test byl proveden pro otestování aplikace jako celku.

6.2.1 Scénář

Scénář je následující:

1. Spustit server na Linux prostředí.

2. Připravit 2 zařízení s označením Z1 a Z2 se zavedením SSH přístupu.
3. Spustit obě zařízení.
4. Vyžádat systémový log od Z1.
5. Změnit URL a vyžádat restart od Z2.
6. Stáhnout systémový log Z1 ze serveru.
7. Přeinstalovat zařízení Z1 s novým označením Z3.
8. Přeinstalovat zařízení Z2 s novým označením Z1.
9. Zkontrolovat, že nové Z1 má stejnou stránku jako původní Z1.
10. Stáhnout historii IP adres pro každé zařízení.
11. Připojit se přes SSH na Z1.
12. Přidat nový SSH klíč pro Z1.
13. Připojit se na Z1 přes nový SSH klíč.

6.2.2 Zhodnocení

Aplikace byla spuštěna podle instalační příručky pomocí Dockeru. Během testování jsem narazil na 2 problémy.

Soubory se na kartu kopírovaly pod špatným UID, tudíž nešlo přistoupit k authorized_keys a nešel použít SSH klíč k připojení. Stačilo zjistit UID a GID uživatele pi a správně nastavit přístupová práva. Druhý problém nastával v případě, kdy se karta automaticky připojovala k systému. Instalační skript poté znovu připojil kartu a z důvodů, které jsem nezkoumal blíže kvůli nedostatku času, docházelo ke korupci dat na kartě. Po vypnutí automatického připojení vše fungovalo v pořádku.

Proces přípravy karty trvá mezi 2 až 5 minutami v závislosti na rychlosti čtečky karet, která byla použita. Tuto rychlost lze zlepšit použitím menšího image sytému. Z pohledu administrátora stačí zjistit jméno zařízení, pod kterým je karta připojena k systému (např. /dev/sdc) pomocí `sudo fdisk -l`, spustit skript se správnými argumenty a zapojit kartu do zařízení. Nejdéle trvá nahrání image systému na kartu, akce administrátora při instalaci nezabere déle než 1 minutu, tudíž požadavek rychlosti instalace je splněn.

Kapitola 7

Další vývoj

Systém je v tento moment funkční a připraven k použití, přesto ho lze rozšířit a zlepšit. V této kapitole krátce shrnu další možnosti vývoje a nasazení v praxi.

7.1 Rozšíření

Momentálně systém podporuje pouze instalaci systému na kartu přes instalační skript. Pokud by se při nasazení v praxi zjistilo, že je tento přístup nevyhovující, lze implementovat jiné způsoby. Navíc je systém limitován pouze na Linux/Raspberry Pi zařízení. Ačkoliv by klient šel použít i na jiných linuxových zařízení, jejich instalace není vyřešena. V budoucnu by také šla přidat podpora pro více různých klientů, např. na mobilní platformu Android.

Systém neřeší zabezpečení, server vykoná každý požadavek a při fyzické přítomnosti u kiosku lze přepnout na jiné aplikace systému. Tento systém byl koncipován pro interní sítě, jež nemají přístup na internet, a interní hrozbu, kde by někdo přepínal stránky kiosků, neřeší. Tento přístup nemusí vzbuzovat důvěru potencionálních uživatelů, proto by bylo vhodné zabezpečit systém, aby nešlo ukončit kiosk mód jinak než přes SSH, a implementovat autentizaci na serveru. Navíc management SSH klíčů není ideální - SSH klíče je nutno mít k dispozici při instalaci kiosku nebo si od každého kiosku vyžádat akci přidání klíče. Ideální by bylo mít na serveru uložen seznam klíčů, který by si klient při instalaci stáhl a uložil.

Akce pro klienta (restart, system log) se uchovávají pouze v paměti a při jejich vyzvednutí klientem se smažou, tudíž se server nedozví, kdyby klient selhal ve vykonání těchto akcí. Proto by bylo vhodné tyto akce mazat v moment, kdy je klient úspěšně vykoná - například při nahrání systémového logu by StorageService vyvolala událost SystemLogUploadedEvent, na kterou by ClientActionService poslouchala, a při jejím přijetí by požadavek smazala.

7.2 Nasazení v praxi

V době odevzdání této práce nebyl čas systém nasadit a otestovat v praxi, částečně kvůli současné pandemické situaci. Jakmile se situace vylepší, systém nasadíme a odladíme v reálném prostředí.

Závěr

Hlavním cílem této práce bylo vytvořit systém, který umožní rychle a jednoduše spravovat Raspberry Pi zařízení v kiosk módu. Tento systém je určen do výrobního prostředí továren, aby pracovníkům umožnil komunikaci s informačním systémem továrny, a tím docílil lepší efektivity práce. Tyto kiosky jsou levné, čímž firmy ušetří za drahé počítače, které nyní slouží tomuto účelu.

Skoro všechny cíle a požadavky byly splněny, až na jednu výjimku a tou je stahování logů z prohlížeče, která kvůli způsobu implementace nelze splnit. Práce obsahuje řešení pro rychlý způsob instalace kiosků na SD kartu, který administrátorům umožňuje jednoduše připravit zařízení. Práce také obsahuje klienta pro kiosky a server pro jejich správu včetně instalačních příruček, podle kterých lze server nasadit a spustit. Díky tomu je práce připravena pro produkční nasazení.

V budoucnu by bylo vhodné systém rozšířit zabezpečením serveru, aby pouze administrátoři mohli spravovat kiosky, a větším omezením kiosku, aby nešlo opustit prohlížeč. Dalším možným rozšířením je podpora více platforem např. Android.

Práce je rozdělena na rešerši existujících řešení, analýzu požadavků na systém, návrh, implementaci a testování nového řešení, diskuzi dalšího vývoje, závěr a instalační příručku.

Výsledkem je systém, který bude otestován a odladěn u klientů Factorify, aby z něj mohli těžit i ostatní, díky čemuž tato práce bude mít společenský přínos.

Bibliografie

1. *Autocont Eshop* [online]. AutoCont IPC a.s., 2021 [cit. 2021-03-24]. Dostupné z: `https://eshop.autocont-ipc.cz/`.
2. PETR, Jakub. *Současné řešení kiosků u klientů Factorify* [online rozhovor]. Factorify, s. r. o. [cit. 2021-03-24].
3. *Ki-Wi Kiosk Solution* [online]. AutoCont IPC a.s., 2021 [cit. 2021-03-24]. Dostupné z: `https://eshop.autocont-ipc.cz/softwarova-reseni/ki-wi-kiosk-solution/`.
4. *Manage Engine* [online]. Zoho Corporation, 2021 [cit. 2021-03-24]. Dostupné z: `https://www.manageengine.com/company.html?MEfooter`.
5. *Mobile Device Manager* [online]. Zoho Corporation, 2021 [cit. 2021-03-24]. Dostupné z: `https://www.manageengine.com/mobile-device-management/`.
6. *Kioware product overview* [online]. KioWare, 2021 [cit. 2021-03-24]. Dostupné z: `https://www.kioware.com/productoverview.aspx`.
7. *Porteus kiosk* [online]. Porteus solutions, 2021 [cit. 2021-03-25]. Dostupné z: `https://porteus-kiosk.org/index.html`.
8. *How Ansible Works* [online]. Red Hat, Inc, 2020 [cit. 2021-03-24]. Dostupné z: `https://www.ansible.com/overview/how-ansible-works`.
9. OTT, Christian. *pikiosk* [online]. GitHub, 2017 [cit. 2021-03-25]. Dostupné z: `https://github.com/chriso0710/pikiosk`.
10. WIEGERS, K.; BEATTY, J. *Software Requirements*. Pearson Education, 2013. Developer Best Practices. ISBN 9780735679627. Dostupné také z: `https://books.google.cz/books?id=nbpCAwAAQBAJ`.
11. *OMG Unified Modeling LanguageTM (OMG UML), Superstructure* [online]. Object Management Group, 2011 [cit. 2021-04-08]. Dostupné z: `http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/`.

12. FOWLER, M.; RICE, D. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2003. ISBN 9780321127426. Dostupné také z: https://books.google.cz/books?id=Jl5rkQnbfAIC.

13. *What is PostgreSQL* [online]. PostgreSQL Global Development Group, 2021 [cit. 2021-04-20]. Dostupné z: https://www.postgresql.org/about/.

14. FOWLER, Martin. *OrmHate* [online]. 2012-05 [cit. 2021-04-20]. Dostupné z: https://martinfowler.com/bliki/OrmHate.html.

15. *Ebean ORM* [online]. 2021 [cit. 2021-04-11]. Dostupné z: https://ebean.io/.

16. PATHAK, Ninad. *Top 10 best programming languages* [online]. JournalDev, 2020 [cit. 2021-04-09]. Dostupné z: https://www.journaldev.com/43017/top-best-programming-languages.

17. *About the Java Technology* [online]. Oracle, 2020 [cit. 2021-04-09]. Dostupné z: https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html.

18. *Spring Framework Overview* [online]. Spring, 2021 [cit. 2021-04-09]. Dostupné z: https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html.

19. FOWLER, Martin. *Inversion Of Control* [online]. 2005 [cit. 2021-04-12]. Dostupné z: https://martinfowler.com/bliki/InversionOfControl.html.

20. *Core Technologies* [online]. Spring, 2021 [cit. 2021-04-10]. Dostupné z: https://docs.spring.io/spring-framework/docs/current/reference/html/core.html.

21. *Ebean ORM* [online]. Gradle Inc, 2021 [cit. 2021-04-20]. Dostupné z: https://gradle.org/.

22. *Docker overview* [online]. 2021 [cit. 2021-05-01]. Dostupné z: https://docs.docker.com/get-started/overview/.

23. *Raspberry Pi OS* [online]. Raspberry Pi Foundation, 2021 [cit. 2021-04-22]. Dostupné z: https://www.raspberrypi.org/software/.

24. *systemd/Timers* [online]. 2020 [cit. 2021-04-21]. Dostupné z: https://wiki.archlinux.org/index.php/Systemd/Timers#As_a_cron_replacement.

25. *Cron vs systemd timers* [online]. Stack Exchange Inc, 2021 [cit. 2021-04-21]. Dostupné z: https://unix.stackexchange.com/questions/278564/cron-vs-systemd-timers.

26. *LXSession* [online]. GitHub, 2021 [cit. 2021-05-01]. Dostupné z: https://github.com/lxde/lxsession.

27. *GitLab CI/CD* [online]. GitLab Inc [cit. 2021-04-10]. Dostupné z: `https://docs.gitlab.com/ee/ci/`.

28. CHACON, S.; STRAUB, B. *Pro Git*. Apress, 2014. The expert's voice. ISBN 9781484200766. Dostupné také z: `https://books.google.cz/books?id=jVYnCgAAQBAJ`.

29. THOMPSON, Keith. *Valid self-update approach for a bash script* [online]. Stack Exchange Inc, 2011 [cit. 2021-04-22]. Dostupné z: `https://stackoverflow.com/questions/8595751/is-this-a-valid-self-update-approach-for-a-bash-script`.

Příloha A

Seznam použitých zkratek

API Application Programming Interface

BASH Bourne Again SHell

BIOS Basic Input/Output System

CI Continuous Integration

CPU Central Processing Unit

CRUD Create Read Update Delete

E2E End to End

GB Gigabyte

GID Group Identificator

HDMI High-Definition Multimedia Interface

HW Hardware

IOC Inversion of Control

IP Internet Protocol

IT Information Technology

JSON JavaScript Object Notation

OS Operating System

RAM Random Access Memory

REST Representational State Transfer

SD Secure Digital

SSH Secure Shell Protocol

UC Use Case

UID User Identifiactor

USB Universal Serial Bus

URL Uniform Resource Locator

VCS Version Control System

VM Virtual Machine

Příloha B

Instalační příručka

V této kapitole se věnuji tomu, jak spustit systém a jak připravovat kiosky.

B.1 Backend

Backend lze nainstalovat jak přes Docker, tak manuálně.

B.1.1 Docker

Spuštění přes Docker je připraveno přes docker-compose nástroj. V docker-compose.yml se nachází konfigurace aplikace a databáze.

Ve složce, kde se nachází docker-compose.yml, je potřeba spustit příkaz `docker-compose up`, který stáhne příslušné images a spustí je.

Pro změnu přístupových údajů do databáze je nutné upravit environment properties jak pro Postgres, tak pro aplikaci. docker-compose.yml obsahuje zakomentované řádky, které demonstrují, jak změnit tyto properties.

Postgres používá svazek postgres_data pro perzistenci dat mezi restartováním databáze. Při vytvoření tohoto svazku se inicializuje databáze pomocí resources/db/01_create_dm.sql skriptu.

B.1.2 Manuální instalace

Při manuální instalaci je potřeba připravit 2 komponenty - PostgreSQL databázi a server.

Databáze se připraví pomocí následujících kroků (skripty se nacházejí mezi zdrojovými soubory serveru v resources/db):

1. Stáhnout a spustit PostgreSQL verze 12[7].
2. Vytvořit roli a databázi pro aplikaci. Pro tento krok je připravený skript 00_create_db.sql.

[7] `https://www.postgresql.org/download/`

3. Vytvořit databázový model. Databázový model se nachází ve skriptu 01_create_dm.sql.

Dále je potřeba sestavit, nakonfigurovat a spustit aplikaci.

1. Stáhnout .jar soubor serveru nebo ho sestavit pomocí gradle wrapper příkazem `./gradlew build`
2. V .jar souboru upravit application.properties pro konfiguraci připojení k databázi.

   Při použití připravených skriptů pro tvorbu databáze je třeba měnit pouze datasource.db.url.

   - storage.path - cesta pro ukládání souborů (logů),
   - datasource.db.username - jméno uživatele databáze,
   - datasource.db.password - heslo uživatele databáze,
   - datasource.db.databaseName - jméno databáze,
   - datasource.db.databaseDriver - driver databáze,
   - datasource.db.url - url připojení k databázi.
3. Spustit aplikaci.

B.2 Instalace kiosků

Pro instalaci kiosků je potřeba Linux počítače se čtečkou SD karet a složky client/ ze zdrojových souborů. Pokud systém automaticky připojuje SD karty, je doporučeno tuto funkcionalitu vypnout - během testování docházelo ke korupci dat. Na začátku je potřeba provést několik přípravných kroků:

1. Stáhnout Desktop image Raspberry Pi OS [8] do složky client/ se jménem raspbian.img.
2. Do client/sd-files/kiosk-client/kiosk-server-domain napsat hostname / IP adresu serveru, ke kterému se kiosek má připojit.
3. Do client/sd-files/wpa_supplicant.conf vyplnit přístupové údaje k WiFi, na kterou se má kiosek připojit.
4. Do client/sd-files/ssh/authorized_keys uložit veřejné části klíče (volitelné).

Pro instalaci kiosku je třeba:

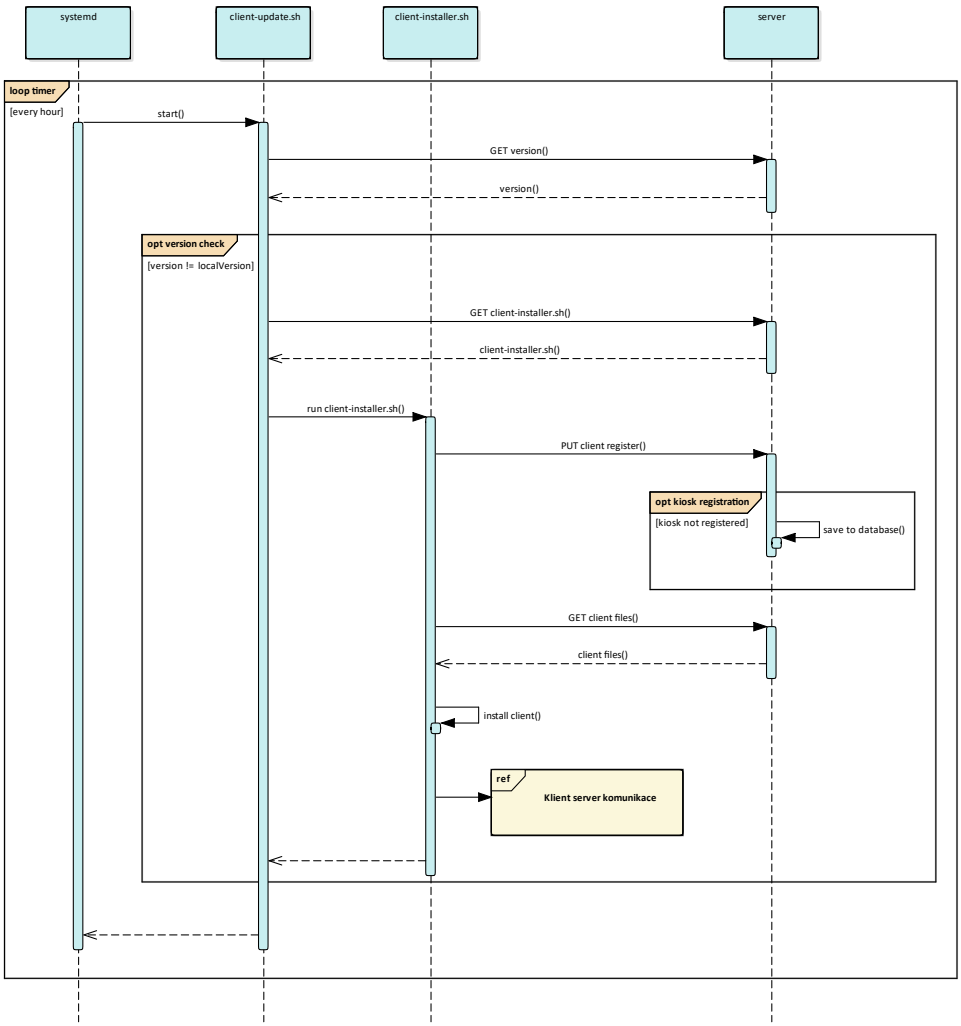
1. Zapojit SD kartu do počítače.

[8] `https://www.raspberrypi.org/software/operating-systems/`

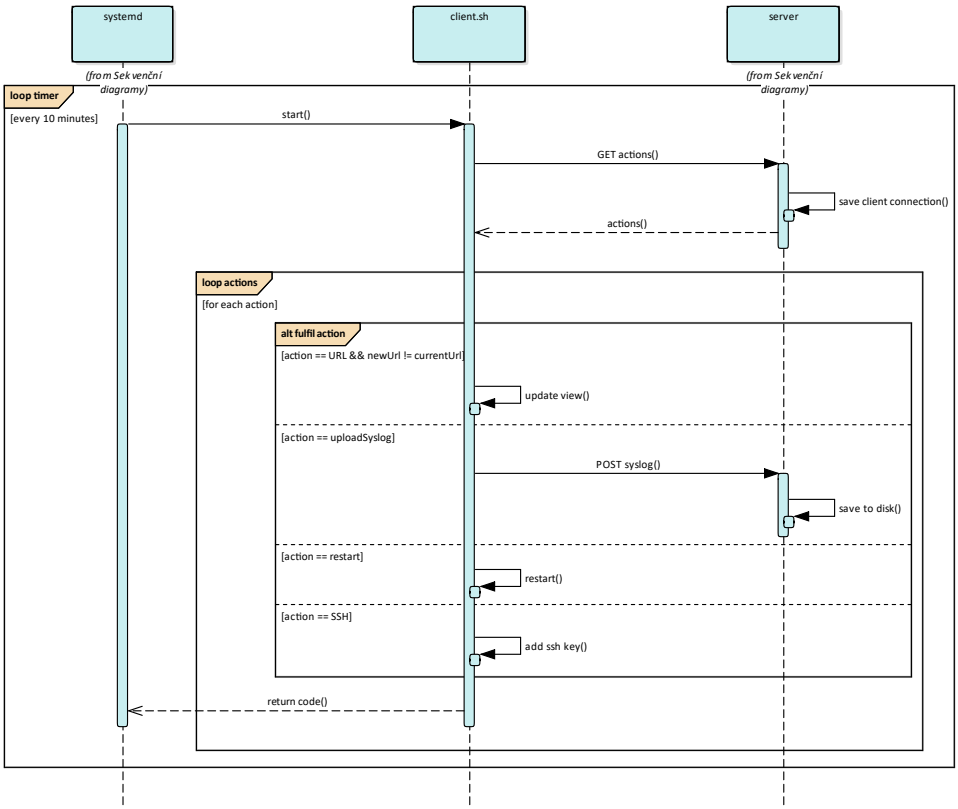
2. Zjistit pod jakým zařízením se karta připojila (např. pomocí `sudo fdisk -l`).
3. Spustit prepare-rpi.sh skript s pozičními argumenty:
   - alfanumerické jméno kiosku,
   - zařízení karty (např. /dev/sda),
   - URL webové stránky pro zobrazení (volitelné).
4. Zapojit zařízení v dosahu WiFi.

Příloha C

Diagramy



Obrázek C.1: Komunikace instalace klienta se serverem



Obrázek C.2: Komunikace klienta se serverem

Příloha D

Obsah přiloženého USB

- `readme.txt` ..... stručný popis obsahu USB
- `bin` ..... adresář se spustitelnou formou implementace
  - `kiosk-manager.jar` ..... jar soubor serveru
  - `docker-compose.yml` ..... konfigurace pro docker-compose
- `src` ..... zdrojové kódy
  - `kiosk-manager` ..... zdrojové kódy implementace
    - `server` ..... zdrojové kódy serveru
    - `client` ..... zdrojové kódy klienta
    - `javadoc` ..... vygenerovaná dokumentace serveru
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `thesis.pdf` ..... text práce ve formátu PDF