**Technická univerzita v Košiciach**
**Fakulta elektrotechniky a informatiky**

# Vzory dátovej synchronizácie v offline prostredí

**Diplomová práca**

**2022** **Bc. Roland Körtvely**

**Technická univerzita v Košiciach**
**Fakulta elektrotechniky a informatiky**

# Vzory dátovej synchronizácie v offline prostredí

**Diplomová práca**

| | |
|---|---|
| Študijný program: | Informatika |
| Študijný odbor: | 9.2.1. Informatika |
| Školiace pracovisko: | Katedra počítačov a informatiky (KPI) |
| Školiteľ: | doc. Ing. Martin Tomášek, PhD. |

**Košice 2022** **Bc. Roland Körtvely**

## Abstrakt v SJ

Diplomová práca sa zaoberá vypracovaním prípadových štúdií realizácie dátovej synchronizácie v offline-first prístupe, pričom analyzuje taktiež online-first prístup a distribuované systémy. Práca tiež navrhuje metódy a vzory pre tvorbu distribuovaných aplikácií, vypracováva prehľad súčasného stavu v oblasti online a offline dátovej synchronizácie a vývoja aplikácií prístupom offline-first. Vypracovaný pseudokód prináša riešenie pre komunikáciu zariadení bez prístupu k Internetu v rámci lokálnej siete a obohacuje tak už existujúce synchronizáčné algoritmy. Úvod práce je venovaný analytickej časti, kde sú spomenuté distribuované systémy, ich opis, znaky, nevýhody, architektúry a tiež bezpečnosť. V závere práce sme experimentálne vyhodnotili výsledok našej prípadovej štúdie na ukážkovom ideovom koncepte.

## Kľúčové slová v SJ

klient-server, peer-to-peer, distribuované systémy, paxos, asynchrónna komunikácia, synchrónna komunikácia, delta synchronizácia, online-first, offline-first

## Abstrakt v AJ

The thesis deals with the development of case studies of the implementation of data synchronization in an offline-first approach, while also analyzing the online-first approach and distributed systems. The thesis also proposes methods and patterns for the development of distributed applications, and develops an overview of the current state of the art in online and offline data synchronization and application development in the offline-first approach. The developed pseudocode provides a solution for the communication of devices without Internet access within a local area network, enriching already existing synchronization algorithms. The introduction of the thesis is devoted to the analytical part, where distributed systems, their description, features, drawbacks, architectures and also security are mentioned. At the end of the paper, we experimentally evaluate the result of our case study on a sample idea concept.

## Kľúčové slová v AJ

client-server, peer-to-peer, distributed systems, paxos, asynchronous communication, synchronous communication, delta synchronization, online-first, offline-first

## Bibliografická citácia

KÖRTVELY, Roland. *Vzory dátovej synchronizácie v offline prostredí*. Košice: Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky, 2022. 66s. Vedúci práce: doc. Ing. Martin Tomášek, PhD.

65048

**TECHNICKÁ UNIVERZITA V KOŠICIACH**

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra počítačov a informatiky

# ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **Informatika**
Študijný program: **Informatika**

Názov práce:

**Vzory dátovej synchronizácie v offline prostredí**

Offline Data Synchronization Patterns

Študent: **Bc. Roland Körtvely**
Školiteľ: **doc. Ing. Martin Tomášek, PhD.**
Školiace pracovisko: **Katedra počítačov a informatiky**
Konzultant práce:
Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:
1. Vypracovať prehľad súčasného stavu v oblasti online a offline dátovej synchronizácie a vývoja aplikácii prístupom offline first.
2. Navrhnúť metódy a vzory pre tvorbu distribuovaných aplikácií s rôznymi modelmi dátovej synchronizácie.
3. Vypracovať prípadové štúdie vzorovej realizácie distribuovaných aplikácií s využitím rôznych modelov dátovej synchronizácie.
4. Vyhodnotiť experimentálne výsledky prípadových štúdií.

Jazyk, v ktorom sa práca vypracuje: slovenský
Termín pre odovzdanie práce: 22.04.2022
Dátum zadania diplomovej práce: 29.10.2021

prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

## Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval(a) samostatne s použitím uvedenej odbornej literatúry.

Košice, 22.4.2022 .....

*Vlastnoručný podpis*

## Poďakovanie

Ďakujem doc. Ing. Martinovi Tomášekovi, PhD. za spoluprácu, ktorú sme mali v priebehu tejto práce, vďaka ktorej som dostal množstvo informácií, bez ktorých by som určite nezvládol napísať túto prácu.

# Obsah

# Zoznam obrázkov

# Zoznam tabuliek

# Úvod

Výpadok služieb Internetu, tak typický pre „internetový pravek“, je pri súčasnom sofistikovanom spôsobe manažovania internetu, resp. vzájomnej komunikácie jednotlivých elementov (serverov, routerov, koncových zariadení, ...) vzácnou udalosťou. No práve o to zradnejšou, o čo menej očakávanou, často s fatálnejšími a málo predvídateľnými, resp. zvládnuteľnými dôsledkami.

Jedna vec je vysporiadanie sa s fatálnym výpadkom služieb Internetu pri aplikácii, pri ktorej je funkčné pripojenie pre optimálny chod nevyhnutný a kde prechod do offline režimu, aj pri sebalepšom zvládnutí, znamená nutnosť zaobísť sa bez najaktuálnejších informácií (vzhľadom na absenciu pripojenia do centralizovanej databázy).

Inou výzvou, snáď môžeme tvrdiť, že jednoduchšou, keďže s offline režimom sa počíta ako s bežným prevádzkovým javom, však je stav absencie pripojenia na Internet pri aplikáciách, pri ktorých funkčné internetové pripojenie vôbec nie je nevyhnutné pre riadne manažovanie prevádzkových operácií.

Ako príklad môžeme uviesť fiktívnu predajňu, kde sa neplatí peniazmi, ale špeciálnymi vouchermi, resp. žetónmi. Ak zákazník zaplatí konkrétnym, jedinečným voucherom/žetónom pri jednom obchodníkovi, mal by systém daný voucher/žetón zablokovať tak, aby neprišlo k jeho zneužitiu opätovným uplatnením u iného obchodníka. Situácia je bezproblémová v online prostredí, kedy sa identifikačné čísla voucherov/žetónov konfrontujú s centrálnou databázou a pri uplatnení voucheru/žetónu sa tento označí (flaguje) ako použitý s príslušným údajom o čase uplatnenia (timestamp) a mieste uplatnenia (identifikačné parametre obchodníka). Tento jednoduchý model je však funkčne limitovaný (nie však nutne úplne nefunkčný) v prípade výpadku online spojenia s centrálnym serverom, kedy prestane byť dostupná referenčná databáza údajov o uplatnených/neuplatnených voucheroch/žetónoch a jednotlivé konečné overovacie zariadenia (napr. mobilné telefóny obchodníkov) začnú byť odkázané iba samé na seba.

Výpadky internetového pripojenia nemusia byť pritom spôsobené iba imanentným technickým problémom (poruchou zariadenia), ktoré je epizodickou

udalosťou, často ľahko odstrániteľnou zásahom odborného pracovníka. Vezmime si ako príklad hudobný festival v horách alebo na letisku vzdialenom od urbárneho prostredia. Geografický profil prostredia a pochopiteľná neochota mobilných operátorov investovať do nákupu a inštalácie výkonných technických vykrývačov signálu, resp. BTS staníc na málo lukratívne prostredie s nízkym počtom zákazníkov na meter štvorcový, je limitujúca realita, ktorá rezultuje do permanentného či znovu sa objavujúceho problému: namiesto kvalitného pokrytia signálom mobilných operátorov, na aké sme zvyknutí v meste, musia organizátori open-air podujatia počítať s nízkou kvalitou signálu, resp. s jeho častými výpadkami.

Každá vstupenka, vygenerovaná cez oficiálneho predajcu vstupeniek (ticketportal.sk, eventim.sk a pod.), obsahuje jedinečný kód. Konfrontácia kódu na vstupenke s databázou vygenerovaných a zakúpeným vstupenkám aj pridelených kódov je spoľahlivým spôsobom, ako znemožniť uplatnenie falošnej vstupenky alebo opätovné uplatnenie dávnejšie uplatnenej vstupenky (samotnému autorovi tejto práce sa neznámy ziskuchtivý účastník nemenovaného festivalu snažil predať jednu z viackrát vytlačenej (možno) pravej vstupenky s ubezpečením, že „na bráne to neskenujú“. Nuž, skenovali a my sme sa, našťastie, nedali nalákať na takýto pokútny biznis). No v nezávideniahodnej situácii výpadku prístupu na Internet nastáva vážny organizačný problém, ako v offline prostredí overovať pravosť lístkov na viacerých koncových zariadeniach (vstupných bránach, vstupných turniketoch), ktorými sa návštevníci preukazujú pri vstupe na kultúrne podujatie.

Mnohé výzvy však pochádzajú aj mimo komerčného prostredia. Predstavme si napríklad športový deň v detskom tábore. Deti súťažia v rôznych disciplínach a na rôznych miestach, ktoré sú pod dohľadom viacerých vedúcich. Keďže v hlbokých a temných horách nemáme prístup na Internet, potrebujeme osobu, ktorá bude obchádzať jednotlivé športové stanovištia a zbierať od vedúcich čiastkové výsledky súťaží, ktoré si vedúci evidujú napr. na svojich mobilných telefónoch. Poverená osoba čiastkové výsledky zozbiera prostredníctvom svojho mobilného telefónu a na služobnom notebooku (jedinom v tábore) ich priebežne vyhodnocuje, resp. na konci dňa rovnakým mechanizmom vyhodnotí výsledky finálne a na tlačiarni (jedinej v tábore) vytlačí diplomy pre prvých troch výhercov.

Pri formulovaní nášho záujmu a cieľov v tejto práci sme mali na zreteli vytvorenie návrhu takej aplikácie, ktorá by umožnila riešiť vyššie spomínané, no i mnohé analogické, problémy, ktoré, domnievame sa, nie sú vôbec zriedkavé, kedy výpadok pripojenia do Internetu ohrozuje fungovanie istého informačného sys-

tému, hoci by dokázal, aspoň na istý čas, fungovať aj na operáciách bezprostredne nezávislých od tohto pripojenia.

Za cieľ našej práce preto považujeme:

- Vypracovať prehľad súčasného stavu v oblasti online a offline dátovej synchronizácie a vývoja aplikácii prístupom offline-first
- Navrhnúť metódy a vzory pre tvorbu distribuovaných aplikácií s rôznymi modelmi dátovej synchronizácie
- Vypracovať prípadové štúdie vzorovej realizácie distribuovaných aplikácií s využitím rôznych modelov dátovej synchronizácie
- Vyhodnotiť experimentálne výsledky prípadových štúdií

# 1 Analytická časť

Naším primárnym zámerom je v tejto kapitole najskôr definovať distribuované systémy, synchronizáciu času a dát, toleranciu chybovosti, bezpečnosť, analyzovať dostupné riešenia offline-first prístupu, zhodnotiť ich princípy, ponúkané možnosti a posúdiť, v kontexte cieľov našej práce, vhodnosť alebo nevhodnosť či nedostatočnosť daného prístupu pre naše riešenie.

## 1.1 Distribuované systémy

Existuje množstvo definícií, ktoré popisujú distribuovaný systém ako aj princípy jeho fungovania. Jednoduchá, ale výstižná je definícia podľa van Steena [1], podľa ktorej distribuovaný systém vieme opísať ako sústavu zloženú zo serverov, resp. iných nezávislých zariadení, ktoré spolu komunikujú a v praxi spolupracujú na riešení spoločného problému. Z tohto dôvodu sa z pohľadu používateľa javia ako jeden systém, s ktorým vie interagovať. V dôsledku toho musí byť prepojený funkčnou sieťou. Zmyslom distribuovaných systémov je túto sieť spravovať, efektívne riadiť a koordinovať prácu všetkých pripojených zariadení a zabezpečiť požadovaný výsledok, respektíve pri danej úlohe taktiež koordinovať používanie zdieľaných zdrojov.

V prípade distribuovaného systému nie je dôležité, či sa zariadenia nachádzajú v jednej budove alebo ich delia celé mestá, krajiny, či dokonca svetadiely - svoj účel si plnia bez ohľadu na ich geografickú lokáciu.

### 1.1.1 Znaky distribuovaných systémov

Distribuované systémy sú vo svojej podstate priamym dôsledkom pokroku v oblasti informačných technológií, kde potreba zvyšovať výkon, zlepšiť spoľahlivosť systémov a rozšíriť možnosti komunikácie medzi jednotlivými zariadeniami bola spočiatku naštartovaná ako prostriedok pre rozšírenie a zrýchlenie možností procesov vykonávaných pri vedecko-technických výskumoch a výpočtových operáciách. Následne s rozšírením osobných počítačov sa táto technológia rozšírila aj

na ďalšie oblasti s mnohostranným využitím. Medzi konkrétne formy využitia podľa [1] patria:

- Využitie pri osobných počítačoch (komunikácia, záloha, zdieľaný výkon)
- Distribuovaný systém riadenia v priemyselných odvetviach, pri systémoch riadiacich roboty, stroje a automatizované pracovné linky
- V bankových systémoch na prepojenie jednotlivých zariadení (počítačov, bankomatov)
- Distribuované spracovanie dát vo firmách (distribuované databázy) - pri súčasnej práci niekoľkých užívateľov databázy sa zmeny v dátach realizujú paralelne a integrita dát musí byť potom zabezpečená práve distribuovaným riadiacim mechanizmom [2]
- Riadenie letovej prevádzky, jadrových reaktorov apod.
- Internet (v súčasnosti predovšetkým ako World Wide Web (WWW), ako však zdôrazňuje Coulouris [3], redukovať Internet iba na platformu WWW nie je korektné vzhľadom k tomu, že Internet poskytuje aj služby iných platforiem, napr. transfer súborov - FTP alebo email - SMTP, POP3)
- V poslednej dobe, ako dodáva Klimeš [2], sa problematika distribuovaných aplikácií objavuje i v oblasti tzv. neurónových sietí, kde sa správanie každého neurónu modeluje stavovým automatom, pričom prepojenie všetkých parciálnych neurónov predstavuje veľmi efektívny distribuovaný systém

Pri budovaní a správe distribuovaných systémov sa vyskytujú isté požiadavky a problémy, s ktorými je potrebné sa zaoberať a adekvátne ich riešiť. Jedna z najdôležitejších výziev je zaručiť bezpečnosť daného systému proti neoprávnenému prístupu. Ďalej je nutné vhodne riadiť komunikáciu a prístupnosť siete a chrániť ju pred preťažením. Nevýhodou je, že distribuované systémy sú často zložitejšie ako centralizované, a preto je ich riadenie principiálne náročnejšie. Medzi najdôležitejšie vlastnosti distribuovaných systémov podľa Klimeša [2] patria:

- Transparentnosť
  Transparentnosť v distribuovaných systémoch funguje presne naopak, ako naznačuje slovo transparentnosť. Princípom je zakryť pred používateľom jednotlivé súčasti procesov, ktoré vykonáva distribuovaný systém, pričom celé sa to má javiť ako jeden celok.

Klimeš rozlišuje Prístupová transparentnosť, kedy proces má rovnaký prístup k lokálnym i vzdialeným prostriedkom. Lokačná transparentnosť spočíva v tom, že užívateľ (proces) nevie povedať kde sú jednotlivé prostriedky umiestnené. Podľa princípu Migračnej transparentnosti sa prostriedky môžu premiestňovať medzi jednotlivými počítačmi. Exekučná transparentnosť znamená, že procesy môžu bežať na ľubovoľnom procesore zapojenom do systému. Podľa Replikačnej transparentnosti si užívateľ nemôže povedať, koľko kópií daného objektu existuje. Na základe Konkurenčnej transparentnosti prostriedky môžu byť automaticky využívané zároveň niekoľkými užívateľmi. A nakoniec, posledná forma - Paralelizmová transparentnosť - znamená, že rôzne činnosti môžu byť vykonávané paralelne bez vedomia užívateľa.

- Autonómia
  Každý počítač je potenciálne schopný samostatnej funkčnosti jednak podľa princípu decentralizovaného riadenia a rozhodovania, pri ktorom každé zariadenie vykonáva rozhodnutie nezávisle na ostatných, jednak podľa princípu migrácie procesov a prostriedkov, kedy procesy aj prostriedky môžu byť premiestnené na iný počítač a jednak podľa princípu vyvažovania výpočtovej záťaže, kde jednotlivé úlohy môžu byť premiestnené na menej vyťažený procesor.

- Spoľahlivosť
  Základnou ideou zvýšenia spoľahlivosti je to, že v prípade výpadku niektorých komponentov systému si ostatní rozdelia ich prácu.

- Výkonnosť
  Konkrétny beh určitej aplikácie v distribuovanom systéme by nemal byť výrazne pomalší ako v klasickom jednoprocesorovom systéme rovnakej triedy. Na dosiahnutie zodpovedajúcej rýchlosti behu aplikácie je však potrebný výkonnejší hardvér. Spomalenie je spôsobené jednak pomalším prenosom správ po sieti, jednak zložitejším softvérom. Pri distribuovaných systémoch ale platí synergický efekt - celok je vždy viac ako súčet jeho častí. Týka sa to, podľa Milojicica [4], nielen výpočtovej efektivity (čo v konečnej miere kompenzuje uvedené sieťové a softvérové spomalenie), ale i nižších nákladov na používanie, servis a údržbu systémov.

- Súbežnosť
  Zariadenia spracúvajú požiadavky a následne ich vykonávajú paralelne,

spolupracujú, pričom sa využíva vyšší výkon a celkový synergický efekt.

- Rozšíriteľnosť
  Systém možno kedykoľvek rozšíriť o ďalšie prvky, ďalšie zariadenia. Existujú však prípady, keď je možné centralizáciu využiť. Ide o tzv. Centralizované komponenty (napríklad jednotný mail server pre všetkých užívateľov.

- Tolerancia chýb
  Schopnosť pokračovať v činnosti po chybe. V klasickej centrálne riadenej sústave s jedným vedúcim procesorom dôjde v prípade jeho zlyhania k úplnému zrúteniu celého systému, čomu sa musí v distribuovanom systéme zabrániť. Cieľom je preto navrhnúť systém pomocou vhodných algoritmov tak, aby boli požiadavky vykonávane mnohými prepojenými procesormi súčasne, pri zachovaní vysokého výkonu.

Ďalším znakom distribuovaných systémov je tiež absencia zdieľanej pamäte, z čoho plynie, že je potrebne zmeniť spôsob zdieľania informácií. To sa zabezpečuje posielaním správ medzi jednotlivými zariadeniami. Významným determinujúcim znakom je absencia globálneho časového mechanizmu. Namiesto toho sú zavedené logické hodiny, ktoré funkciu synchronicity zabezpečujú. Vďaka nim má systém prehľaď o jednotlivých udalostiach a poradí, v ktorom sa stali, a môže na to patrične reagovať.

### 1.1.2 Nevýhody distribuovaných systémov

Napriek nesporným výhodám spomína Klimeš [2] aj isté nevýhody, ktoré distribuované systémy majú:

- nedostatok koordinačného softvéru a veľké požiadavky na hardvér,
- možnosť vzniku problémov s pripojením do sieti, keďže sieť môže byť zahltená alebo dokonca nedostupná,
- bezpečnostné výzvy, keďže relatívne jednoduchý prístup k dátam vyvoláva nutnosť šifrovania a v nevyhnutných prípadoch aj utajenia.

### 1.1.3 Architektúry distribuovaných systémov

Vyššie popísané výhody sú vlastné všetkým typom architektúr, ktoré popíšeme v tejto kapitole.

### Centralizovaná architektúra (Klient-Server)

Klient/server architektúra pracuje tak, že klient posiela požiadavky a server ich spracuje a vykoná. Takto prebieha ich vzájomná komunikácia zakaždým, keď sa odošle požiadavka. Tento model je jeden z najrozšírenejšie využívaných v praxi, jeho výhodou je jednoduchá obsluha z hľadiska oboch strán, teda administrátora siete, ale aj z hľadiska používateľa. Na druhej strane, nevýhodami danej centralizácie je absolútna závislosť na príslušnom serveri, nedostupnosť v prípade výpadku siete a možné preťaženie siete pri zvyšovaní počtu požiadaviek klientov.

Rozšírením modelu Klient/server je takzvaný trojvrstvový model, skladajúci sa z prezentačnej (používateľské rozhranie), aplikačnej (vykonáva spracovanie dát) a dátovej vrstvy (zahŕňa databázový server).

Využitie trojvrstvového modelu uvádza van Steen [1] v podobe elektronického obchodu - e-shopu. Web server (prezentačná vrstva) odosiela požiadavky užívateľa na aplikačný server (aplikačná vrstva), kde sa požiadavka spracuje a realizuje vo forme dopytu na databázový server (dátová vrstva) týkajúcu sa produktov e-shopu. Výsledok spracovanej požiadavky spätne web server prezentuje užívateľovi.

### Decentralizovaná architektúra (Peer-to-Peer)

Peer-to-peer alebo tiež P2P je decentralizovaný systém, ktorý nevyžaduje žiadne ústredné riadenie na správu svojho chodu a správu systémových zdrojov [1]. V takom prípade spolu môžu klienti komunikovať priamo. Vo svojej najčistejšej podobe sú v P2P architektúre všetci klienti rovnocenní, každý zastupuje úlohu klienta aj servera súčasne a každý klient má svoju vlastnú kópiu dát. Server v tomto prípade neexistuje. Pravé títo používatelia, respektíve klienti, udržujú sieť v chode. Okrem čistého P2P systému existuje aj systém hybridný (viď nižšie). Jednou zo základných výhod P2P sietí je, že s rastom počtu používateľov sa zvyšuje celková rýchlosť prenosu, zatiaľ čo pre model klient-server musia používatelia zdieľať konštantnú kapacitu servera, takže priemerná rýchlosť s rastom používateľov klesá.

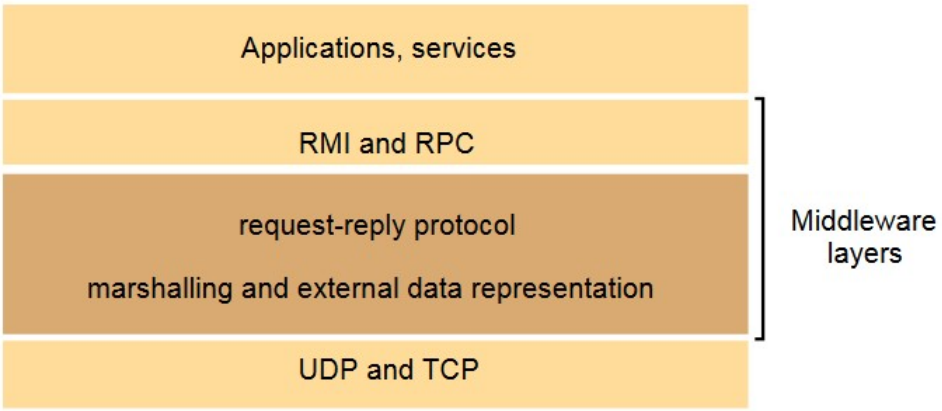
V práci [5] vo svojej definícii Peer-to-Peer systémov tiež vyzdvihuje samoorganizujúcu (self-organizing) funkciu navzájom rovnocenných, autonómnych entít (peers), ktorá sa zameriava na zdieľané využívanie distribuovaných zdrojov v sieťovom prostredí, a to bez potreby centrálnych (serverových) služieb. V stručnosti ide o systém s kompletne decentralizovanou samoorganizáciou a decentralizovaným využitím zdrojov.

Medzi výhody sietí P2P oproti klient-serverovým riešeniam vyzdvihuje Kumar [6] napríklad *symetrickú povahu uzla* (každý uzol môže pracovať ako klient alebo server), *vyššia škálovateľnosť* (počet zariadení nie je obmedzený), *výkonnostná heterogenita* (môžu byť použité výkonovo odlišné zariadenia), *nižšia citlivosť voči útokom*, *vyššia dynamika* (efektívne zvládajú dynamické zmeny topológie) a pod.

### 1.1.4 Komunikácia v distribuovaných systémoch

Komunikácia entít v distribuovaných systémoch môže byť realizovaná jednak technológiami známymi z lokálnych sietí, no rovnako aj technológiami používanými pri systémoch MAN alebo WAN [7]. Špecifickým, pri rozľahlých sieťach WAN, ako dodáva Klimeš [2], je to, že sa na spojenie na veľké vzdialenosti využívajú dvojbodové spojenia, ktoré sa prepájajú pomocou smerovačov (routerov) do polygonálnych sietí, kým LAN siete sa opierajú o zdieľaný komunikačný kanál.

Medzi dva základné programové rozhrania [7] na komunikáciu radíme *systémy posielania správ* (používanie modifikovaných funkcií systému manipulácie so súbormi), *volanie vzdialených podprogramov RMI a RPC* (nadstavba nad systémom posielania správ) [3]



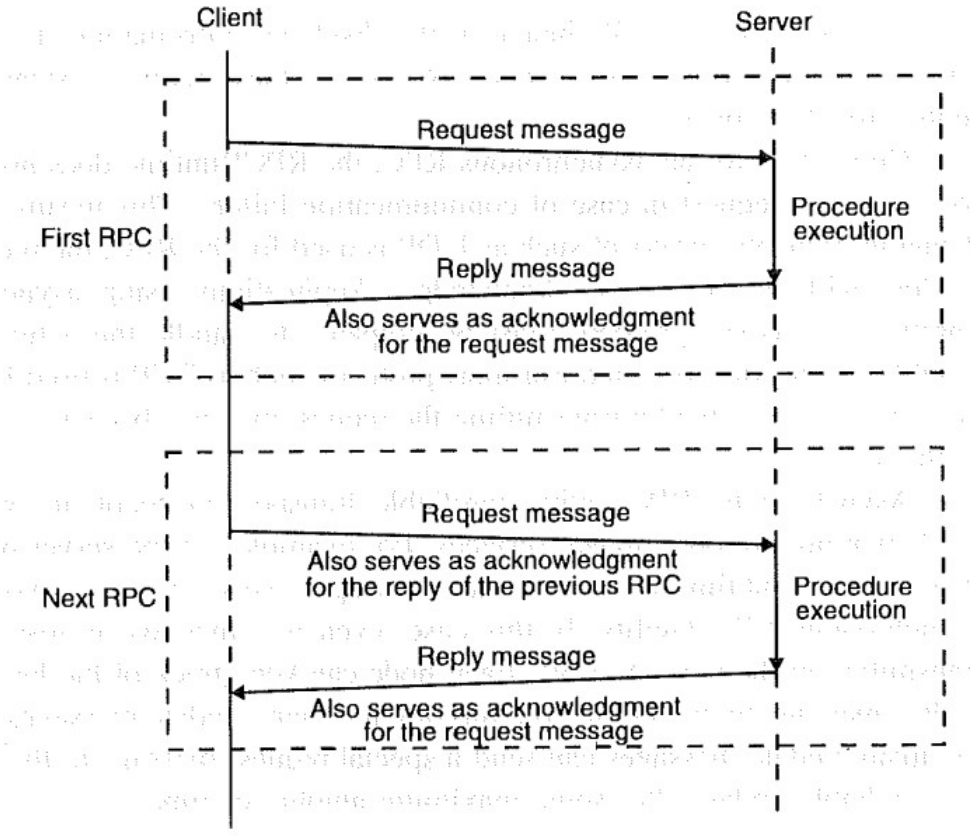

Obr. 1.1: Vrstvy middlewaru

RMI a RPC sú súčasťou tzv. middlewaru (viď obrázok 1.1 publikovaný podľa [3]). Middleware je vrstva distribuovaného softvéru, ktorá stojí nad sieťovým operačným systémom a súčasne pod aplikačnou vrstvou. Poskytuje integrované distribuované prostredie, ktorého úlohou je zjednodušenie programovania a manažovania distribuovaných aplikácií, a tiež poskytovanie rôznych ďalších služieb s pridanou hodnotou ako napríklad transakcie. Middleware, ako dodáva Mahmoud [8], je o integrácii a interoperabilite aplikácií a bežiacich služieb na hetero-

génnych počítačových a komunikačných zariadeniach.

Na nižšej úrovni middlewaru sa nachádza protokol Request/Reply (RR), ktorý je podľa [9] určený pre dizajn systémov zahrňujúcich jednoduchý kód RPC. Tento jednoduchý (simple RPC) je charakteristický tým, že všetky jeho argumenty a rovnako aj výsledky sa zmestia do jedného paketového bufferu, a tiež, že trvanie volania (call) a intervaly medzi jednotlivými volaniami sú krátke. Protokol je založený na myšlienke používania implicitných potvrdení, ktorými sa eliminuje potreba správ s explicitným potvrdením.

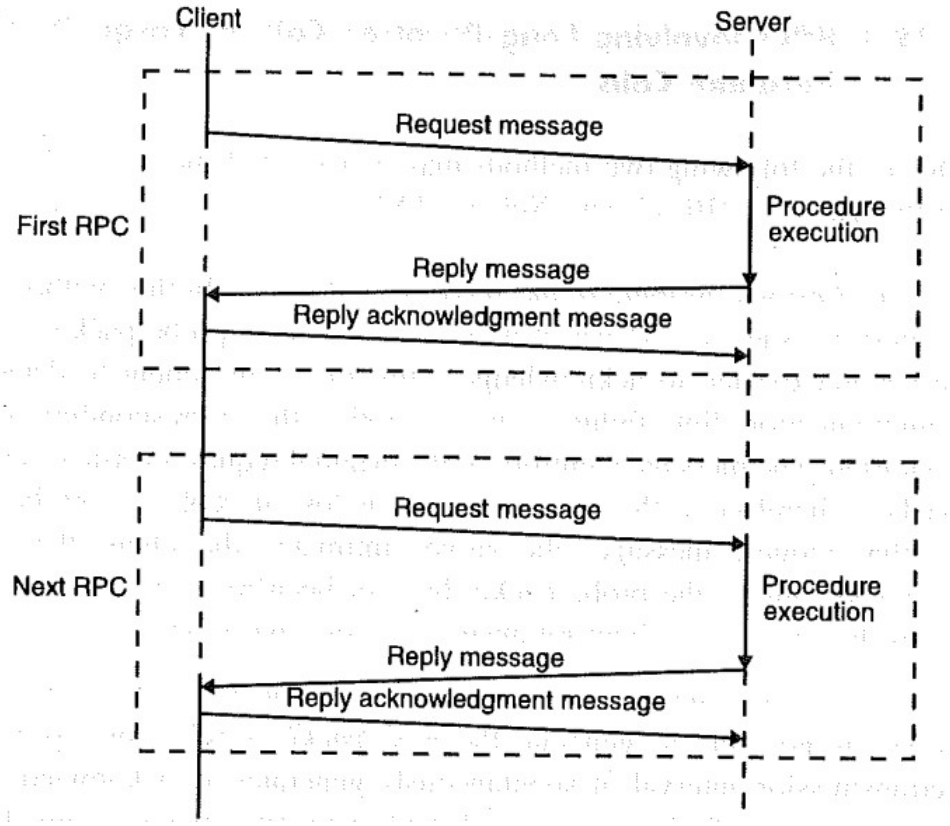
Preto v tomto protokole správa odoslaná serverom ako odpoveď (*reply*) na volanie klienta je považovaná za potvrdenie jeho požiadavky (*request*) a následný volací paket zo strany klienta je považovaný za potvrdenie tejto odpovednej (*reply*) správy zo servera. RR komunikáciu medzi klientom a serverom prezentuje obrázok 1.2 prevzatý z [9].




Obr. 1.2: Request/reply protokol

Jednoduchá implementácia RR protokolu však vyžaduje, aby server ukladal záznamy o odpovediach vo svoje cache. No v situácii, kedy server obsluhuje veľké množstvo klientov, musí server tiež uchovávať veľké množstvo informácií o zaslaných odpovediach (*replies*). V niektorých implementáciách preto server obmedzí množstvo uchovávaných informácií (uchovávané dáta pravidelne maže)

- to niekedy môže spôsobiť stratu informácií o odpovediach, ktoré ešte neboli úspešne doručené klientom. Na zvládnutie tohto problému existuje riešenie v podobe Request/Reply/Acknowledge-Reply (RRA) protokolu. Ten požaduje od klientov, aby finálne potvrdili serveru doručenie odpoveďových správ od neho. Server následne maže informácie zo svojej cache iba po definitívnom prijatí potvrdenia zo strany klienta. RRA tak vyžaduje prenos troch správ na jedno volanie (dve zo strany klienta a jedno zo strany servera). RRA komunikáciu medzi klientom a serverom prezentuje obrázok 1.3 prevzatý z [9].




Obr. 1.3: Request/reply/acknowledge-reply protokol

Tzv. marshalling je podľa Henrikssona [10] prevedenie internej dátovej reprezentácie (napr. objektu) na externú reprezentáciu údajov (napr. text). Opačným procesom je tzv. unmarshalling, čiže prevedenie externých dát na internú reprezentáciu. O marshalling aj unmarshalling sa stará middleware bez zapojenia aplikačnej vrstvy.

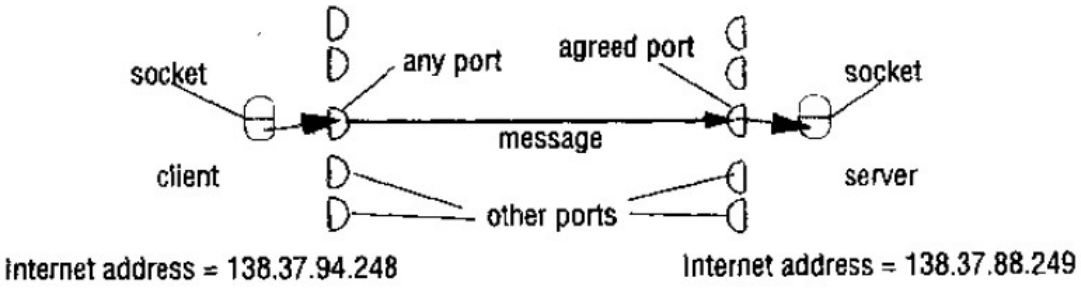
### Posielanie správ

Posielanie správ (*message passing*) je v distribuovaných systémoch formou komunikácie dvoch procesov. Zasielanie správ medzi dvoma procesmi sa realizuje pomocou dvoch základných operácií: *send* a *receive*. Jeden proces posiela správu

(*send*) ako sekvenciu bajtov smerom k cieľu (adresátovi), kde ju príjme lokálny prijímač (*receiver*).

Komunikačný protokol vyžaduje, aby zaslaná správa obsahovala Internetovú adresu a port. Na tomto porte prijíma správy vždy jeden receiver, ktorý však slúži pre príjem správ od viacerých odosielateľov. Procesy môžu využívať viacero vopred definovaných portov (a teda viacero receiverov). Každý proces, ktorý pozná číslo príslušného portu, môže na tento port zaslať správu [3].

Samotné správy sú vytvárané podľa princípov tzv. socketového rozhrania, ktorého účelom je štandardizovať zasielanú správu a jej parametre. Socket je pre oba procesy (vysielací aj prijímací) komunikačným koncovým bodom (*end point*), do ktorého jeden proces zapíše dáta, ktoré majú byť zaslané pomocou siete a na opačnej strane komunikácie druhý proces môže tieto dáta prečítať [1]. Princíp prenosu je znázornený na obrázku 1.4 nižšie.




Obr. 1.4: Význam socketov a portov

Operácie vykonávané v komunikačnom režime posielania správ je podľa [7] možné deliť podľa rôznych kritérií do niekoľkých skupín. Niekoľko z nich sú napríklad *blokované/neblokované operácie* (synchrónne, resp. asynchrónne) (proces, ktorý odoslal dáta je blokovaný), *s využitím vyrovnávacej pamäte/bez využitia vyrovnávacej pamäte*, *spoľahlivý/nespoľahlivý komunikačný protokol* (spojené s väčšou réžiou na úrovni komunikačného programového vybavenia), *pevná/premenná dĺžka správy*, *priama/nepriama komunikácia*, *mapovanie adries* (môže byť vykonávané osobitným protokolom alebo tabuľkou dvojíc meno - adresa), *randevous* (synchrónne prenosy správ medzi procesmi).

Skupina s využitím vyrovávacej pamäte obsahuje správy, ktoré môžu byť v rôznych lokáciách. Správa sa môže nachádzať v **pamäti odosielateľa** (odosielateľ musí vyčleniť pamäť, vhodné pre synchrónne prenosy), **v pamäti komunikačného programového vybavenia odosielateľa** (dáta sú prenesené do systémovej vyrovnávajúcej pamäti), **v pamäti komunikačného programového vybavenia príjemcu** (dáta sú prenesené do komunikačného uzla príjemcu), **v pamäti príjemcu** (dáta presunuté do vyrovnávacej pamäti príjemcu) alebo **niekde v sys-**

**téme** (nepoužíva sa pre prenos medzi procesmi).

Príkladom systému komunikujúceho prostredníctvom posielania správ je programové vybavenie (knižnica) operačného systému UNIX s rozhraním TLI alebo socketov. Podporované volania sú si v oboch uvedených systémoch veľmi podobné.

#### RPC a RMI

Základnou myšlienkou volania vzdialených podprogramov (RPC - remote procedure call) je použitie mechanizmu volania podprogramov tak, ako je známe z mnohých programovacích jazykov pre sekvenčné úlohy. V najjednoduchšom prípade je pri RPC volajúci proces pozastavený v exekúcii, parametre volania uložené do správy, a tá je prenesená do vzdialeného uzla. Tu je vykonané vyvolanie požadovaného podprogramu, výsledok volania uložený opäť do správy, a tá spätne prenesená do uzla volajúceho procesu. Pozastavený volajúci proces je po prijatí správy a spracovania parametrov opätovne aktivovaný tak, ako by volal lokálnu procedúru [7].

RPC je prvotný komunikačný mechanizmus pre distribuované programy. Je jednoduchý, a táto jednoduchosť robí RPC atraktívnym. Je ľahko pochopiteľný, dostatočne všeobecný a môže byť preto efektívne realizovaný v praxi.

RMI je skratka pre Remote Method Invocation, je podobný ako RPC, ale podporuje objektovo orientované programovanie [11]. Všetky rozdiely uvádzame podľa [11] v tabuľke 1.1.

### 1.1.5 Odolnosť proti chybám

Charakteristickou črtou distribuovaných systémov, ktorá ich odlišuje od systémov s jedným zariadením (single-machine systems), je fenomén rizika parciálneho zlyhania - časť systému „spadne“, no zvyšok distribuovaného systému pokračuje v práci, pravdepodobne korektne. Dôležitou súčasťou dizajnu distribuovaných systémov je snaha konštruovať systém takým spôsobom, aby sa dokázal automaticky obnoviť (recover) z parciálneho zlyhania bez toho, aby zlyhanie malo vplyv na funkčnosť, spoľahlivosť a efektivitu celého distribuovaného systému. V prípade objavenia sa chyby by systém mal pokračovať vo svojej činnosti očakávaným spôsobom, až pokiaľ sa vykonajú nevyhnutné opravy - distribuovaný systém teda má byť tolerantný k chybám (fault tolerant), ako uvádza Van Steen [1].

Tabuľka 1.1: Rozdiel medzi RPC a RMI

| Rozdiel medzi RPC a RMI | |
|---|---|
| RPC | RMI |
| Platforma závislá od knižnice a OS | Platforma Java |
| Podporuje procedurálne programovanie | Podporuje objektovo orientované programovanie |
| Menej efektívne v porovnaní s RMI | Efektívnejšie ako RPC |
| Vytvára viac režijných nákladov | Vytvára menej režijných nákladov |
| Parametrami, ktoré sa odovzdávajú v RPC, sú bežné údaje | V RMI sa ako parametre odovzdávajú objekty |
| RPC je staršia verzia RMI | Nástupnícka verzia RPC |
| Jednoduchosť programovania v RPC | Vyššia zložitosť programovania v RMI |
| RPC neposkytuje žiadnu úroveň zabezpečenia | Poskytuje zabezpečenie na úrovni klienta |
| Náklady na vývoj sú obrovské | Náklady na vývoj sú primerané |
| Veľkým problém s určením verzie | Verzie sú ľahko identifikovateľné |
| Na jednoduchú aplikáciu v RPC je potrebných viacero kódov | Nie je potrebných viacero kódov na jednoduchú aplikáciu v RMI |

**Paxos**

Požiadavky na konsenzus sú: Môže sa zvoliť len spomedzi navrhovaných hodnôt, volí sa iba jedna hodnota a proces sa nikdy nedozvie, že hodnota bola zvolená, pokiaľ sa skutočne jedna nezvolí. Algoritmus sa skladá z troch fáz, ktoré sa môžu opakovať (v dôsledku porúch):

- Zvoľte repliku za koordinátora.
- Koordinátor vyberie hodnotu a vysiela ju všetkým replikám v správe s názvom prijať správu. Ostatné repliky buď potvrdia túto správu, alebo ju odmietnu.
- Keď väčšina replík uzná koordinátora, dosiahol sa konsenzus a koordinátor vysiela správu o úspešnom procese s cieľom upozorniť repliky.

Jednoduchý spôsob, ako implementovať distribuovaný systém je ako kolekcia klientov, ktoré vydávajú príkazy na centrálny server. Server môže byť opísaný ako deterministický stavový počítač, ktorý vykonáva príkazy klienta v určitom poradí. Implementácia, ktorá používa jeden centrálny server zlyhá, ak zlyhá tento

server. Aby sme zaručili, že všetky servery vykonávajú rovnakú postupnosť príkazov centrálneho počítača, implementujeme postupnosť samostatných inštancií algoritmu konsenzu Paxos, hodnota zvolená i-tou inštanciou je príkaz centrálneho počitača. Každý server hrá všetky roly v každej inštancii algoritmu.

### 1.1.6 Bezpečnosť v distribuovaných systémoch

Bezpečný systém v komunikačných systémoch by mal poskytovať niekoľko záruk ako je dôvernosť, integrita a dostupnosť údajov, ich pravosť a nepopierateľnosť. Môže tiež vzájomne overovať odosielateľov a príjemcov [12].

#### Typy útokov na sieťovú komunikáciu

Dva hlavné typy útokov, ktoré sa protivník môže pokúsiť vykonať v sieti [12] sú **pasívne** (počúva segment siete a pokúša sa čítať citlivé informácie) a **aktívne útoky** (vydáva sa za klienta alebo server a zachytáva komunikáciu počas prenosu).

Ochrana dôvernosti a integrity, ktorú ponúkajú kryptografické protokoly, ako SSL/TLS, môže chrániť komunikáciu pred škodlivým odpočúvaním a manipuláciou. Ochrana pravosti poskytuje istotu, že používatelia skutočne komunikujú so systémami tak, ako bolo zamýšľané.

#### Bezpečnostné ciele v distribuovaných systémoch

Bezpečnosť v distribuovaných systémoch môže byť podľa Van Steena [1] rozdelená do dvoch častí:

Jedna časť sa zameriava na komunikáciu medzi užívateľmi, ktorí sú pravdepodobne dislokovaní na viacerých zariadeniach. Principiálnym otázkou je tu bezpečná komunikácia prostredníctvom zabezpečeného komunikačného kanála, presnejšie - mechanizmov autentifikácie, integrity zasielaných správ a nevyhnutnosť zachovania dôvernosti.

Druhá časť sa zameriava na autorizáciu, ktorá zahŕňa mechanizmus dostatočného zabezpečenia, komunikujúci proces dostáva iba také prístupy k zdrojom dát, na ktoré je kompetentný a oprávnený.

Podľa [13] je prakticky nemožné zaručiť absolútnu zabezpečenú peer-to-peer sieť, ani sa to nedá urobiť s ohľadom na iné procesy a technológie. Existujú však spôsoby, ako znížiť svoje riziká, a hlavnými z nich je byť opatrný pri otvorených údajoch a starostlivo analyzovať možnú prítomnosť podvodu.

Kľúčovým problémom pre zabezpečenie najmä P2P sietí je, že kvôli ich prirodzenej decentralizovanej povahe chýbajú prostriedky na centrálnu správu, a teda kontrolu, potrebné na boj proti bezpečnostným útokom (Friedman, in [14]).

Aj podľa [15] môžu byť siete typu peer-to-peer z hľadiska bezpečnosti veľmi nebezpečné - zvyčajne si takáto komunikácia totiž vyžaduje otvorenie jedného alebo viacerých portov pre prenos súborov. Problém je v tom, že sa takto nedá spoľahlivo kontrolovať, čo ide dovnútra a von z týchto otvorených portov. Sú ako „otvorené dvere“, ktorými si používatelia siete P2P umožnili prístup k počítaču. Existujú určité obmedzenia prístupu, ktorý môžu mať ostatní používatelia siete, ale tieto otvorené porty sa napriek tomu môžu stať ľahkým vstupným bodom pre útočníkov, ktorí sa snažia získať prístup k počítaču alebo sieti, a to bez vedomia užívateľa.

Množstvo poskytovateľov internetových služieb (ISP) blokuje alebo obmedzuje súvisiaci prenos rôznych P2P systémov využívajúc jednoduchý a súčasne rýchly prístup známy ako filtrovanie portov [16] .

Nejde o univerzálny prístup, keďže na P2P komunikáciu sa dajú využiť porty spojené s inými službami (webový prenos, e-mailová prevádzka, atď.). Nedávne štúdie dokonca naznačujú, že P2P pripojenia sú teraz distribuované cez všetky porty.

Niektorí poskytovatelia používajú na vyhľadávanie P2P komunikácie pokročilé heuristické filtrovanie: Skúmaním užitočného zaťaženia každého paketu sa dá určiť aplikácia, ktorá ho vygenerovala. Bohužiaľ, heuristické filtrovanie má tendenciu k vysokej miere falošnej pozitivity a falošnej negativity, je tiež náročné na údržbu a výpočtovú kapacitu. Naviac, niektorí P2P klienti dokážu šifrovať prenos P2P a zakryť tak signatúry, ktoré heuristika používa na identifikáciu P2P údajov [16].

**Zabezpečenie bezpečnosti pri vývoji a využívaní P2P aplikácií**

Základné prístupy k bezpečnosti P2P systémov sú triviálne [15]:

- Obmedzenie zdieľaných informácií (súborov a priečinkov) - vo všeobecnosti si užívateľ musí sám starostlivo strážiť, čo sa zdieľa.
- Používanie kvalitných antivírusových riešení.
- Správne nakonfigurovaný firewall pre službu P2P.

No hlavnou požiadavkou pre akékoľvek netriviálne P2P aplikácie je predovšetkým ochrana výmeny dát medzi partnermi. Podrobnosti ochrany závisia od

toho, ako sa aplikácia používa a čo presne potrebuje chrániť. Existujú tri základné spôsoby, ako zaistiť netriviálnu bezpečnosť P2P komunikácie [13]:

- Zabezpečené spojenie pomocou SSL, resp. TLS,
- Ochrana identity (Identity protection),
- Blockchain.

#### Secure Sockets Layer (SSL)

SSL je bezpečnostný protokol, ktorý zabezpečuje súkromie pri interakcii so sieťami, ako je internet. SSL umožňuje aplikáciám interagovať bez rizika špehovania a vonkajšej intrúzie. Keď chce aplikácia (klient) interagovať s inou aplikáciou (serverom), klient otvorí soketové pripojenie k serveru. Klient a server vytvoria zabezpečené spojenie. Počas tejto interakcie sa server identifikuje klientovi. V prípade potreby sa klient môže identifikovať aj na serveri. Po dokončení autentifikácie a vytvorení zabezpečeného pripojenia môžu obe aplikácie navzájom bezpečne interagovať [13].

#### Transport Layer Security (TLS)

Transport Layer Security, je široko používaný bezpečnostný protokol navrhnutý na uľahčenie ochrany súkromia a bezpečnosti dát pri komunikácii cez internet, pričom je nástupcom SSL. Primárnym prípadom použitia TLS je šifrovanie komunikácie medzi webovými aplikáciami a servermi, ako sú webové prehliadače načítavajúce webovú stránku. TLS možno použiť aj na šifrovanie inej komunikácie, ako je e-mail, správy a hlas cez IP (VoIP) [17].

#### Ochrana identity

Na autentifikáciu klientov webové servery zvyčajne používajú najjednoduchší spôsob - kontrolu pomocou hesla - a nepoužívajú SSL. No SSL je možné použiť rovnako na autentifikáciu servera ako na autentifikáciu klienta pomocou podobného mechanizmu, a podľa [13] je rozhodne vhodnejšie použiť túto transparentnú autentifikáciu klienta a servera uskutočnenú medzi partnermi práve pomocou SSL.

#### Blockchain ako nástroj na zaistenie bezpečnosti P2P

Technológia blockchain je teraz veľmi populárna, nielen preto, že sa používa ako základná technológia na vytváranie a prevádzkovanie kryptomien, ale aj ako ná-

stroj, ktorý umožňuje bezpečne vykonávať online transakcie.

Ako už názov napovedá, blockchain je séria blokov, ktoré zapisujú údaje o blockchaine brokera do hash s odkazom na predchodcu bloku. Bloky sú uložené anonymne všetkými zainteresovanými stranami, čo eliminuje centralizované zraniteľné miesta obchodovania s kryptomenami, ktoré využívajú podvodníci.

Každý uzol P2P siete má svoju vlastnú databázu v reálnom čase, ktorá neustále aktualizuje údaje. Každý z uzlov v sieti ponúka svoj vlastný blok v reťazci. Ostatné uzly, ktoré sú v sieti, skontrolujú cudzí blok porovnávaním s tým, ktorý majú v datacaste. Ak všetky uzly siete akceptujú blok, potom sa zaznamená v reťazci.

V rámci blockchainovej siete sa operácie vykonávajú na základe tzv. smart kontraktov.

### Kryptografia

Kryptografia poskytuje bezpečnú komunikáciu pri potenciálnej prítomnosti škodlivých tretích strán. Šifrovanie používa špeciálne navrhnutý šifrovací algoritmus a šifrovací kľúč na transformáciu vstupu (t. j. čistého textu) na zašifrovaný výstup (t. j. šifrovaný text). Daný algoritmus vždy transformuje rovnaký otvorený text na rovnaký šifrový text, ak sa použije rovnaký kľúč. Algoritmy sa považujú za bezpečné, ak útočník nedokáže určiť žiadne vlastnosti otvoreného textu alebo kľúča vzhľadom na šifrovaný text. Útočník by nemal byť schopný určiť nič o kľúči vzhľadom na veľký počet kombinácií otvoreného textu/šifrovaného textu.

### Symetrická kryptografia

Pri symetrickej kryptografii sa na šifrovanie aj dešifrovanie používa rovnaký kľúč. Obe participujúce strany - aj odosielateľ aj príjemca - musia mať zdieľaný kľúč, ktorý obaja poznajú. Distribúcia kľúčov je zložitý problém a bola impulzom pre vývoj asymetrickej kryptografie [12].

Symetrický systém je vo všeobecnosti veľmi rýchly a ideálny na šifrovanie veľkého množstva údajov (napr. celého diskového oddielu alebo databázy).

### Asymetrická kryptografia

Pri asymetrickom šifrovaní sa na šifrovanie a dešifrovanie používajú dva rôzne kľúče. Každý používateľ v asymetrickom kryptosystéme má verejný aj súkromný kľúč. Súkromný kľúč je vždy udržiavaný v tajnosti, ale verejný kľúč môže byť voľne distribuovaný.

Dáta zašifrované verejným kľúčom možno dešifrovať iba zodpovedajúcim súkromným kľúčom - odoslanie správy osobe X vyžaduje zašifrovanie tejto správy verejným kľúčom osoby X. Správu môže dešifrovať iba prijímateľ - osoba X, pretože iba osoba X má svoj súkromný kľúč, ktorý na dešifrovanie môže ako jediná použiť.

Asymetrický je oveľa pomalší ako symetrický a dokáže šifrovať iba časti údajov, ktoré sú menšie ako veľkosť kľúča (zvyčajne 2048 bitov alebo menej). Asymetrické šifrovanie sa teda vo všeobecnosti používa na šifrovanie symetrických šifrovacích kľúčov, ktoré sa potom používajú na šifrovanie oveľa väčších blokov údajov. V prípade digitálnych podpisov sa asymetrické šifrovanie vo všeobecnosti používa na šifrovanie hashov správ, a nie celých správ. Kryptosystém poskytuje správu kryptografických kľúčov vrátane generovania, výmeny, ukladania, používania, odvolania a výmeny kľúčov.

### 1.1.7 Synchronizácia času

Presný čas v sieti je, ako zdôrazňuje Scarpati [18], dôležitý z mnohých dôvodov. Aj distribuované procedúry závisia od koordinovaných časov, a to preto, aby sa zabezpečilo dodržanie správnych sekvencií. Bezpečnostné mechanizmy závisia rovnako od dôsledného udržiavania času v celej sieti. Podobne rôzne aktualizácie súborového systému vykonávaného viacerými počítačmi závisia aj od synchronizovaných časov hodín. Systémy sieťovej akcelerácie a správy siete sa tiež spoliehajú na presnosť časových pečiatok pri meraní výkonu a odstraňovaní problémov.

V centralizovaných systémoch je "veliteľským"časom systémový čas servera, ktorý je klientmi jednoznačne referencovaný. Ak bežiaci proces potrebuje vedieť čas, jednoducho vytvorí funkcionálne volanie na operačný systém. Ak proces A požiada o informáciu o čase a o niečo neskôr proces B tiež požiada o čas, je zaručené, že hodnota času určená procesu B bude vyššia (alebo teoreticky rovnaká, nie však vyššia) ako hodnota času určená procesu A [1].

Situácia je odlišná v decentralizovaných a hybridných systémoch. Pri P2P architektúre sú klienti rovnocenní, neexistuje tu funkčne nadradený systém (server) s referenčným časom. Daný problém je riešiteľný pomocou dvoch prístupov, a to synchronizáciou reálneho času alebo synchronizáciou logických hodín.

## 1.2 Dátová synchronizácia

Synchronizácia údajov je podľa Hansmanna [19] proces zosúladenia prenosu dát zo zdroja do cieľového ukladacieho priestoru a naopak, a tiež nepretržitá harmo-

nizácia dát v čase. Je základom pre širokú škálu aplikácií, vrátane synchronizácie súborov a synchronizácie mobilných zariadení. Jednou z najjednoduchších, ale prekvapivo bežných potrieb synchronizácie údajov, je zasielanie zmien zo servera na ľubovoľný počet klientskych zariadení. V týchto scenároch majú klienti z hľadiska údajov, ktoré synchronizujú, prístup iba na čítanie a každý klient synchronizuje spravidla identickú množinu údajov.

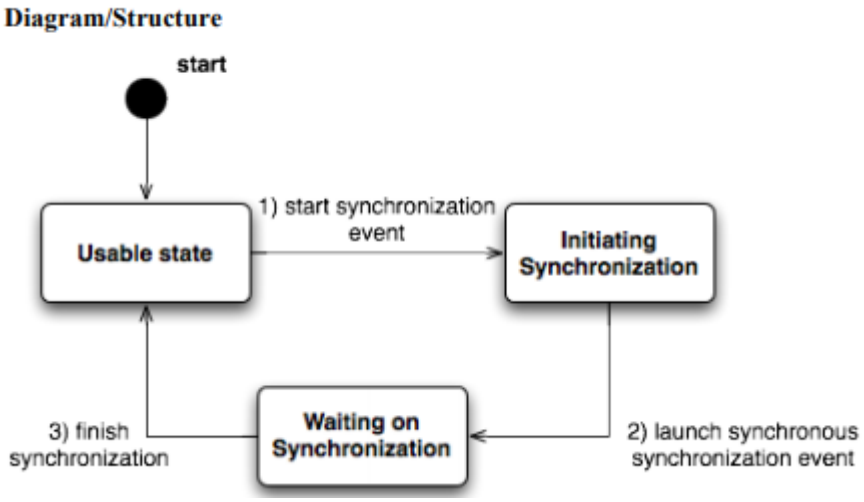
Vzory dátovej synchronizácie riešia, kedy by mala aplikácia synchronizovať údaje medzi zariadením a vzdialeným systémom (napríklad cloudovým serverom). Vývojári mobilných aplikácií musia preto zvážiť mnohé faktory a obmedzenia, vrátane dostupnosti siete, požiadaviek na aktualitu dát, komunikáciu s používateľským rozhraním, atď.

V procese vzájomnej synchronizácie dát sú v cente pozornosti predovšetkým problémy, ako napríklad zložitosť dátových formátov, skutočná aktuálnosť, bezpečnosť a kvalita dát a v neposlednom rade rýchlosť.

### 1.2.1 Synchrónna dátová synchronizácia

Synchrónna dátová synchronizácia je vykonávaná sekvenčným spôsobom, pričom aplikácia zostáva v stave čakania (t.j. nepoužiteľnosti pre užívateľa), kým synchronizácia nie je kompletne dokončená (či už s pozitívnym alebo negatívnym výsledkom), následne sa aplikácia vracia do použiteľného stavu.

Nasledujúci obrázok 1.5, ktorý demonštruje tento proces, citujeme opäť podľa McCormicka [20].




Obr. 1.5: Synchrónny návrhový vzor

K nesporným výhodám tohto riešenia patrí lepšie a jednoduchšie manažovanie synchronizácií bez rizika inkonzistencie dát, resp. vzniku iných mimoriad-

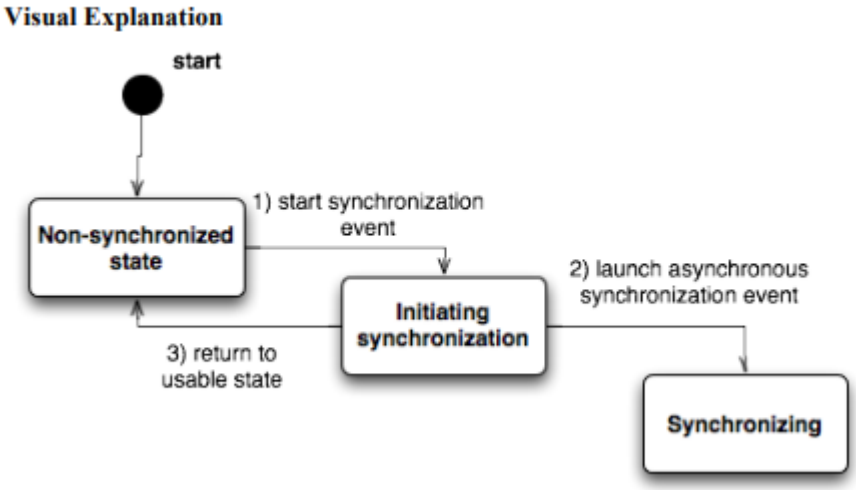
nych stavov spôsobených asynchrónnymi udalosťami.

Naopak, nevýhodou je blokovanie užívateľského rozhrania, a tým práce užívateľa s aplikáciou. Pre riziko zdĺhavej databázovej operácie môže aplikácia prestať reagovať promptne, resp. môže prestať reagovať vôbec. Môže byť preto praktickejšie vykonať prenos asynchrónne v inom než hlavnom vlákne, no zaobchádzať s týmto prenosom ako so synchrónnou akciou pomocou špeciálneho dialógového okna načítania, ktoré neblokuje používateľské rozhranie. Tento prístup má ďalšiu výhodu i v tom, že umožňuje v prípade potreby úplne zrušiť synchronizačnú udalosť.

### 1.2.2 Asynchrónna dátová synchronizácia

Keď je aplikácia v aktívnej prevádzke (čo v praxi znamená, že používateľ môže aktívne interagovať s touto aplikáciou), môže sa v aplikácii spustiť proces jej dátovej synchronizácie. Namiesto postupného spúšťania synchronizácie sa však tento proces spustí asynchrónne a aplikácia sa okamžite vráti do použiteľného stavu.

Nasledujúci obrázok 1.6, ktorý demonštruje tento proces, citujeme podľa McCormicka [20].




Obr. 1.6: Asynchrónny návrhový vzor

Synchronizačný prenos údajov sa aktivuje prostredníctvom iniciátora (spúšťača), napr. akciou používateľa, udalosťou časovača a pod.

Asynchrónna dátová synchronizácia má svoje výhody aj nevýhody. Výhody tohto typu synchronizácie sú nasledujúce:

- Užívateľ má prístup k funkcionalite aplikácie aj počas synchronizácie dát. Aplikácia je počas synchronizácie stále použiteľná, síce nebudú k dispozícii

najnovšie údaje, no v mnohých situáciách môže používateľ pracovať aj so zastaranými údajmi počas synchronizácie údajov.

- Synchronizácia údajov sa realizuje na pozadí.
  Kým aplikácia nie je práve využívaná používateľom, aplikácia môže synchronizovať dáta na pozadí, takže pri ďalšom otvorení (aktivácii) aplikácie sú dáte už aktuálne.

Na druhej strane, do úvahy je treba zobrať aj nasledujúce mínusy:

- Riziko nekonzistencie dát vyplývajúce zo súbežného prístupu viacerých zariadení k zdieľanému súboru údajov.
- Riziko zvýšených poplatkov za množstvo dát (v prípade, ak užívateľ hradí poplatky podľa množstva prenesených dát, pri paušálnych platbách ten aspekt nemá väčší význam).
- Množstvo dát môže zahltiť sieť.

### 1.2.3 Online-first approach

Online-first approach predstavuje pre pochopenie jednoduchý preferovaný synchronizačný princíp, resp. ideu, ktorá funguje v súlade s tým, na čo je bežný používateľ zvyknutý z praktického života: dáta sa sťahujú zo vzdialeného servera a následne sa prezentujú užívateľovi. Používateľ môže vidieť všetky údaje, upraviť ich a potom ich poslať späť na server.

Pre väčšinu ľudí je takmer vždy dostupné spoľahlivé pokrytie siete, čo je nevyhnutné, keďže mnohé aplikácie sa spoliehajú na neustály online prístup. V takomto prípade je popisovaný online-first approach, vyžadujúci si kontinuálne online pripojenie, optimálnym riešením.

Mnohé podniky však potrebujú, aby ich zamestnanci vykonávali svoju prácu aj keď je online pripojenie k dispozícii iba určitý čas alebo nie je dostupné vôbec. Napríklad rôzne práce počas núdzového stavu je potrebné vykonať aj v prípade výpadku elektriny alebo iných katastrofických scenárov, keď mobilná sieť môže byť mimo prevádzky alebo zablokovaná v dôsledku intenzívneho používania, resp. preťaženia. Ďalším príkladom je situácia, keď sú pracovníci na odľahlých miestach, ako sú napríklad podzemné staveniská alebo veterné parky na mori bez pokrytia štandardnou komunikačnou sieťou. V týchto prípadoch je práca s offline dátami s originálnymi mechanizmami inteligentnej synchronizácie dát nevyhnutná. V rámci mobilnej komunikácie je najnáročnejším aspektom vytvárania mobilných riešení [21].

Z tohto dôvodu existuje koncept Offline-first approach, ktorý si popíšeme v nasledujúcej kapitole.

### 1.2.4 Offline-first approach

#### Koncept offline-first

Od offline prístupov užívatelia očakávajú, že všetky údaje budú vždy dostupné, nikdy nebudú neaktuálne, synchronizované budú na pozadí pre všetky aplikácie bez potreby čakania na dokončenie aktualizácií a pod. Nanešťastie, ako zdôrazňuje Fundinger [21], je to zvyčajne v rozpore s technologickými možnosťami, pričom limitujúcim faktorom ja obrovské množstvo údajov, ktoré je potrebné synchronizovať alebo obmedzený výpočtový výkon.

Pri implementácii offline synchronizácie údajov teda, žiaľ, neexistuje žiadne univerzálne riešenie, je preto treba ad-hoc analyzovať konkrétne požiadavky konkrétneho systému a synchronizačné modely budovať až na základe tejto analýzy, zdôrazňuje Fundinger [21].

Koncept Offline-first definuje Melamed [22] ako vývojový prístup, ktorý zaisťuje, že aplikácia bude fungovať rovnako dobre offline ako online. V takomto prístupe vidí zásadnú výhodu v tom, že ak aplikácia musí ísť na server iba vtedy, keď to potrebuje, a nie stále, bude rýchlejšia a spoľahlivejšia. To je dôležité najmä tam, kde sa obsah často nemení, ale používatelia vyžadujú rýchly prístup.

Pritom to, či má byť aplikácia offline-first, závisí podľa neho od kontextu. Tri najčastejšie dôvody pre voľbu offline-first prístupu sú nasledovné:

- Aplikácia sa pravdepodobne bude používať v oblastiach so slabým pripojením.
- Aplikácia obsahuje množstvo dát, ktoré sú dostupné predovšetkým alebo výlučne prostredníctvom funkcie vyhľadávania (vyhľadávanie veľkých objemov údajov online môže byť časovo náročné, zvlášť ak je internetové pripojenie pomalé alebo nespoľahlivé).
- Aplikácia ponúka obmedzený počet funkcií, pričom tieto nevyžadujú pripojenie online.

#### Implementácia offline-first

Offline-First je podľa [23] softvérová paradigma, ktorá predpokladá nevyhnutnosť ukladať údaje na strane klienta, aby k nim aplikácie mohli pristupovať aj v prípade straty pripojenia na Internet. To sa dá urobiť dvoma spôsobmi:

- Pomocou zložitých stratégií ukladania do vyrovnávacej pamäte,
- Pomocou lokálnej offline databázy, ktorá replikuje údaje z a do backendu na pozadí.

**Prečo offline-first?**

**Dôvod prvý: Funkčnosť aplikácie i bez pripojenia na Internet**

Aj keď riziko nemožnosti ad-hoc pripojiť sa na Internet je významnou motiváciou pre tvorbu aplikácií podľa paradigmy offline-first, ktorú musíme uviesť na prvom mieste, podľa [23] nie je táto motivácia v súčasnej dobe zďaleka tak významná ako kedysi. V minulosti nepochybne často platilo, že internetové pripojenie je nestabilné a nespoľahlivé, no veci sa menia, a to najmä v mobilných zariadeniach. Mobilné siete sa zlepšujú a výpadok pripojenia k internetu sa stáva menej bežným aj vo vzdialených lokalitách.

Preto [23] nastoľuje zásadnú otázku, že „ak nás v minulosti nezaujímali offline-first aplikácie, prečo by nás to malo zaujímať teraz?“, pričom poukazuje na to, že offline-first aplikácie sú lepšie nie preto, že podporujú ich používanie offline, ale z úplne iných dôvodov:

**Dôvod druhý: Lepšia užívateľská skúsenosť (User experience - UX)**

V klasických online-first webových aplikáciách väčšina používateľských interakcií, ako je načítanie, ukladanie alebo mazanie údajov, sa realizuje priamou požiadavkou na backendový server. To znamená, že každá z týchto interakcií vyžaduje, aby používateľ čakal na odpoveď zo vzdialeného servera, zatiaľ čo je prinútený pasívne čakať na načítavanie prvkov rozhrania.

V offline aplikáciách prebiehajú operácie priamo proti miestnemu úložisku na strane klienta, čo sa deje takmer okamžite - hneď ako používateľ klikne, na používateľskom rozhraní sa zobrazí nový stav, ako keby už bol zmenený na backende.

**Dôvod tretí: Používanie viacerých kariet funguje jednoduchšie a správnejšie**

Mnohé, dokonca aj veľké webové stránky ako Amazon, Reddit a StackOverflow, nezvládajú používanie viacerých kariet. Keď má používateľ otvorených viacero kariet webovej lokality a vykoná prihlásenie na jednej z týchto kariet, stav na ostatných kartách sa nezmení. Na offline-first aplikáciách je vždy presne jeden stav údajov (vrátane statusu prihlásenia) paralelne na všetkých kartách.

**Dôvod štvrtý: Latencia je významnejší problém ako šírka pásma (bandwidth)**


Obr. 1.7: Latencia Londýn - San Francisco

V minulosti bola šírka pásma často limitujúcim faktorom pri načítaní aplikácie. Ale zatiaľ čo sa šírka pásma v priebehu rokov zlepšila, limitujúcim faktorom sa stala latencia.

Zvýšiť šírku pásma je možné pomocou viacerých káblov alebo odoslaním viacerých komunikačných satelitov do vesmíru. Zníženie latencie 1.7 však nie je také jednoduché. Je definovaná fyzikálnymi vlastnosťami prenosového média, rýchlosťou svetla a vzdialenosťou k serveru. Všetky tieto tri faktory sa ťažko optimalizujú.

Offline-first aplikácia načíta dáta vopred, vďaka čomu už nie je treba starať sa o latenciu backendového servera.

**Dôvod piaty: Systematická aktualizácia dát**

Väčšina offline-first databáz poskytuje určitú koncepciu (typ aktualizácie, pravidelnosť aktualizácie) odberu zmien údajov z backendu. Vďaka tomu je možné mať dáta vždy na základe aktuálnej potreby.

**Dôvod šiesty: Mierka s veľkosťou údajov, nie s množstvom interakcie používateľa**

V bežných aplikáciách môže každá interakcia používateľa viesť k viacnásobným požiadavkám na backend server, čo zvyšuje jeho zaťaženie. Čím viac používateľov interaguje s aplikáciou, tým viac backendových zdrojov je treba poskytnúť.

Offline-first aplikácie nezväčšujú množstvom používateľských akcií na backend - prenesenie údajov na klienta je iba jedna izolovaná akcia. Po prenesení údajov na klienta môže používateľ s nimi vykonávať toľko interakcií, koľko je potrebné, a to bez toho, aby nimi zaťažoval server.

**Dôvod siedmy: Moderné aplikácie majú dlhšiu dobu prevádzky**

V minulosti sa webové stránky používali iba krátko. Užívateľ ich otvoril, vykonal nejakú akciu a potom ju znova zatvoril. Tým sa čas prvého načítania stal dôležitým ukazovateľom pri hodnotení rýchlosti stránky. Dnes sa webové aplikácie zmenili a s nimi aj spôsob, akým ich používame. Jednostránkové aplikácie sa otvoria raz a potom sa používajú počas celého dňa. Chatovacie aplikácie, e-mailoví klienti, atď... - tie všetky boli vytvorené tak, aby mali dlhú dobu prevádzky. Vďaka tomu je čas na interakciu používateľa dôležitejší ako počiatočný čas načítania.

**Dôvod ôsmy: Jednoduchšia implementácia komunikácie s backendom**

S offline-first aplikáciami odpadá nevyhnutnosť implementovania komplexných klient - serverových operácií, a jediné, čo je treba zabezpečiť, je implementovať správny spôsob replikácie údajov.

**Dôvod deviaty: Jednotné úložisko pre viaceré aplikácie**

V offline-first aplikáciách sú dáta na jednom mieste uloženom v lokálnej databáze. To dáva možnosť využiť tieto dáta viacerými aplikáciami, viacerými rozhraniami, dokonca viacerými zariadeniami užívateľa.

**Offline úprava údajov**

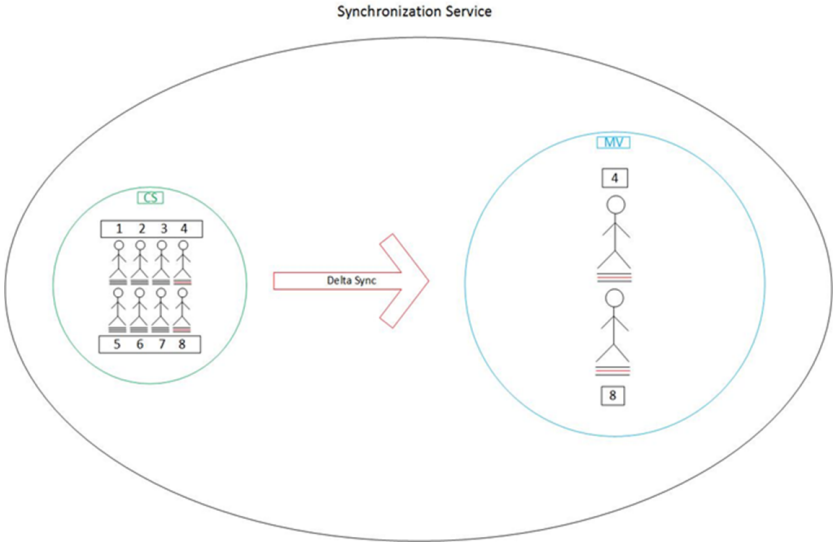
Samostatným problémom offline-first approach je offline úprava údajov. Používanie zariadení v režime offline si vyžaduje nielen predchádzajúce získavanie údajov do zariadenia, ale aj prípadná úprava (editácia) údajov v režime offline, bez možnosti ich konfrontácie s ostatnými zariadeniami.

V prípade núdzovej situácie, je hlavnou požiadavkou pre používateľov, aby mali všetky relevantné údaje k dispozícii offline až do odstránenia výpadku, no nie vždy je potrebné v núdzovom stave aj upravovať tieto údaje. Ako tvrdí Fundinger [24], „je najlepšie, ak akcie na úpravu údajov nie sú povolené počas celého offline režimu“. To je však možné iba pre niektoré obchodné scenáre.

Preto úpravy údajov offline, pokračuje Fundinger, sú vo svojej podstate zvyčajne určené len na dokumentáciu, a nie na spoluprácu s inými zariadeniami. Stačí preto uložiť tieto upravované údaje do vyrovnávacej pamäte a prehrať ich na server až pri ďalšom pripojení používateľa online. Pre používateľa je najpohodlnejšie potom to, že sa tak môže diať na pozadí.

### 1.2.5 Delta synchronizácia

Delta synchronizácia je jedna z efektívnych dostupných návrhových technológií, ktoré môžu pomôcť pri riešení výziev, ktoré so sebou prináša problém synchronizácie dát. Výhodou tohto systému je podľa [25] predovšetkým to, že sa zvyšuje rýchlosť prenosu aktualizovaných súborov z dôvodu nahrávania, resp. sťahovania iba upravených súborov (v prípade ich zmeny). Týmto prístupom sa nezvyšuje časová náročnosť operácií synchronizácie, resp. sa nezvyšuje ani náročnosť na prostriedky pri neustálej aktualizácii dát.




Obr. 1.8: Princíp Delta synchronizácie

Ako demonštruje ilustračný obrázok 1.8 (citovaný podľa [25]), keďže z ôsmich objektov (súborov) sa zmenili iba dva, synchronizované sú iba tieto dva objekty.

Delta synchronizácia pomáha pri riešení problému nadbytočnej synchronizácie údajov, a preto je rozumným riešením pri práci, ako podotýka [26], s bežnými súbormi a dátami. No pri manipulácii s veľkými komprimovanými súbormi však výrazne zlyháva, pričom hlavným zdrojom zlyhania je, že sa synchronizuje celý komprimovaný súbor, a nie iba aktualizované podsúbory. Napríklad veľké súbory (video, hudba, digitálne fotografie, atď.) sa ukladajú komprimované, preto zmena v malej časti komprimovaného súboru vyžaduje opätovnú synchronizáciu celého súboru komprimovaného archívu. To pri danom type súborov spôsobuje preťaženie siete a znižuje dostupnú šírku prenosového pásma, najmä pri synchronizácii mobilných zariadení v bezdrôtovom prostredí.

Jedným z možných riešení tohto problému pri synchronizácii veľkých komprimovaných súborov je synchronizácia iba zmenených podsúborov obsiahnutých v komprimovanom súbore. Vzhľadom na to, že komprimované súbory majú zvy-

čajne veľa podsúborov, je logickým predpokladom aktualizácie súborov obsiahnutých v komprimovaných súboroch ich primárne dekomprimovanie, následná synchronizácia aktualizovaných podsúborov a opätovná re-komprimácia. Tento proces môže trvať značné množstvo času, najmä ak existuje veľký počet podsúborov.

Preto je, uzatvára [26], potrebný vývoj metódy na synchronizáciu iba zmenených podsúborov bez toho, aby bolo treba posielať celý archív s potrebou zdĺhavého procesu extrahovania a komprimovania. Takáto metóda by znížila čas, režijné náklady na prevádzku a náklady na synchronizáciu.

Táto forma synchronizácie je nepoužiteľná pri nových užívateľoch, lebo je aj tak nevyhnutná úplná synchronizácia, keďže u nich, pochopiteľne, neexistujú staršie verzie súborov [26].

## 1.3 Analýza súčasného stavu riešenia dátovej synchronizácie

V tejto časti práce by sme radi zanalyzovali situáciu ohľadom stavu riešenia dátovej synchronizácie v súčasnosti. Budú tu spomenuté existujúce riešenia.

Nami prezentované existujúce riešenia predstavujú niektoré platformy budované na princípe offline - first so zámerom zvládnutia situácie, pri ktorom z nejakého dôvodu nie je zabezpečený dostatočný, resp. vôbec nejaký využiteľný prístup k službám backendu. V konkrétnych problémových situáciách ponúkame praktické aplikácie, ktorých úlohou je zvládnuť praktickú konfrontáciu s potrebou offline fungovania.

#### Aplikácie využívajúce offline cloud Cloudant

Cloudant je softvérový produkt IBM, ktorý sa primárne dodáva ako cloudová služba, je to však i rovnomenná nerelačná (NoSQL) distribuovaná databázová služba. Ako distribuovaná databáza je optimalizovaná pre veľké pracovné zaťaženia pre rýchlo rastúce webové a mobilné aplikácie [27].

IBM Cloudant je organizovaná ako distribuované úložisko údajov JSON s HTTP API [28]. Príťažlivosť JSON pre vývojárov jednak pramení z jeho schémy, ktorá sa môže rýchlo vyvíjať bez zásahu správcu, jednak pre svoju príbuznosť s JavaScriptom [27], v rámci ktorého JSON ponúka elegantný model pre trvalé dátové objekty Java alebo JavaScript.

Cloudant poskytuje vývojárom možnosť:

- Tvorby webových aplikácia formou API bez potreby servera
- Tvorbu mobilných aplikácií - taktiež bez potreby servera

### Cloudant Mobile Sync

IBM Cloudant Sync je sada knižníc pre iOS a Android, ktorá umožňuje lokálne ukladanie údajov v mobilnom zariadení a ich synchronizáciu s IBM Cloudant v čase, keď to mobilné pripojenie umožňuje.

Pomocou špeciálnej služby Cloudant Mobile Sync môžu mobilné aplikácie zhromažďovať a čítať údaje aj vtedy, keď je sieť nedostupná. Po nadviazaní spojenia sa údaje zosynchronizujú so základnou službou Cloudant.

Replikácia je dokončená, keď sa najnovšia verzia každého dokumentu v zdroji prenesie do cieľovej databázy. Prevody zahŕňajú nové dokumenty, aktualizácie existujúcich dokumentov a mazania. Po replikácii zostane iba najnovšia verzia dokumentu; staršie verzie sú vynechané [28].

### Aplikácie využívajúce offline databázu PouchDB

PouchDB je tzv. databáza v prehliadači, ktorá umožňuje aplikáciám ukladať údaje lokálne, takže používatelia môžu využívať všetky funkcie aplikácie, aj v čase, keď sú offline.

Okrem toho má databáza PouchDB tieto vlastnosti [29]:

- PouchDB je bezplatný open-source projekt napísaný v JavaScripte a riadený komunitou nadšencov.
- PouchDB možno použiť ako priame rozhranie k serverom kompatibilným s databázou CouchDB resp. PouchDB môže bežať aj ako vlastný webový server kompatibilný s CouchDB.
- Rozhranie PouchDB API funguje rovnako v každom vývojovom prostredí, preto vývojár nemusí stráviť veľa času implementovaním rozdielov v jednotlivých prehliadačoch a viac času môže venovať písaním samotného kódu.
- PouchDB pritom podporuje všetky moderné prehliadače, resp. mobilné operačné systémy.
- Dokumenty, ktoré sa ukladajú do PouchDB, musia byť serializovateľné ako JSON. Úprava prototypu objektu alebo ukladanie tried nie je podporované [30].

**Iné databázové možnosti**

Replikačný protokol IBM Cloudant je okrem vlastnej štruktúry JSON kompatibilný s inými databázami a knižnicami určenými pre rôzne aplikácie v reálnom svete, konkrétne Apache CouchDB a PouchDB.

Apache CouchDB je databáza typu open source, ktorá dokáže komunikovať s IBM Cloudant a vyžaduje pritom minimálne nastavenie. Súčasťou implementácie sú nasledujúce aplikácie:

- Zálohovanie – formou replikácie údajov z IBM Cloudant do vlastných databáz CouchDB.
- Lokálne zhromažďovanie údajov – údaje sa najskôr zapíšu do lokálnej databázy Apache CouchDB a potom replikujú do IBM Cloudant na dlhodobé ukladanie, agregáciu, prípadne dátovú analýzu.

PouchDB je taktiež open source databáza určená pre internetové prehliadače, ktorá umožňuje replikáciu údajov v oboch smeroch medzi prehliadačom a IBM Cloudant. Ukladanie údajov do webového prehliadača na strane klienta umožňuje fungovanie webových aplikácií aj bez internetového pripojenia. PouchDB dokáže synchronizovať akékoľvek zmenené údaje do alebo z IBM Cloudant, a to vtedy, keď je k dispozícii internetové pripojenie. Nastavenie replikácie zo strany klienta je pritom jednoduché - vyžaduje iba niekoľko riadkov JavaScriptu (podrobnejšie viď 1.3).

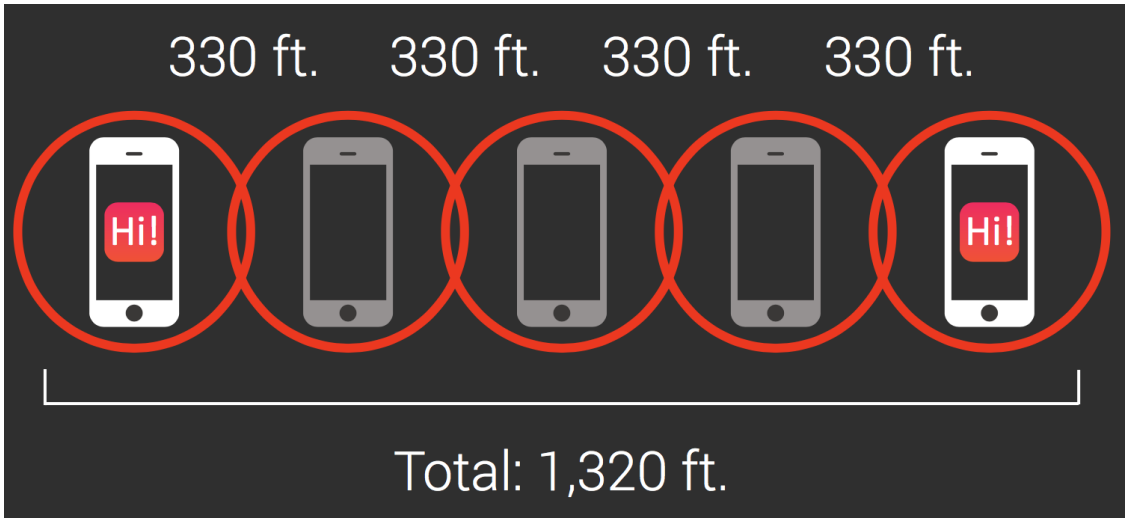
Príbehy „o úspechu spoločností, kedy sa firmy rozhodli použiť pri vlastných aplikáciách primárne offline prístup“, sú publikované na osobitnej PR (public relation) stránke spoločnosti IBM, ktorá ponúka svoj produkt Cloudant [31].

### 1.3.1 Prípadová situácia číslo 1 - Bridgefy

Problémová situácia: Komunikácia topograficky blízkych (fyzicky nie príliš vzdialených od seba navzájom) spolupracovníkov, spolužiakov alebo rodinných príslušníkov pri slabom alebo žiadnom pripojení na internet, pričom pre komunikáciu sú k dispozícii zariadenia s Bluetooth. V blízkosti nie je dostupná ani žiadna využiteľná sieť Wi-Fi.

Riešenie: Aplikácia Bridgefy

Bridgefy je aplikácia, ktorá umožňuje odosielať offline textové správy, keď neexistuje prístup na internet, a to pomocou Bluetooth. Ako tvrdí výrobca, aplikácia je „ideálna na spojenie s ostatnými počas prírodných katastrof, na veľkých podujatiach a v škole“[32].




Obr. 1.9: Bridgefy: one-on-one long distance message

Aplikácia Bridgefy na odosielanie offline správ využíva svoj vlastný softvér, Bridgefy SDK, na prepojenie dvoch rôznych smartfónov do vzájomnej vzdialenosti 100m (330ft.). Používatelia Bridgefy Android aj iPhone môžu spoločne vytvoriť sieť typu mesh. Týmto spôsobom môže správa „preskakovať" z jedného mobilného telefónu na iný mobilný telefón, čím komunikácia môže pokryť veľké vzdialenosti (viď. obrázok 1.9). Adresáti správy nemusia byť pritom ani v zozname odosielateľových kontaktov.

Podľa návodu na stránke, aplikácia Bridgefy funguje na veľkú vzdialenosť tak, že odosielateľ najprv pošle vašu správu všetkým okolo seba, tieto zariadenia v bezprostrednom okolí potom prepošlú túto správu všetkým ďalším zariadeniam v svojom okolí, atď., čím sa komunikáciou pokryje obrovské množstvo používateľov [33].

Od októbra 2020 sú tiež všetky priame správy medzi dvoma používateľmi šifrované.

Okrem hotovej komunikačnej aplikácie ponúka výrobca i platformu Bridgefy Software Development Kit (SDK). Podľa ich vlastnej deklarácie [34], je to najmodernejšia sieťová súprava SDK, ktorá vývojárom ponúka možnosť vytvárať vlastné originálne mobilné aplikácie pre komunikáciu v prípade, keď nemajú prístup na internet, ale majú možnosť využiť Bluetooth.

### 1.3.2 Prípadová situácia číslo 2 - Briar

Problémová situácia: Podobne ako v predchádzajúcom príklade, vyžaduje sa komunikácia topograficky blízkych (fyzicky nie príliš vzdialených od seba navzájom) spolupracovníkov, spolužiakov alebo rodinných príslušníkov pri slabom

alebo žiadnom pripojení na internet, pričom pre komunikáciu sú k dispozícii jednak zariadenia s Bluetooth, jednak je v blízkosti dostupná reálne využiteľná sieť Wi-Fi.

Riešenie: Aplikácia Briar.

Briar je aplikácia na odosielanie správ určená (podľa slov vývojára) „pre aktivistov, novinárov a kohokoľvek iného, kto potrebuje bezpečný, jednoduchý a robustný spôsob komunikácie“. Na rozdiel od tradičných aplikácií na odosielanie správ sa Briar nespolieha na centrálny server – správy sa synchronizujú priamo medzi zariadeniami používateľov. Ak je internet nefunkčný, Briar sa môže synchronizovať cez Bluetooth alebo Wi-Fi.

Briar používa priame šifrované spojenia medzi používateľmi, aby sa zabránilo vonkajšiemu dohľadu a prípadnej cenzúre [35].

### 1.3.3 Prípadová situácia číslo 3 - AERO

Problémová situácia: Obchodní zástupcovia alebo predajcovia pri práci v teréne potrebujú zostať v kontakte a budovať lepšie vzťahy so zákazníkmi i v prípade výpadku spojenia s Internetom.

Riešenie: Aplikácia AERO od spoločnosti eFrontech.

eFrontech je francúzka spoločnosť, ktorá na princípe offline - first ponúka svoj produkt AERO - riešenie offline mobility s podporou cloudu IBM Cloudant [36]. Produkt bol predstavený počas veľtrhu Big Data v Paríži v roku 2016.

Aero ponúka úplný offline prístup k údajom CRM spolu s ultrarýchlou automatickou synchronizáciou údajov. Aero používa Cloudant na synchronizáciu údajov s back-end systémami v reálnom čase, čím umožňuje tímom predajcov okamžitý prístup k údajom CRM, a to kedykoľvek a kdekoľvek [31].

### 1.3.4 Prípadová situácia číslo 4 - Quetzal

Problémová situácia: Prístup k informáciám o tovaroch a POS terminály pre lokálne predajne oblečenia a obuvi. Predavačky a pokladníčky v predajniach potrebujú systém, ktorý im umožní získať informácie o produkte, ktorý si zákazník žiada a súčasne umožniť zákazníkom platiť za tovar priamo u predavačky. Obsluhované zariadenie (napríklad tablet patriaci predavačke) nemusí mať prístup na Internet.

Riešenie: Systém POS terminálov Quetzal.

Quetzal je riešenie na miestne predajne. Nezáleží na tom, či má klient (zákazník systému Quetzal) skvelé alebo slabé, či dokonca žiadne pripojenie v každej

časti svojho obchodu - Cloudant ponúka bezproblémové a spoľahlivé offline prostredie.

Všetky transakcie sú uložené lokálne vo formáte JSON na iPad a ihneď, ako sa iPad znova pripojí k internetu, sa údaje (pomocou Cloudant Sync) nahrajú do služby Cloudant.

### 1.3.5 Prípadová situácia číslo 5 - YukonBaby

Problémová situácia: Prístup k informáciám o starostlivosti o deti pre matky žijúce v odľahlých oblastiach Yukonu s nespoľahlivým prístupom k Internetu.

Riešenie: Aplikácia YukonBaby.

YukonBaby vyvinula aplikáciu, ktorá pomáha rodičom na odľahlom území Yukonu v severnej Kanady (kde je pripojenie veľmi nespoľahlivé) získať prístup k najnovším zdrojom starostlivosti o deti, a to formou online aj offline. Pokročilé možnosti synchronizácie Cloudantu umožňujú YukonBaby zaujať offline prístup, ktorý používateľom umožňuje pracovať primárne offline a synchronizovať dáta iba vtedy, keď je dostupné pripojenie. To tiež udržuje nízke náklady pre používateľov tým, že minimalizuje poplatky za dáta.

### 1.3.6 Prípadová situácia číslo 6 - HospitalRun

Problémová situácia: Lekári v oblastiach tretieho sveta, kde kvalita pokrytia Internetom nedosahuje kvalitu, ktorá je štandardom v rozvinutom svete, vyžadujú aplikáciu pre správu zdravotných záznamov svojich pacientov a spoľahlivý nemocničný informačný systém.

Riešenie: Webová aplikácia HospitalRun.


Obr. 1.10: Užívateľské rozhranie HospitalRun

HospitalRun prvá elektronická offline databáza zdravotných záznamov (Electronic Health Record (EHR)) a webová aplikácia nemocničného informačného systému (Hospital Information System (HIS)). Rozhranie je jednoduché, ale prehľadné (viď. obrázok 1.10). HospitalRun je navrhnutý tak, aby umožňoval prenos zdravotných záznamov od terénnych lekárov pracujúcich v ťažko dostupnom teréne s problematickým pripojením na Internet, špeciálne pre regióny tretieho sveta. Zdravotné záznamy prenáša aj na kliniky vzdialené stovky kilometrov od miesta pôsobenia lekára. Funguje aj v prípade, keď nie je dostupný internet, a synchronizuje sa, keď sa pripojenie obnoví [37].

Softvér je možné nasadiť v rôznych zdravotníckych prostrediach. Vďaka svojim technickým vlastnostiam, ktoré umožňujú použitie aj bez pripojenia, je vhodný aj pre kliniky nachádzajúce sa v najvidieckejších oblastiach planéty.

Vďaka vývojárskym dobrovoľníkom a dobrovoľným prispievateľom je HospitalRun bezplatné softvérového riešenie s otvoreným zdrojom pre využitie v lekárskej praxi [38].

## 1.4 Aplikačné problémové okruhy

Pri analýze existujúcich riešení, pri ich ideovej syntéze, posudzovaní vhodnosti a pri následnom návrhu vlastnej aplikácie budeme dbať na to, aby boli vhodne riešené nasledujúce problémové okruhy:

- Výpadok pripojenia na WAN server. Systém musí byť dostatočne robustný, aby zvládol primárny kritický bod vlastného fungovania - výpadok prístupu k službám serveru (backendu).
- Zdieľanie informácií medzi pripojenými zariadeniami. Systém po výpadku prístupu na server musí byť schopný prejsť na offline režim, v ktorom bude schopný umožniť vzájomnú komunikáciu podriadených uzlov (pracovných staníc, resp. frontendov), a to bez vlastnej dezorganizácie, zvýšenej chybovosti či dokonca fatálneho zrútenia sa.
- Opätovné nastolenie pripojenia a synchronizácia so serverom. Systém identifikuje opätovné pripojenie sa k službám servera

# 2 Syntetická časť

Na demonštráciu riešenia toho, ako môže distribuovaný systém funkčne zvládnuť výpadok pripojenia k štandardne využívaným službám backendu poskytovaným na platforme Internetu, sme si zvolili situáciu popísanú v úvode našej práce.

## 2.1 Proces online-first synchronizácie

MVC je ukážkou online-first synchronizácie. Tento návrhový vzor v tomto prípade je možné poňať ako rozdelenie celého projektu na časti, kde *Model* predstavuje databázu na strane servera, *View* (pohľad) predstavuje samotnú aplikáciu na strane klienta, poprípade akúkoľvek prezentačnú vrstvu používanú koncovými používateľmi na príjem údajov. Príkladom môžu byť rôzne knižnice a softvérové rámce pre vývoj frontend aplikácií pre webové prehliadače, ako napríklad knižnica React alebo rámce Angular a Vue.js. *Controller* (kontrolér) je v tomto prípade komunikačný kanál, pomocou ktorého sú samotné údaje prenášané z úložiska dát (napr. databáza, čo je v tomto prípade model) do aplikácie, resp. prezentačnej vrstvy pre používateľa (v tomto prípade view, resp. pohľad). Ďalšie informácie o dátovej synchronizácií je možné získať v rámci tejto práce v kapitole 1.2.

V nasledujúcej časti bude opísaný pseudokód, ktorý bude slúžiť pre validáciu vstupeniek na rôznych udalostiach. Cieľom pseudokódu je opísať správanie online-first aplikácie bez toho, aby bolo nutné písať zdrojový kód v špecifickom programovacom jazyku. V skratke je princíp fungovania aplikácie taký, že používateľ zadá (oskenuje) čiarový kód do aplikácie, následne sa tento čiarový kód odošle a overí na serveri pomocou HTTP požiadavky. Po overení správnosti a pravosti lístka je umožnené jeho použitie. Funkcia *enterTicket* slúži pre používateľa na zadanie, resp. oskenovanie čiarového kódu z lístka, funkcia *fetchOnlineTicket* slúži na validáciu a získanie lístka, funkcia *validateTicket* slúži na lokálne overenie a v neposlednom rade, funkcia *useTicket* slúži na samotné použitie vstupenky, či lístku.

```
Function enterTicket()
     output "Ticket barcode"
     input <- userinput
     return input
end function

Function fetchOnlineTicket(barcode)
    ticket <- HTTP.get(barcode)
    return ticket
end function

Function validateTicket(ticket)
    If ticket.status = 0 Then
        return False
    Else If ticket.amount = 0 Then
        return False
    Endif
    return True
end function

Function useTicket(ticket, amount)
    result <- HTTP.post(ticket, amount)
    return result
end function

barcode <- enterTicket()
ticket <- fetchOnlineTicket(barcode)
is_valid_ticket <- validateTicket(ticket)
If is_valid_ticket Then
    output "Ticket is valid"
Else
    output "Ticket is invalid"
Endif

is_used_ticket <- useTicket(ticket, 5)
If is_used_ticket Then
    output "Ticket was used successfully"
Else
    output "Ticket is invalid"
Endif
```

## 2.2 Proces offline-first synchronizácie

V rámci tejto práce sme sa venovali taktiež prístupu offline-first, presnejšie v kapitole 1.2.4. Tento prístup má za úlohu poskytnúť používateľovi možnosť pracovať s aplikáciou aj napriek tomu, že nedisponuje internetovým pripojením a teda nemôže kontaktovať server a požiadať od neho nové aktualizácie. Nasledujúci pseudokód obsahuje niekoľko funkcií a subrutín, ktoré sú potrebné pre prácu v offline režime. Funkcia *enterTicket* slúži na podobne ako v prípade online-first prístupu na načítanie čiarového kódu.

```
Function enterTicket()
    output "Ticket barcode"
    input <- userinput
    return input
end function
```

Funkcia *fetchOnlineTicket* slúži na overenie a získanie lístka či vstupenky, pričom sa využíva HTTP GET požiadavka.

```
Function fetchOnlineTicket(barcode)
   ticket <- HTTP.get(barcode)
   return ticket
end function
```

Nasleduje funkcia *fetchLocalTicket*, ktorá je presným opakom predošlej funkcie, pretože táto má za tú istú úlohu, avšak v tomto prípade sa lístok má získať z lokálnej databázy a nie vzdialeného servera.

```
Function fetchLocalTicket(barcode)
   ticket <- DB.get(barcode)
   return ticket
end function
```

*ValidateTicket* je, podobne ako v prípade online-first, funkcia, ktorá sa stará o lokálne overenie správnosti a pravosti lístkov.

```
Function validateTicket(ticket)
   If ticket.status = 0 Then
       return False
   Endif
   If ticket.amount = 0 Then
       return False
   Endif
   return True
```

```
end function
```

Ďalšou je funkcia *useTicket*, ktorá slúži na použitie daného lístka, pričom v tomto prípade má za úlohu nastavenie času a dátumu poslednej aktualizácie a tiež uloženie do databázy.

```
Function useTicket(ticket, amount)
   ticket.lastUpdate <- Date.now()
   result <- DB.update(ticket, amount)
   return result
end function
```

Ďalšie funkcie sú veľmi dôležité pre prístup offline-first, keďže je potrebné zistiť, kedy je aplikácia v režime offline a tiež je potrebné periodicky synchronizovať a aktualizovať údaje aplikácie.

```
Function isNetworkAvailable()
   result <- System.isNetworkAvailable()
   return result
end function

Async Sub periodicSynchronization()
   While app.isRunning()
       data <- HTTP.get()
       DB.storeData(data)
       sleep 30000
   Endwhile
end Sub

Async Sub periodicUpload()
   lastUpload <- Date.now()
   While app.isRunning()
       data <- DB.getChangedFrom(lastUpload)
       If data.size > 0 Then
           HTTP.post(data)
           lastUpload <- Date.now()
       Endif
       sleep 30000
   Endwhile
end sub
```

Použitie týchto funkcií a subrutín je možné vidieť na pseudokóde nižšie.

```
periodicSynchronization()
```

```
barcode <- enterTicket()
is_network_available = isNetworkAvailable()
If is_network_available Then
   ticket <- fetchOnlineTicket(barcode)
Else
   ticket <- fetchLocalTicket(barcode)
Endif

is_valid_ticket <- validateTicket(ticket)
If is_valid_ticket Then
   output "Ticket is valid"
Else
   output "Ticket is invalid"
Endif

is_used_ticket <- useTicket(ticket, 5)
If is_used_ticket Then
   output "Ticket was used successfully"
Else
   output "Ticket is invalid"
Endif
```

#### Porovnanie online-first a offline-first prístupov

Z pseudokódov oboch prístupov vyššie spomenutých je možné odvodiť, že implementácia offline-first prístupu je ťažšia a komplikovanejšia na implementáciu ako prístup online-first, keďže je nutné periodicky synchronizovať dáta oboma smermi, teda smerom ku serveru a tiež smerom ku lokálnej databáze, pričom v online-first prístupe dokonca nie je požadovaná ani lokálna databáza. V offline-first prístupe je tiež nutné overovanie stavu sieťového pripojenia, keďže je nutné zistiť, kedy sa aplikácia nachádza v stave online a kedy v stave offline.

#### Rozšírená offline-first synchronizácia

Na druhej strane, je nutné podotknúť, že offline-first prístup má aj svoje výhody. Medzi takého výhody môžeme zaradiť responzivitu aplikácie a nižšiu náročnosť na sieťovú premávku vďaka delta synchronizáciám (kapitola 1.2.5).

## 2.3 Prípadová štúdia: validácia zakúpených vstupeniek na outdoorové kultúrne podujatie s viacerými vstupmi

Outdoorové podujatia (koncerty, hudobné alebo divadelné festivaly a pod.), ale napríklad aj workshopy veľkých firiem, sa vyznačujú účasťou veľkého množstva navzájom neznámych ľudí, prípadne známych iba "z videnia". Ich účasť na podujatí musí byť preto aj formálne validovaná - z účasti na podujatí musia byť vylúčení ľudia, ktorých účasť nie je preukázateľná formálnym dokumentom - vstupenkou alebo pozvánkou. Overenie musia vykonávať osobitne na tento účel vyčlenení pracovníci, či už interní zamestnanci, resp. dobrovoľníci na festivaloch, alebo outsourceovaní pracovníci najatých firiem.

V prípade, ak je na podujatie iba jedno oficiálne vstupné miesto (vstupná brána, vchod, checkpoint), je situácia jednoduchá, keďže i pri prípadnom výpadku služieb Internetu, je možné využiť lokálnu databázu vstupeniek a pozvánok, ktorá bude dostupná na danom vstupnom mieste a zmeny v tejto databáze (flagovanie vstupenky ako autorizovaná) bude dostatočnou garanciou toho, že vstupenka nebude uplatnená na inom vstupnom mieste (keďže iné vstupné miesto neexistuje) a nebude uplatnená ani inou osobou (keď prvá autorizovaná osoba po validácii už vstúpila do priestoru konania podujatia).

Situácia je principiálne odlišná vtedy, ak je vstupných miest viac, resp. ak je v rámci jedného miesta viacero paralelných priechodov (napríklad formou turniketov). V takomto prípade strata kontaktu s centrálnou databázou na strane backendu predstavuje riziko viacnásobného neautorizovaného, a teda nepovoleného vstupu na miesto konania podujatia. Napríklad, ak majiteľ vstupenky, ktorá bola na jednom vstupnom mieste v lokálnej databáze označená za validovanú, túto vstupenku jednoducho odovzdá na nestráženom mieste doslova cez plot niekomu inému, kto sa s ňou môže preukázať na inom vstupnom mieste, resp. inom turnikete, kde v lokálnej databáze ešte nemá vstupenka označenie ako validovaná.

Okrem skutočnosti, že účasťou neoprávnených osôb na podujatí prichádza organizátor o časť príjmu, je ešte závažnejším problémom to, že organizátor stráca prehľad o reálnej fyzickej účasti osôb na podujatí. V prípade krízových situácií, ktoré na outdoorovom podujatí môžu vzniknúť (spomeňme pád veľkokapacitného stanu počas búrky na festivale), nebude môcť organizátor poskytnúť záchranným zložkám hodnoverný počet účastníkov aktuálne sa nachádzajúcich v

areáli konania podujatia.

**Ideový zámer návrhu riešenia**

Na minimalizáciu vyššie uvedeného rizika navrhujeme nižšie popísané riešenie. Naším zámerom je ponúknuť riešenia aplikácie schopnej synchronizovať dáta v dvoch režimoch:

- Optimum: služby backendu, v ktorom budú synchronizované databázy klientov s centrálnou databázou, považujeme za štandardné služby, ktoré sú systémom využívané v režime bežnej prevádzky (optimálny režim prevádzky).
- Exception: služby peer-to-peer komunikácie, v ktorom budú vzájomne synchronizované lokálne databázy klientov, považujeme za núdzové riešenie v čase výpadku Internetu, ktoré však musí svojou funkcionálnou povahou v dostatočnej miere nahradiť služby režimu optimum (núdzový režim prevádzky).

**Technicko - organizačné propozície**

Pri návrhu aplikácie vychádzame z nasledujúcich technicko - organizačných propozícií:

Každá vstupenka vygenerovaná cez oficiálneho predajcu vstupeniek (ticketportal.sk, eventim.sk a pod.) obsahuje jedinečný kód. Konfrontácia kódu na vstupenke s databázou kódov pridelených jednotlivým zakúpeným vstupenkám je spoľahlivým spôsobom, ako znemožniť uplatnenie falošnej vstupenky alebo opätovné uplatnenie dávnejšie uplatnenej vstupenky. V lokálnej i centrálnej databáze bude preto vstupenka identifikovaná svojím kódom, ktorý je rozličný podľa predajcu lístkov.

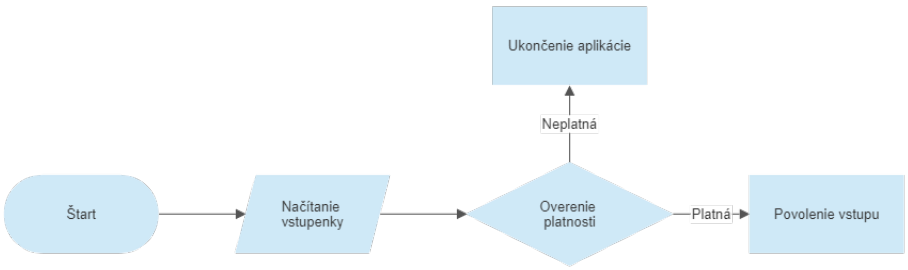
Iné formy oprávnenia na vstup (pozvánky) budú musieť disponovať vlastným identifikačným kódom, pod ktorým budú pozvánky evidované v databázach.

Užívateľ A je užívateľ, ktorý príde ku turniketu a prvýkrát použije vstupenku pre vstup na akciu. Užívateľ B je známy užívateľa A, ktorý sa chce nelegálne dostať na akciu cez iný turniket a použije použitú vstupenku od užívateľa A.

### 2.3.1 Analýza činností a katalóg požiadaviek

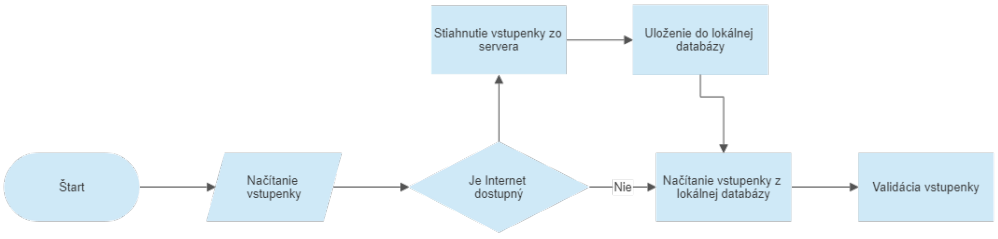
Základné modelovanie architektúry systému sme realizovali pomocou vývojového diagramu (flow chart diagram), ktorý uvádzame na obrázku 2.1. Popisuje správanie systému z hľadiska užívateľa pri prechode turniketom.

Prvým krokom pri overovaní užívateľa je načítanie jeho vstupenky aplikáciou, pričom aplikácia rozhodne o validite vstupenky, kde v prípade valídnej vstupenky bude môcť užívateľ prejsť cez turniket. V opačnom prípade, teda ak nemá validnú vstupenku, mu bude vstup zamietnutý.




Obr. 2.1: Základné overenie lístka

Ako je možné vidieť na obr. 2.2, tak pri kombinovanej synchronizácii aplikácia pracuje vždy s dátami z lokálnej databázy. To znamená, že ak je Internet dostupný, iný proces musí zabezpečiť, aby sa stiahli čo najnovšie dáta. Stiahnutie dát zo servera funguje na základe vyskladania požiadavky na server, kde sa priamo vyžiada daná vstupenka a nebudú sťahované všetky dáta. Komplikovať sa to začne, ak je dostupná iba lokálna sieť a nie sieť s prístupom na Internet. V takomto prípade je riešenie opísané v diagrame 2.3.

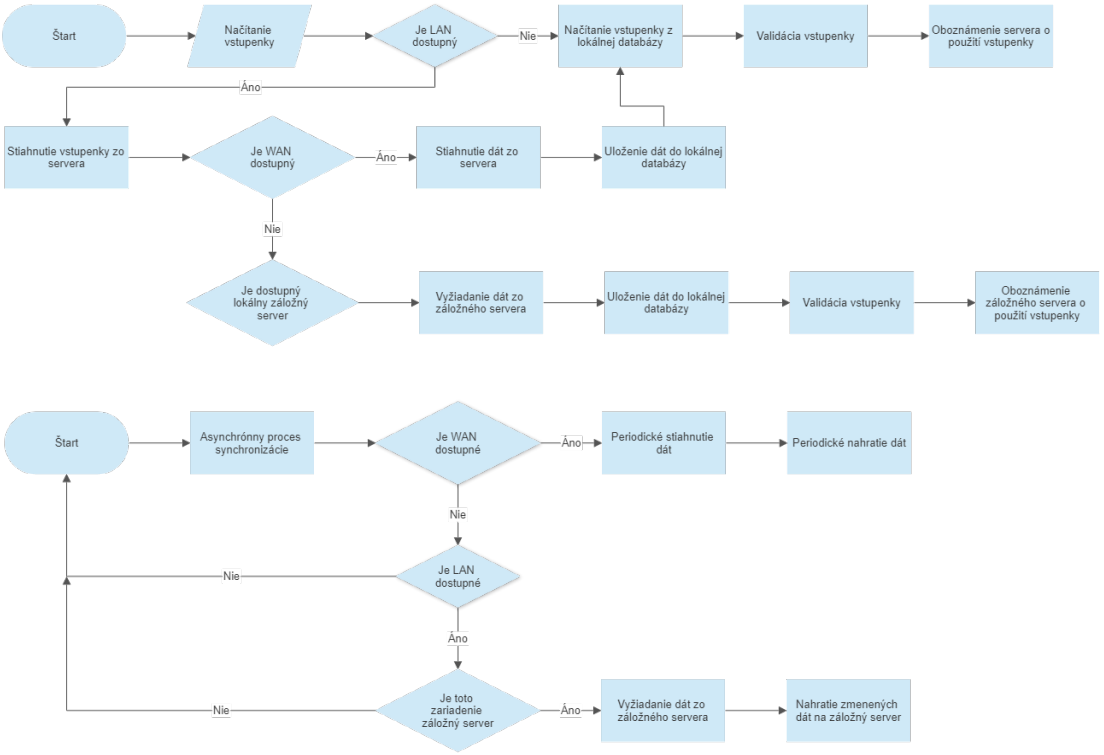



Obr. 2.2: Základné overenie lístka cez internet

Zabezpečenie aktuality dát je v prípade, keď môže nastať nedostupnosť WAN siete. Je to riešené procesmi na pozadí, ktoré zabezpečujú pravidelnú aktualizáciu dát pred výpadkom Internetu. Pravidelnou synchronizáciou sa sťahujú vždy delta zmeny od poslednej synchronizácie. Na to je použitá dodatočná tabuľka, do ktorej sa ukladá časová pečiatka s informáciou o poslednej synchronizácii a tabuľke, ktorá bola stiahnutá. Ak pri synchronizácii nedôjde k stiahnutiu nových záznamov (počet záznamov je rovný nule), záznam v doplnkovej tabuľke sa neaktualizuje. Týmto zabezpečíme to, že ak by niektorý systém vykonal transakciu v offline režime a bola by spracovaná serverom neskôr, tak všetky zariadenia budú mať možnosť zosynchronizovať túto zmenu.

Aby bola pravidelná synchronizácia efektívna, musí existovať pomocný proces, ktorý zabezpečí pravidelnú kontrolu spojenia so serverom (liveness). Tento

proces bude v pravidelnom intervale robiť tzv. *ping* na server. Dôvodom pre tento proces je, že systém, na ktorom beží aplikácia, môže mylne hlásiť, že je dostupné Internetové spojenie, aj keď tomu tak nemusí byť a je dostupné iba LAN spojenie.




Obr. 2.3: Pokročilá správa dát

V prípade, že počas pravidelnej synchronizácie dôjde k nedostupnosti WAN siete, bude o tom proces vedieť na základe pomocného procesu. V tej chvíli prichádza do úvahy záložná synchronizácia v LAN sieti. Tá prebieha vždy medzi dvoma zariadeniami, z ktorých jedno zariadenie je označené ako **záložný server**. Označenie zariadenia ako záložného serveru môže byť dosiahnuté rôznymi algoritmami, napr. Paxos (1.1.5), Raft alebo i. V tejto práci sme sa rozhodli nezaoberať konsenzusom, ale rozhodnutie o výbere zariadenia ako záložného servera prenecháme na používateľovi. Keď je aplikácia označená ako záložný server, tak pri pravidelnej synchronizácii nesťahuje dáta z ostatných zariadení, ale naopak, ostatné zariadenia odosielajú údaje na tento záložný server, teda ak aplikácia nie je označená ako záložný server, musí mať v pamäti definovanú cestu (IP adresu a port), na ktorej nájde záložný server. Aplikácia vyšle požiadavku podobného tvaru, akú by posielala na server, a to informácie o požadovanej tabuľke spolu s informáciou o poslednej časovej pečiatke, od ktorej očakáva najnovšie dátové zmeny. Záložný server po prijatí takejto požiadavky vytvorí nový proces, v ktorom dohľadá požadované dáta, spracuje ich (*serializácia*) a odošle cez socketové spojenie.

Aby bola zabezpečená konzistencia dát medzi zariadeniami, tak všetky zariadenia musia informovať záložný server o všetkých zmenách, ktoré nastali na dátovej vrstve. Keď po naskenovaní vstupenky a jej overení bude vstupenka uplatnená, tak zariadenie v prípade nedostupnosti WAN siete odošle túto zmenu na záložný server.

### 2.3.2 Algoritmus kombinovanej synchronizácie

Riešenie synchronizácie pri problémoch so sieťou opíšeme na príklade so vstupenkami pomocou vývojového diagramu 2.3.

- Základom aplikácie by malo byť to, že pozná cestu na server. Tento algoritmus principiálne bude fungovať, aj keď aplikácia nikdy nebola pripojená na reálny server. Funkčnosť bude zaručená stiahnutím všetkých dát (*replikácii*) zo záložného servera (*zariadenia*).

- Keď aplikácia pozná cestu na server, prvým krokom bude to, že si urobí lokálnu repliku dát, s ktorými pracuje. Samozrejme, ak server obsahuje veľké množstvo dát, aplikácia nemusí stiahnuť všetky z nich, ale môžu byť filtrované na danú lokalitu, v ktorej sa bude používať (*multitenancy*). Napríklad, keď máme podujatie A, tak nemusíme sťahovať do zariadenia dáta o podujatí B, je zrejmé, že keď sa bude zariadenie používať na podujatí B, tak bude priestor na synchronizáciu so serverom. Prvá replika dát (*full synchronization*) zabezpečí to, že od tej chvíle už bude prebiehať len delta synchronizácia od tejto časovej pečiatky.

- Spustí sa proces na popredí, ktorý zabezpečí neustálu primárnu komunikáciu so serverom. Proces na popredí by mal zaručiť to, že aj keď používateľské rozhranie aplikácie bude ukončené, tak dátová synchronizácia bude aj naďalej prebiehať. Dáta takýmto prístupom budú vždy aktuálne.

- Spustí sa proces, ktorý bude kontrolovať dostupnosť WAN siete pomocou nástroja *ping* na server.

- Keď je aplikácia plne načítaná, mala by mať informáciu o tom, či je označená ako záložný server alebo nie. Môže to byť riešené pomocou konfiguračného súboru, používateľského rozhrania alebo algoritmu (1.1.5).

- Keď je aplikácia označená ako záložný server, sprístupní socketové spojenie na vopred definovanom a známom porte. Zabezpečíme to tým, že iné

aplikácie, ktoré by mohli mať problémy so sieťou, budú môcť pristupovať k záložnému serveru (tomuto zariadeniu).

- Keď proces na popredí zaregistruje nedostupnosť WAN siete a je označený ako záložný server, nevykoná žiadnu akciu.
- Keď proces na popredí zaregistruje nedostupnosť WAN siete a nie je označený ako záložný server, vyšle žiadosť o dáta, ako bolo opísané v diagrame 2.3.
- Ak nastane zmena dát na záložnom serveri, zvyšné aplikácie sa o tejto zmene dozvedia pomocou pravidelnej synchronizácie.
- Ak nastane zmena dát na aplikácii, ktorá nie je označená ako záložný server, táto aplikácia nahrá svoje zmenené dáta na záložný server, aby boli prístupné aj pre zvyšné zariadenia.
- Možné krajné prípady budú vyriešené pomocou servera po obnovení spojenia (*WAN siete*). Je potrebné si uvedomiť, že každá zmena na referenčných dátach je reprezentovaná transakciou. Tieto transakcie sa nahrávajú výlučne na server. Môžeme si to predstaviť ako bloček, kde každý má vlastný identifikátor. A teda, keby aj nastal okrajový prípad, kedy by bolo z jednej karty stiahnutých viac peňazí ako by bolo na karte dostupných, tak na serveri bude táto karta reprezentovaná zápornou sumou a rozdiel v cene bude od majiteľa spätne domáhaný. Tiež akonáhle server urobí zmenu na dátach, zmení sa časová pečiatka a tieto zmeny budú rozdistribuované medzi zariadenia.

### 2.3.3 Pseudokód kombinovanej synchronizácie

Táto sekcia práce sa zaoberá ukážkovým pseudokódom realizácie kombinovanej synchronizácie, ktorá umožňuje zvýšenú dostupnosť aktuálnych dát medzi zariadeniami v lokálnej sieti. Algoritmus predpokladá, že sú dostupné konfiguračné parametre a nie sú v sieti prekážky typu NAT Firewall.

Pomocná funkcia, ktorá načíta z konfigurácie statický parameter o tom, či dané zariadenie bude slúžiť ako záložný server.

```
Function isLocalServer()
    is_local_server <- configuration.get('is_local_server')
    return is_local_server
end function
```

Aplikácia, ktorá nie je záložným serverom, s ním udržiava aktívne socketové spojenie. Naopak, záložný server má zoznam socketových spojení s klientami.

```
Function hasConnectionToLocalServer()
    has_connection_to_local_server <- LocalState.localServer()
    return has_connection_to_local_server
end function
```

Reprezentácia skenovania vstupenky na reálnom zariadení môže byť reprezentovaná priložením NFC karty alebo prečítaním čiarového kódu skenerom.

```
Function enterTicket()
     output "Ticket barcode"
     input <- userinput
     return input
end function
```

Funkcia, ktorá stiahne špecifické dáta zo servera. Myslí sa tým jeden aktuálny záznam, s ktorým aplikácia pracuje.

```
Function fetchOnlineTicket(barcode)
    ticket <- HTTP.get(barcode)
    return ticket
end function
```

Načítanie záznamu, s ktorým pracujeme, z lokálnej databázy.

```
Function fetchLocalTicket(barcode)
    ticket <- DB.get(barcode)
    return ticket
end function
```

Overenie, či dáta, s ktorými aplikácia pracuje, sú správne.

```
Function validateTicket(ticket)
    If ticket.status = 0 Then
        return False
    Endif
    If ticket.amount = 0 Then
        return False
    Endif
    return True
end function
```

Zmena údajov v dátach, s ktorými aplikácia pracuje, a ich synchronizácia so záložným serverom v prípade nedostupnosti primárneho servera.

```
Function useTicket(ticket, amount)
    ticket.lastUpdate <- Date.now()
    result <- DB.update(ticket, amount)
    If !isNetworkAvailable() and !isLocalServer() Then
        sendDataToLocalServer(ticket)
    Endif
    return result
end function
```

Táto funkcia zistí, či je dostupné sieťové pripojenie na sieťovej karte daného zariadenia. Informáciu o dostupnosti siete sprostredkuje systém.

```
Function isNetworkAvailable()
    result <- System.isNetworkAvailable()
    return result
end function
```

Aplikácia potrebuje pravidelne prísun dát. Ak je WAN sieť nedostupná, dáta zabezpečí záložný server.

```
Function requestDataFromLocalServer()
    last_update <- DB.get('last_update')
    socket <- LocalState.localServer()
    data <- socket.send('REQUEST_DATA', last_update)
    return data
end function
```

Pomocná funkcia, ktorá umožní komunikáciu so záložným serverom.

```
Sub sendDataToLocalServer(data)
    socket <- LocalState.localServer()
    sendDataToSocket(socket, data)
end sub
```

Pomocná funkcia, ktorá sprostredkuje socketovú komunikáciu medzi zariadeniami.

```
Sub sendDataToSocket(socket, data)
    socket.send(data)
End sub
```

Aplikácia musí vedieť spracovať prijaté socketové dáta. Môže sa jednať o požiadavku o zaslanie dát alebo o uloženie zmenených dát.

```
Sub parseReceivedData(socket, data)
    decoded <- decode(data)
```

```
    If data.type = 'REQUEST_DATA' Then
        sendDataToSocket(socket, data)
    Else If data.type = 'STORE_DATA' Then
        DB.storeData(decoded)
    Endif
end sub
```

Proces, ktorý v pravidelnom intervale synchronizuje dáta z primárneho alebo záložného serveru podľa dostupnosti spojenia.

```
Async Sub periodicSynchronization()
    While app.isRunning()
        If isNetworkAvailable() Then
            return
        Else If LocalState.isServerAlive() Then
            data <- HTTP.get()
            DB.storeData(data)
        Else
            periodicFallbackSynchronization()
        Endif
        sleep 30000
    Endwhile
end Sub
```

Ak nie je dostupné WAN spojenie, aplikácia stiahne dáta zo záložného servera.

```
Async Sub periodicFallbackSynchronization()
    If isLocalServer() Then
        return
    Else
        If hasConnectionToLocalServer() Then
            data <- requestDataFromLocalServer()
            DB.storeData(data)
        Else
            searchForLocalServer()
        Endif
    Endif
end Sub
```

Ak je známa IP adresa a port, nadviaže sa spojenie so záložným serverom.

```
Function establishConnectionToLocalServer(ip, port)
    socket <- socket.open(ip, port)
    If socket.is_connected Then
```

```
        LocalState.setLocalServer(socket)
    Endif
    return socket
end function
```

Aplikácia, ktorá nie je označená ako záložný server, musí byť schopná vyhľadať záložný server v rámci lokálnej siete. Vyhľadávanie sa uskutočňuje pomocou lokálneho mapovania siete, kde sa jednotlivým účastníkom siete oskenujú otvorené porty. Keď daný port zodpovedá konfiguračnému parametru, tak sa aplikácia pokúsi nadviazať spojenie.

```
Function searchForLocalServer()
    devices <- scanLocalNetwork()
    For device in devices
        available_ports <- scanForOpenPorts(device)
        For port in available_ports
            If port = configuration.get('local_server_port') Then
                return establishConnectionToLocalServer(device.ip, port)
            Endif
        Endfor
    Endfor
end Function
```

Aj keď je to v tejto práci zjednodušené, aplikácie reálne generujú transakcie, ktoré musia byť synchronizované so serverom. Aplikácia v pravidelnom intervale nahráva zmeny na databázovej vrstve s primárnym serverom.

```
Async Sub periodicUpload()
    lastUpload <- Date.now()
    While app.isRunning()
        data <- DB.getChangedFrom(lastUpload)
        If data.size > 0 Then
            HTTP.post(data)
            lastUpload <- Date.now()
        Endif
        sleep 30000
    Endwhile
end sub
```

Proces, ktorý počúva na nové dáta zo socketového pripojenia.

```
Async Sub handleSocketData(socket)
    While app.isRunning()
```

```
        data <- socket.read()
        parseReceivedData(socket, data)
    Endwhile
end sub
```

Po spustení aplikácie je potrebné buď vytvoriť serverový socket záložného servera alebo inicializovať s ním spojenie.

```
Function createSocketConnection()
    If isLocalServer() Then
        return LocalState.localServer()
    Else
        return establishConnectionToLocalServer()
    Endif
end function
```

Tento proces zabezpečuje kontrolu s primárnym serverom. Systém môže mylne hlásiť dostupnosť siete, táto funkcia skontroluje reálnu existenciu tohto spojenia.

```
Async Sub handleServerPing()
    While app.isRunning()
        server_is_alive <- serverPing()
        LocalState.setServerIsAlive(server_is_alive)
    Endwhile
end sub
```

Vykonanie príkazu **ping** sieťovou knižnicou.

```
Function serverPing()
    return HTTP.ping()
end function
```

Pri štarte aplikácie inicializujeme všetky potrebné procesy a umožníme prácu s dátami.

```
socket <- createSocketConnection()
handleSocketData(socket)
periodicUpload()
periodicSynchronization()
handleServerPing()

barcode <- enterTicket()
is_network_available = isNetworkAvailable()
If is_network_available Then
    ticket <- fetchOnlineTicket(barcode)
```

```
Else
    ticket <- fetchLocalTicket(barcode)
Endif

is_valid_ticket <- validateTicket(ticket)
If is_valid_ticket Then
    output "Ticket is valid"
Else
    output "Ticket is invalid"
Endif

is_used_ticket <- useTicket(ticket, 5)
If is_used_ticket Then
    output "Ticket was used successfully"
Else
    output "Ticket is invalid"
Endif
```

## 2.4 Prototyp kombinovanej synchronizácie

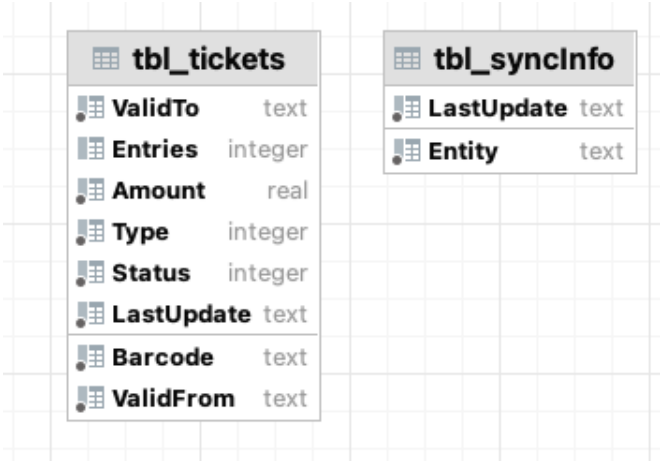
Funkčná ukážka prototypu pseudokódu (2.3.3) pozostáva zo serverovej časti a aplikačnej časti (klientov). Serverová časť je reprezentáciou primárneho servera a ukážkou zjednodušeného prostredia na správu zakúpených lístkov - vstupeniek. Mobilná aplikácia je reprezentáciou zjedodušeného modulu Point of sale (POS) systému, ktorý je používaní predajcami na validáciu platnosti vstupeniek. Aplikácia má schopnosť byť nastavená ako záložný server a teda v prípade výpadku spojenia s Internetom môže slúžiť ako záložný server.

Ukážku webového rozhrania a používateľského rozhrania mobilnej aplikácie je možno vidieť v Používateľskej príručke (A). Opis technickej časti prototypu, spolu s návodom ako projekt nastaviť, je dostupný v systémovej príručke (B). V nasledujúcich sekciách je opísaný technický koncept ako bolo pristúpené k programovaniu prototypu.

### 2.4.1 Zdieľaný databázový model

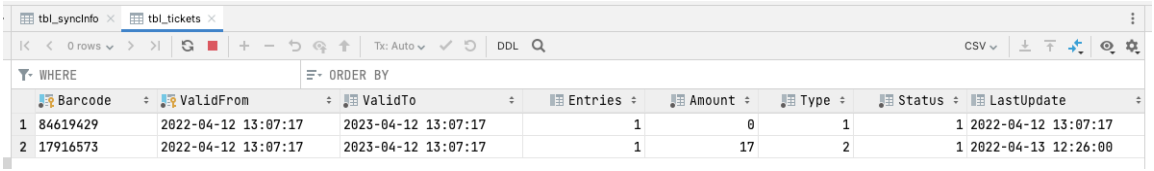
Pri tvorbe architektúry databázového modelu sme vychádzali z ideového konceptu (kapitola 2.3), podľa ktorého sme potrebovali namodelovať databázu (obrázok 2.4) obsahujúcu tabuľky *tbl_tickets* pre lístky - vstupenky (obrázok 2.6) a

tabuľku *tbl_syncinfo* (obrázok 2.5), ktorá bude uchovávať časové pečiatky synchronizácií tabuliek v aplikácii. V našom prípade bude obsahovať len jeden časový záznam o tabuľke *tbl_tickets*.



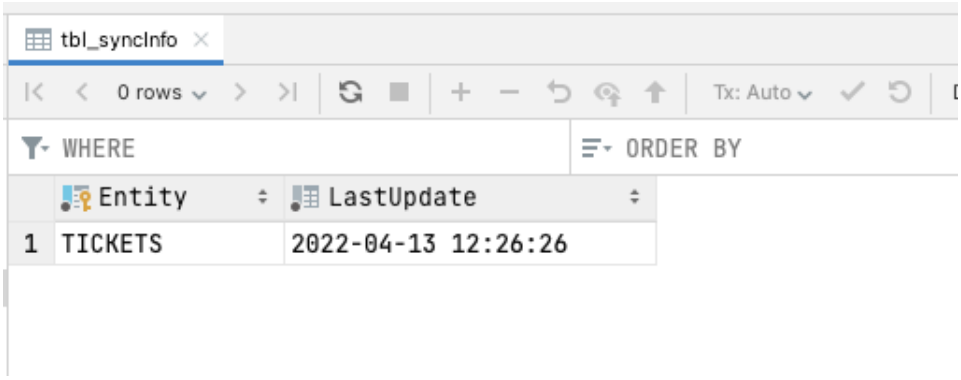

Obr. 2.4: Štruktúra prototypu databázy

Tabuľka s lístkami (*tbl_tickets*) pozostáva z čiarového kódu lístka, rozsahu platnosti, stavu platnosti (blokovaná/archivovaná), a typu. Lístky môžu predstavovať rôzne typy podľa použitia a to buď s počtom použití alebo so sumou, ktorú je možné uplatniť. Preto tabuľka obsahuje aj tieto stĺpce. Najdôležitejším údajom je ale stĺpec *LastUpdate*, ktorý zabezpečuje pravidelnú synchronizáciu všetkých aplikácií a musí byť preto pri každej zmene údajov aktualizovaný. Ak by sa stalo, že hodnota stĺpca *LastUpdate* by nebola zmenená, zariadenia by neboli schopné dostať sa ku zmenám z dôvodu využívania sťahovania zmien z databázy na základe porovnávania časovej pečiatky od poslednej sycnhronizácie.




Obr. 2.5: Tabuľka s lítkami

Tabuľka *tbl_syncInfo* je naopak nepotrebná na backendovej - serverovej časti aplikácie. Je to tabuľka výhradne používaná aplikáciami na udržovanie časových pečiatok všetkých synchronizovaných tabuliek, ktoré sa v aplikácii používajú. V našom prototype je to len jedna, **TICKETS**. Je nutné poznamenať, že v prípade, že by sa tento záznam nenachádzal v tabuľke, tak aplikácia musí previesť plnú synchronizáciu všetkých záznamov a vytvoriť nový záznam v tabuľke *tbl_syncInfo*.




Obr. 2.6: Tabuľka s časovými pečiatkami

Ďalším dôležitým faktorom pri funkčnosti synchronizácie je aj to, kedy aktualizovať tieto záznamy. Uveďme si príklad, keď aplikácia vykonáva pravidelnú synchronizáciu v *30* sekundovom intervale. V takomto prípade by sa vykonávala synchronizácia v nasledovných časoch: *10:00:00*, *10:00:30*, *10:01:00*, *10:01:30*, a podobne. Naskytuje sa otázka čo zapísať do tabuľky *tbl_syncInfo*. Musíme si uvedomiť to, že táto tabuľka neslúži na historizáciu informácii o úspešnosti synchronizácie, ale na reprezentáciu kotvy medzi jednotlivými zmenami v dátovej štruktúre. A z toho dôvodu je potrebné ukladať časovú pečiatku výhradne v prípade, ak je počet stiahnutých záznamov väčší ako nula. Ak sa nestiahnú žiadne záznamy v danom cykle intervalu, tak sa ani neaktualizuje časová pečiatka, a teda pri najbližšej iterácii bude synchronizácia v čase napríklad *10:01:30* používať časovú pečiatku *10:00:30* v ktorej boli naposledy stiahnuté nové zmeny. Ak sa stiahnú nové zmeny, uloží sa čas synchronizície *10:01:30*, v opačnom prípade zostane čas nezmenení.

Pre lepšie pochopenie si môžeme predstaviť aj nasledovnú modelovú situáciu. Majme dve mobilné zariadenia *A* a *B*, server *S* a zákazníkov *Z1* a *Z2* s rovnakým lístkom na jedno použitie *L*. Mobilné zariadenia sú v čase *10:00:00* zosynchronizované a majú všetky najnovšie informácie zo servera *S*. Mobilná aplikácia *B* prejde do režimu offline, v ktorom nemá spojenie ani so serverom *S* a ani s mobilnou aplikáciou *A*. V čase *10:00:30* príde zákazník *Z1* k predajcovi s mobilnou aplikáciou *B* a vykoná transakciu, pri ktorej uplatní lístok *L*. Kedže aplikácia *B* je v režime offline, neinformuje o tejto zmene server *S* a ani mobilnú aplikáciu *A*. V tom istom čase sa aplikácia *A* zosynchronizuje a keby uložila časovú pečiatku *10:00:30* pri ktorej nenastali z jej pohľadu žiadne zmeny, tak by preskočila údaje, ktoré by inak mohla zosynchronizovať. Z toho dôvodu pri opäťovnom oobnovení spojenia na mobilnej aplikácií *B* so serverom *S* nastane nahranie zmien, a server *S* tak nastaví v zázname lístka *L* novú časovú pečiatku *LastUpdate* rovnú aktuálnemu času *10:00:33*. A teda nastane synchronizácia na oboch mobilných zariadeniach

*A* aj *B*, ktoré stiahnú zmeny od času poslednej zmeny a to *10:00:00*. A preto keď príde zákazník *Z2* ku predajcovi s mobilnou aplikáciou *A* a pokúsil by sa použiť lístok *L* tak mu bude táto operácia zamietnutá. Tento algoritmus zároveň rieši situáciu s výpadkom spojenia s Internetom (WAN). Keby v modelovej situácii došlo ku strate spojenia WAN na mobilnej aplikácii *B*, ale nedošlo ku strate spojenia s lokálnou sieťou (LAN), tak aplikácia *B* pri uplatnení lástka *L* pošle túto zmenu záložnému serveru - aplikácii *A* alebo ak je aplikácia *B* v roli záložného servera, tak si aplikácia *A* stiahne túto zmenu pri synchronizácii v čase *10:00:30* od aplikácie *B*.

### 2.4.2 Backend aplikácie

Serverová (backendová) aplikácia v kontexte tejto práce pozostáva z jednej tabuľky *tbl_tickets* (obrázok 2.5) a zjednodušeného aplikačného programového rozhrania, ktoré sprostredkuváva dáta, ktoré mobilné aplikácie požadujú. V zdrojových súboroch sa nachádza aplikácia vyvinutá pomocou technológie Node.js a za použitia rámca Express.js. Táto aplikácia obsahuje viacero koncových bodov, ktoré je možné použiť pre prácu s databázou. Komunikácia prebieha pomocou HTTP požiadaviek a JSON formátu.

Základnú štruktúru odpovede zo servera môžeme vidieť v ukážke nižšie. Pozostáva zo statusu a dátovej časti. Status reprezentuje úspešnosť požiadavky a dáta sú vrátene v podobe zoznamu dát.

```
{
    "status": "success",
    "data": []
}
```

Aplikácia vyonáva nasledovné API požiadavky v pravidelných intervaloch.

```
http://192.168.100.43:8000/sync/?table=tickets
&filter=LastUpdate%3E%222022-04-15%2011%3A27%3A32%22
```

Požiadavka pozostáva z adresy primárneho servera, parametru *table*, ktorý odkazuje na zdrojovú tabuľkú dát, ktorú aplikácia požaduje, a parametru *filter*, ktorý slúži ako SQL filter.

Keď dekódujeme *filter*, dostaneme výraz porovnávajúci stĺpec *LastUpdate* s časovou peťiatkou poslednej synchronizácie.

```
LastUpdate>"2022-04-15 11:27:32"
```

Pri prvotnej synchronizácii apikácia vyžiada celú tabuľku a parameter *filter* preto nebude prítomný. Ukážka návratových dát je nasledovná:

```
{
    "status": "success",
    "data": [
        {
        "Barcode": "84619429",
        "ValidFrom": "2022-04-12 13:07:17",
        "ValidTo": "2023-04-12 13:07:17",
        "Entries": 10,
        "Amount": 0,
        "Type": 1,
        "Status": 1,
        "LastUpdate": "2022-04-15 11:25:21"
        },
        {
        "Barcode": "17916573",
        "ValidFrom": "2022-04-12 13:07:17",
        "ValidTo": "2023-04-12 13:07:17",
        "Entries": 1,
        "Amount": 10,
        "Type": 2,
        "Status": 1,
        "LastUpdate": "2022-04-16 12:57:46"
        }
    ]
    }
```

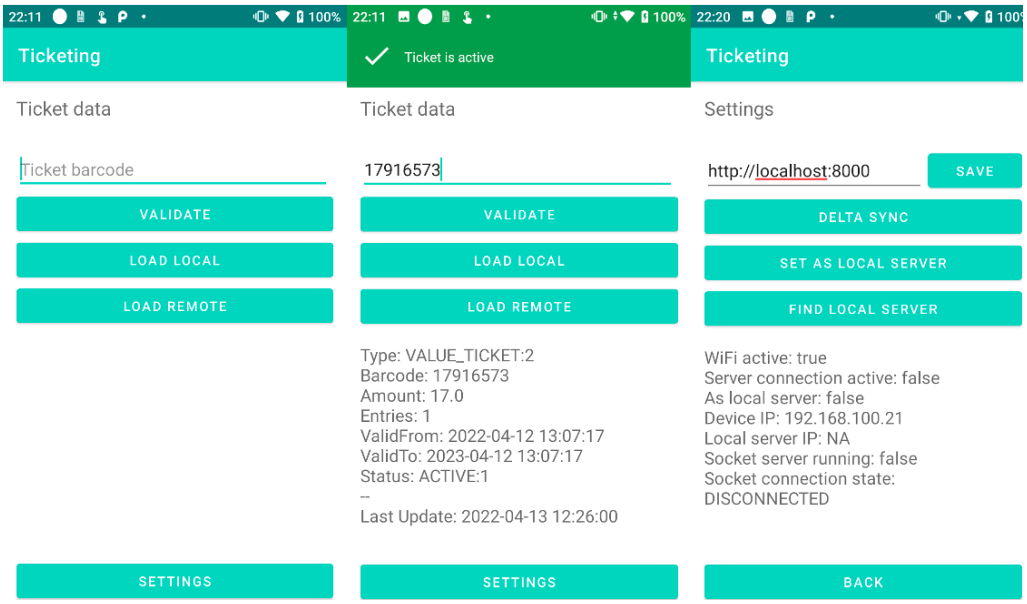

Môžeme vidieť pole objektov lístkov. Každý lístok kopíruje štruktúru tabuĺky *tbl_tickets*, aby aplikácia mohla dáta jednoducho premapovať na cieľovú tabuľku a uložiť ich do databázy.

### 2.4.3 Mobilná aplikácia

Prototyp aplikácie (obrázok 2.7) sme realizovali ako natívnu aplikáciu operačného systému Android. Využili sme na to moderný jazyk Kotlin a niekoľko verejne dostupných knižníc. Aplikácia pozostáva z dvoch obrazoviek - Skenovanie lístkov a Nastavení.

Po prvotnom nainštalovaní aplikácie sa vytvorí a inicializuje rovnaká štruktúra databázy ako bola opísaná v kapitole 2.4.1. Ak dôjde ku prvotnému pripojeniu na primárny server v aplikácii pomocou obrazovky nastavenia, tak aplikácia vykoná replikáciu dát zo servera. V opačnom prípade máme možnosť vykonať

vyhľadanie lokálneho záložného servera. Po jeho pripojení sa dáta zosynchronizujú.




Obr. 2.7: Mobilná aplikácia

Tento prototyp spolu so pseudokódom je ukážkou funkčného riešenia problémového fungovania aplikácii bez dostupnosti Internetového pripojenia. Pseudokód by malo byť možné vďaka jeho univerzálnosti aplikovať na rôzne dátové štruktúry a využiť ho pri mnohých problémoch.

# 3 Vyhodnotenie

Táto práca bola koncipovaná formou vytvárania pseudokódu spolu s funkčným prototypom aplikácie, ktorá daný kombinovaný synchronizačný algoritmus znázorňuje vo funkčnej, interaktívnej podobe. Vďaka robustnému backendu aplikácie (nie backendu servera) bolo možné dôkladne sledovať a analyzovať správanie aplikácie v rôznych situáciách, do ktorých sme uviedli testované zariadenie.

Pri profilovaní aplikácie sme neváhali využiť viaceré hardvérové zariadenia značky *Daishin DS6* spolu so zariadením *Zebra MC2200*. Aby sme mali dostatok vzoriek, tak sme pri testovaní neváhali a využili sme aj emulované zariadenia sprostredkované integrovaným vývojovým prostredím Android Studio.

Testovali sme prevažne stabilitu aplikácie, algoritmu a integritu prenášaných dát. Tiež sme sa zamerali na aktuálnosť dát po ich alternácii na rôznych zariadeniach pri rôznom stave siete. Výsledky testovania sú opísané v nasledujúcich tabuľkách, v ktorých sú zhrnuté jednotlivé aplikačné problémové okruhy 1.4.

Tabuľka 3.1: Testovací scenár č. 1 - Výpadok spojenia WAN

| Testovací scenár č. 1 | |
|---|---|
| Názov | Výpadok spojenia WAN |
| Účel | Overenie funkčnosti načitavania lístkov z lokálnej databázy pri výpadku WAN |
| Postup | Prvotné zosynchronizovanie aplikácie, Vypnutie Wi-Fi v systéme Android, Zadanie čiarového kódu lístka |
| Očakávaný výsledok | Úspešné overenie lístka |
| Výsledok | OK |

Tabuľka 3.2: Testovací scenár č. 2 - Výpadok spojenia WAN, funkcia záložného servera

| Testovací scenár č. 2 | |
|---|---|
| Názov | Výpadok spojenia WAN, funkcia záložného servera |
| Účel | Overenie funkčnosti synchronizácie zo záložného servera |
| Postup | Zresetovanie aplikácie (vymazanie dát), Aplikácia nebola nikdy pripojená na primárny server, Vyhľadanie záložného servera pomocou tlačidla v nastaveniach, Zadanie čiarového kódu lístka |
| Očakávaný výsledok | Úspešné overenie lístka |
| Výsledok | OK |

Tabuľka 3.3: Testovací scenár č. 3 - Výpadok spojenia WAN, úpdava dát na serveri

| Testovací scenár č. 3 | |
|---|---|
| Názov | Výpadok spojenia WAN, úpdava dát na serveri |
| Účel | Overenie funkčnosti synchronizácie cez záložný server |
| Postup | Prvotné zosynchronizovanie aplikácie A, Prvotné zosynchronizovanie aplikácie B, Nastavenie aplikácie B ako záložného servera, Vyhľadanie záložného servera pomocou tlačidla v nastaveniach aplikácie A, Prenastavenie primárneho servera na neexistujúci v aplikácii A, Úprava dát na primárnom serveri, Zadanie čiarového kódu lístka v aplikácii A |
| Očakávaný výsledok | Úspešné overenie lístka a jeho hodnota musí byť zhodná s aktuálnou hodnotou na serveri |
| Výsledok | OK |

Tabuľka 3.4: Testovací scenár č. 4 - Úprava dát na mobilnom zariadení

| Testovací scenár č. 4 | |
|---|---|
| Názov | Úprava dát na mobilnom zariadení |
| Účel | Overenie funkčnosti úpravi dát na mobilnej aplikácii |
| Postup | Prvotné zosynchronizovanie aplikácie, Zadanie čiarového kódu lístka v aplikácii, Úprava dát |
| Očakávaný výsledok | Úspešné načítanie lístka a možnosť jeho úpravy |
| Výsledok | Nie je možné updaviť dáta cez aplikáciu |
| Dôvod | Bolo by potrebné implementovať komplexný transakčný systém, čo je nad rámec synchronizačného algoritmu |

Tabuľka 3.5: Testovací scenár č. 5 - Obnova spojenia s primárnym serverom

| Testovací scenár č. 5 | |
|---|---|
| Názov | Obnova spojenia s primárnym serverom |
| Účel | Overenie funkčnosti pokračovania automatickej synchronizácie po obnovení WAN |
| Postup | Prvotné zosynchronizovanie aplikácie, Odpojenie aplikácie z WAN, Úprava dát na serveri, Obnovenia WAN spojenia v aplikácii, Načítanie lístka |
| Očakávaný výsledok | Lístok by mal obsahovať aktuálne dáta zo servera |
| Výsledok | OK |

Tabuľka 3.6: Testovací scenár č. 6 - Lokalizácia záložného servera

| Testovací scenár č. 6 | |
|---|---|
| Názov | Lokalizácia záložného servera |
| Účel | Overenie funkčnosti úpravi dát na mobilnej aplikácii |
| Postup | Prvotné zosynchronizovanie aplikácie A, Prvotné zosynchronizovanie aplikácie B, Nastavenie aplikácie B ako záložného servera, Vyhľadanie záložného servera pomocou tlačidla v nastaveniach aplikácie A |
| Očakávaný výsledok | Aplikácia automatizovane vyhľadá a pripojí sa na záložný server v LAN |
| Výsledok | OK |

# 4 Záver

Cieľom tejto diplomovej práce bolo vypracovať prehľad súčasného stavu v oblasti online a offline dátovej synchronizácie a vývoja aplikácií prístupom offline-first, taktiež bolo potrebné navrhnúť metódy a vzory pre tvorbu distribuovaných aplikácií s rôznymi modelmi dátovej synchronizácie. Ďalšou úlohou bolo vypracovať prípadové štúdie vzorovej realizácie distribuovaných aplikáciís využitím rôznych modelov dátovej synchronizácie a v neposlednom rade vyhodnotiť experimentálne výsledky prípadových štúdií.

Analytická časť tejto práce zahŕňa prípadové štúdie opisujúce riešenia dátovej synchronizácie v súčastnosti, taktiež opisuje a definuje distribuované systémy a dátovú synchronizáciu. V rámci definovania distribuovaných systémov sú spomenuté znaky, nevýhody, typy architektúr, schopnosť komunikácie a odolnosť voči chybám v distribuovaných systémoch.

Ďalej v tejto práci sa nachádza syntetická časť. Jej úlohou je definovať a vysvetliť proces synchronizácie ako v online-first tak aj v offline-first prístupe. Tieto prístupy boli implementované v ukážkovom prototype aplikácie. Syntetická časť obsahuje tiež pseudokód kombinovanej synchronizácie, pomocou ktorej je možné úspešne docieliť komunikáciu medzi zariadeniami v LAN bez dostupnosti Internetu.

V práci bol navrhnutý algoritmus formou pseudokódu, pričom bola dokázaná aj jeho funkčnosť na ukážkovom riešení formou aplikácie na operačnom systéme Android v spolupráci s backendovým riešením v Node.js. Funkčnosť riešenia bola otestovaná a vyhodnotená pomocou aplikačných okruhov, ktoré sme zhrnuli v testovacích scenároch, vďaka ktorým sme mohli efektívne testovať rôzne okrajové prípady synchronizačného algoritmu. Počas testovania sa objavilo množstvo nedostatkov návrhu pseudokódu, ktoré sme opravili a algoritmus sme zlepšili.

Algoritmus spĺňa všetky náležitosti, ktoré sme si v cieľoch diplomovej práce zadefinovali a podarilo sa nám implementovať aj funkčný prototyp, ktorý bude prínosný ako ukážka demo implementácie pre integráciu do väčších použitelných projektov. Pseudokód umožní navýšenie interakcie medzi zariadeniami a

zvýši aktuálnosť dát v prostredí bez stabilného Internetu. Do budúcna je možné algoritmus zlepšiť v medziach bezpečnosti, stability a spôsoboch náhradných komunikačných kanálov.

# Literatúra


1. VAN STEEN, Maarten; TANENBAUM, Andrew S. *Distributed systems*. Maarten van Steen Leiden, The Netherlands, 2017. ISBN 153028175X.
2. KLIMEŠ, Cyril. *Distribuované systémy. Texty pro distanční studium.* 1993. Diz. pr. Ostravská univerzita v Ostravě. Dostupné online: https://web.archive.org/web/201407 chazka/ds/SkriptaKlimes.pdf.
3. COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. *Distributed systems - concepts and design*. 3. vyd. Addison-Wesley Publishers Limited, 2001. ISBN 7-111-11749-2.
4. MILOJICIC, Dejan S; KALOGERAKI, Vana; LUKOSE, Rajan; NAGARAJA, Kiran; PRUYNE, Jim; RICHARD, Bruno; ROLLINS, Sami; XU, Zhichen. Peer-to-Peer Computing. [B. r.], s. 52.
5. STEINMETZ, Ralf. *Peer-to-Peer Systems and Applications*. Springer Science & Business Media, 2005. ISBN 978-3-540-29192-3. Google-Books-ID: A8CLZ1FB4qoC.
6. KUMAR, D Giridhar; SUBBARAO, Ch D V. Peer – To – Peer Computing: Architectures, Applications and Challenges. 2019, roč. 8, č. 1, s. 7.
7. LEDVINA, Jiří. *Přednášky z distribuovaných systémů*. 2002. Dostupné tiež z: `https://zcu.arcao.com/kiv/ds/ds.pdf`.
8. MAHMOUD, Qusay. *Middleware for Communications*. John Wiley & Sons, 2004. ISBN 978-0-470-86206-3. Google-Books-ID: ZwGhIZGxi7oC.
9. SINHA, Pradeep K. *DISTRIBUTED OPERATING SYSTEMS* [online]. [B. r.] [cit. 2021-12-21]. Dostupné z: `https://books.google.com/books/about/DISTRIBUTED_OPERATING_SYSTEMS.html?hl=sk&id=SewHKWac2I4C`.
10. HENRIKSSON, Daniel. *Distributed Systems*. 2008. Dostupné tiež z: `http://www8.cs.umu.se/kurser/5DV020/HT08/slides/lecture01.pdf`.
11. *Difference between RPC and RMI* [online]. 2019 [cit. 2021-12-21]. Dostupné z: `https://www.geeksforgeeks.org/difference-between-rpc-and-rmi/`. Section: Operating Systems.

12. SYNOPSYS, INC. *What Is Cryptography and How Does It Work?* [Online]. [B. r.] [cit. 2022-02-11]. Dostupné z: `https://www.synopsys.com/glossary/what-is-cryptography.html`.

13. WOXAPP. *How to Ensure the Security In P2P* [online]. [B. r.] [cit. 2022-02-10]. Dostupné z: `https://woxapp.com/our-blog/security-in-a-p2p-app-peer-to-peer-the-main-issues-you-should-be-aware-of/`.

14. LI, James. *A Survey of Peer-to-Peer Network Security Issues* [online]. [B. r.] [cit. 2022-02-10]. Dostupné z: `https://www.cse.wustl.edu/~jain/cse571-07/ftp/p2p/`.

15. CYBERSECURITY. *Peer-2-Peer Networking* [online]. 2018 [cit. 2022-02-10]. Dostupné z: `https://cybersecurity.osu.edu/cybersecurity-you/avoid-threats/peer-2-peer-networking`.

16. JOHNSON, M Eric; MCGUIRE, Dan; WILLEY, Nicholas D. The Security Risks of Peer-to-Peer File Sharing Networks. [B. r.], s. 24.

17. CLOUDFLARE, INC. *What is Transport Layer Security? | TLS protocol* [online]. [B. r.] [cit. 2022-02-10]. Dostupné z: `https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/`.

18. SCARPATI, Jessica. *What is Network Time Protocol (NTP)? - Definition from WhatIs.com* [online]. [B. r.] [cit. 2021-12-18]. Dostupné z: `https://www.techtarget.com/searchnetworking/definition/Network-Time-Protocol`.

19. HANSMANN, Uwe; METTALA, Riku M.; PURAKAYASTHA, Apratim; THOMPSON, Peter. *SyncML: Synchronizing and Managing Your Mobile Data*. Prentice Hall Professional, 2003. ISBN 978-0-13-009369-1.

20. MCCORMICK, Zach. Data Synchronization Patterns in Mobile Application Design. In: *Data Synchronization Patterns in Mobile Application Design*. Vanderbilt University, 2012, s. 14.

21. FUNDINGER, Danny. *Offline data synchronization - Strategies to address offline data synchronization for mobile apps* [online]. 2019 [cit. 2021-12-30]. Dostupné z: `https://developer.ibm.com/articles/offline-data-synchronization-strategies/`.

22. MELAMED, Tom. *Offline-first: what is it, and how could your app benefit? - Calvium* [online]. 2018 [cit. 2021-12-30]. Dostupné z: `https://calvium.com/offline-first-what-is-it-and-how-could-your-app-benefit/`.

23. RXDB. *About Offline First* [online]. [B. r.] [cit. 2022-02-09]. Dostupné z: `https://rxdb.info/offline-first.html`.

24. FUNDINGER, Danny. *Strategies to address offline data synchronization for mobile apps* [online]. 2019 [cit. 2022-02-08]. Dostupné z: https://developer.ibm.com/articles/offline-data-synchronization-strategies/.

25. KEXUGIT. *The Complete Synchronization Process - Part 4: Delta/Full Import/Synchronization Explained* [online]. [B. r.] [cit. 2021-12-30]. Dostupné z: https://docs.microsoft.com/en-us/archive/blogs/connector_space/the-complete-synchronization-process-part-4-deltafull-importsynchronization-explained.

26. AL-SAYYED, Rizik M. H.; NAMOUS, Feras F.; ALKHALAFAT, AlMonther H.; AL-SHBOUL, Bashar; AL-SAQQA, Samar. New Synchronization Algorithm Based on Delta Synchronization for Compressed Files in the Mobile Cloud Environment. *International Journal of Communications, Network and System Sciences* [online]. 2017, roč. 10, č. 4, s. 59–74 [cit. 2021-12-30]. Dostupné z DOI: 10.4236/ijcns.2017.104004. Number: 4 Publisher: Scientific Research Publishing.

27. IBM. *Cloudant - Overview* [online]. 2022 [cit. 2022-02-07]. Dostupné z: https://www.ibm.com/cloud/cloudant.

28. IBM. *What is replication?* [Online]. [B. r.] [cit. 2022-02-07]. Dostupné z: https://cloud.ibm.com/docs/Cloudant?topic=Cloudant-replication-guide.

29. POUCHDB. *About PouchDB* [online]. [B. r.] [cit. 2022-02-09]. Dostupné z: https://pouchdb.com/learn.html.

30. POUCHDB. *FAQ* [online]. [B. r.] [cit. 2022-02-09]. Dostupné z: https://pouchdb.com/faq.html.

31. IBM. *Offline First Case Studies* [online]. [B. r.] [cit. 2022-02-07]. Dostupné z: http://offlinefirst.org/casestudies/.

32. BRIDGEFY, INC. *Offline Messages App & SDK* [online]. 2021 [cit. 2022-02-08]. Dostupné z: https://bridgefy.me/.

33. BRIDGEFY, INC. *How does the Bridgefy App Work?* [Online]. 2021 [cit. 2022-02-08]. Dostupné z: https://bridgefy.me/how-does-the-bridgefy-app-work/.

34. BRIDGEFY, INC. *SDK* [online]. 2021 [cit. 2022-02-08]. Dostupné z: https://bridgefy.me/sdk/.

35. ROGERS, Michael. *How it works - Briar* [online]. [B. r.] [cit. 2022-02-08]. Dostupné z: https://briarproject.org/how-it-works/.

36. CHAUMET, Alexandre. *AERO, La solution mobilité offline développée par eFrontech* [online]. 2017 [cit. 2022-02-08]. Dostupné z: `https://efrontech.com/aero-la-solution-mobilite-offline/`.

37. HOSPITALRUN. *HospitalRun - Open source software for developing world hospitals* [online]. [B. r.] [cit. 2022-02-09]. Dostupné z: `https://hospitalrun.io/`.

38. HOSPITALRUN. *Why HospitalRun?* [Online]. 2016 [cit. 2022-02-09]. Dostupné z: `https://hospitalrun.io/blog/why-hospitalrun/`.

# Zoznam skratiek

**API** Application programming interface.

**BTS** Base Station Transceiver Subsystem.

**CRM** Customer relationship management.

**EHR** Electronic Health Record.

**FTP** File Transfer Protocol.

**HIS** Hospital Information System.

**HTTP** Hypertext Transfer Protocol.

**IP** Internet Protocol address.

**ISP** Internet service provider.

**JSON** JavaScript Object Notation.

**LAN** Local Area Network.

**MAN** Metropolitan Area Network.

**NAT** Network address translation.

**NFC** Near Field Communication.

**OS** Operating system.

**P2P** Peer-to-Peer.

**POP3** Post Office Protocol.

**POS** Point of sale.

**RMI** Remote Method Invocation.

**RPC** Remote Procedure Call.

**RR** Request/Reply.

**RRA** Request/Reply/Acknowledge-Reply.

**SDK** Software Development Kit.

**SMTP** Simple Mail Transfer Protocol.

**SQL** Structured Query Language.

**SSL** Secure Sockets Layer.

**TLI** Transport Layer Interface.

**TLS** Transport Layer Security.

**UX** User experience.

**VoIP** Voice over IP.

**WAN** Wide area network.

**WWW** World Wide Web.

# Slovník

**blockchain** Blockchain je distribuovaná databáza, ktorá je zdieľaná medzi uzlami počítačovej siete. Ako databáza uchováva blockchain informácie v elektronickej podobe v digitálnom formáte. Každý blok v reťazci dostane pri pridaní do reťazca presnú časovú pečiatku.

**multitenancy** V oblasti cloud computingu znamená viacnásobné využívanie (multitenancy), že viacerí zákazníci dodávateľa cloudu využívajú tie isté výpočtové zdroje. Napriek tomu, že zdieľajú zdroje, zákazníci cloudu o sebe navzájom nevedia a ich údaje sú úplne oddelené.

**záložný server** Je typ servera, ktorý slúži ako zástupca servera v čase nedostupnosti primárneho servera. Zaručuje tak zvýšenú dostupnosť služieb v prípade sieťových chýb.

# Zoznam príloh

**Príloha A** CD médium – záverečná práca v elektronickej podobe,

**Príloha B** Používateľská príručka,

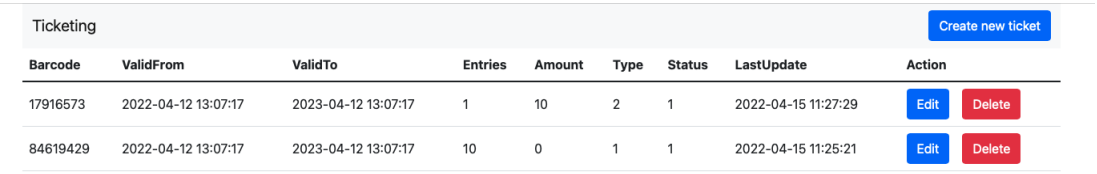
**Príloha C** Systémová príručka

# A Používateľská príručka

## Webové rozhranie na správu lístkov

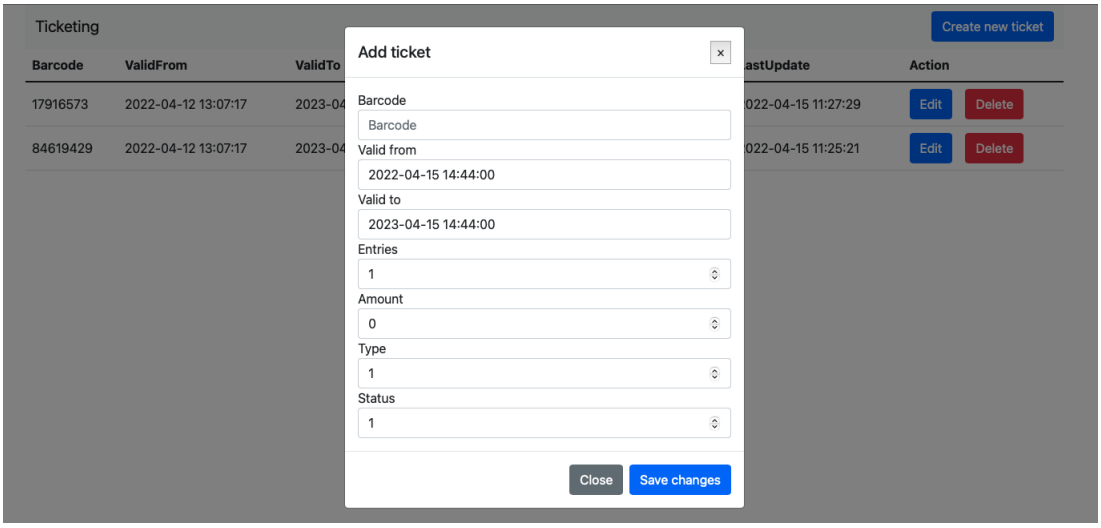
Pre spustenie webového rozhrania je potrebné pristúpiť na adresu *localhost:8000/web*, prípadne ak je aplikácia nakonfigurovaná na inom špecifickom porte, tak je nutné použiť správnu adresu.

Po jeho otvorení môžeme vidieť zoznam všetkých lístkov (obrázok A.1) prístupných v databáze, pridávať nové, upravovať a mazať ich.



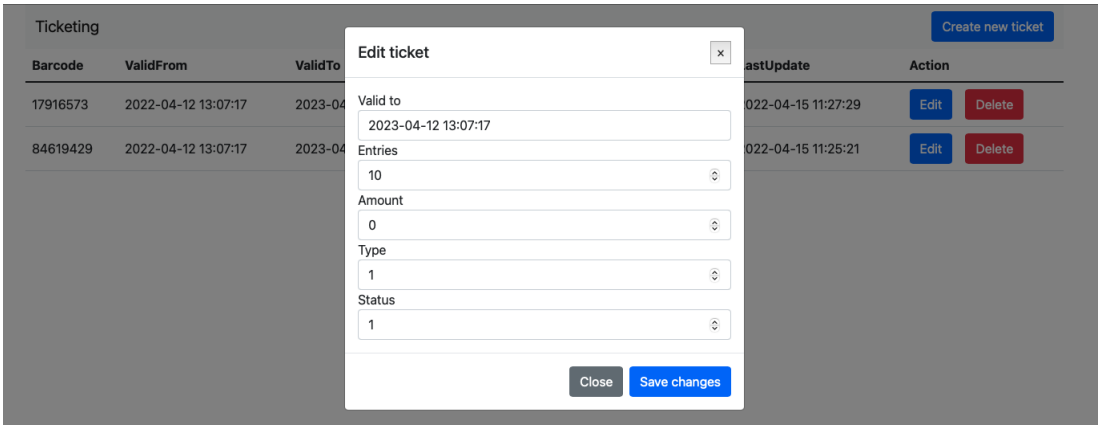

Obr. A.1: Server - Tabuľka s lítkami

Pridávanie nového lístka (obrázok A.2) zobrazí modálne okno, v ktorom je potrebné správne vyplniť všetky požadované údaje. Časové pečiatky validity musia dodržiavať formát (*yyyy-MM-dd HH:mm:ss*). Typ lístka *1* reprezentuje lístky s pevným počtom použití a typ lístka *2* repzrezetuje typ lístka s kreditovou hodnotou. Stav lístka môže byť reprezentovaný hodnotami *0* - neplatný alebo hodnotou *1* - aktívny.




Obr. A.2: Server - Pridanie nového lístka

Úprava lístka (obrázok A.3) zabezpečí automatické nastavenie správnej časovej pečiatky v stĺpci *LastUpdate*, a bude preto dostupná v krátkom časovom horizonte na všetkých aktívnych mobilných zariadeniach.



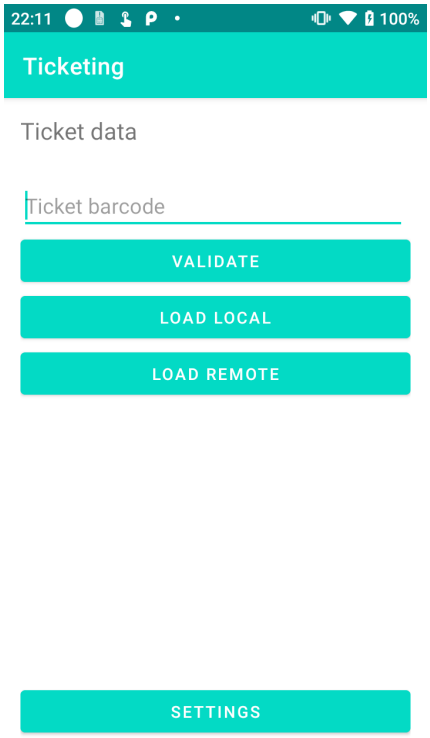

Obr. A.3: Server - Úprava lístka

## Používateľské rozhranie mobilnej aplikácie

Na obrázku A.4 je vidno úvodnú obrazovku aplikácie. Táto obrazovka slúži na skenovanie lístkov. Po zadaní čiarového kódu lístka máme na ukážku tri možnosti:

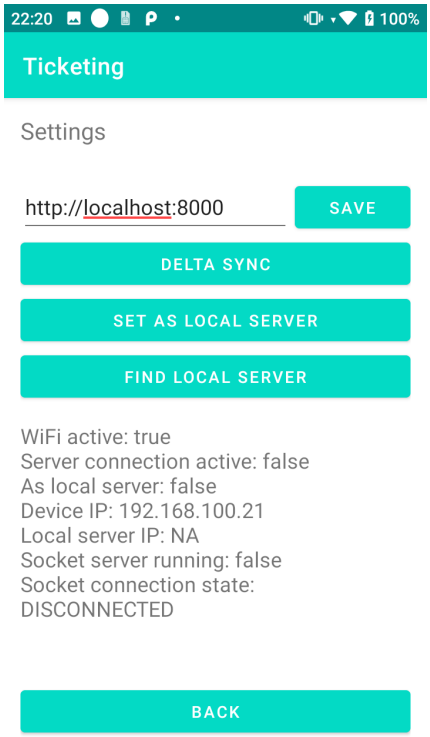
- **Load Remote** - reprezentuje online-first prístup k spravovaniu dát, čiarový kód sa odošle na backendový server, ktorý vráti dáta vo formáte JSON. Tieto dáta sa uložia do lokálnej databázy a vyrenderujú sa na obrazovke pre užívateľa.

- **Load Local** - reprezentuje offline-first prístup k spravovaniu dát, dáta sú synchronizované na pozadí a po zadaní čiarového kódu sa načítajú z lokálnej databázy a vyrenderujú sa na obrazovke pre užívateľa.
- **Validate** - toto tlačidlo má byť ukážkou kombinovaného riešenia pre synchronizáciu dát. Po jeho stlačení dochádza k načítaniu dát podľa diagramu 2.3. Dáta sa načítajú v závislosti podľa dostupnosti siete.



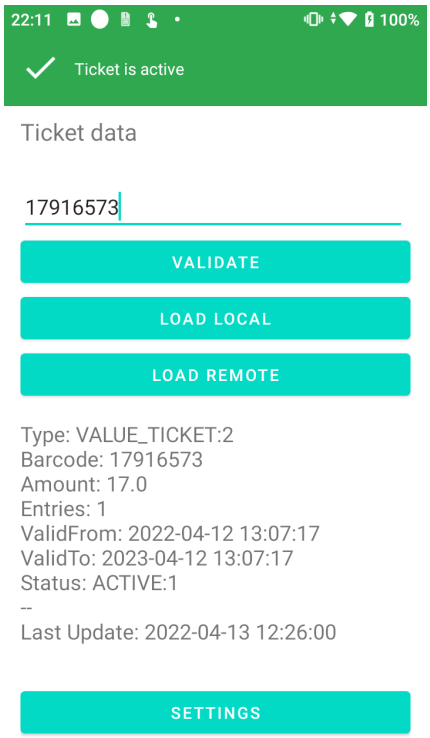

Obr. A.4: Úvodná obrazovka aplikácie

Na obrazovke nastavení A.5 môžeme vidieť vstupné pole pre zadanie adresy primárneho servera, z ktorého sa budú prioritne synchronizovať všetky dáta aplikácie. Taktiež vidíme tlačidlo **Delta Sync**, pomocou ktorého vieme manuálne zavolať delta synchronizáciu dát. Tiež tu máme tlačidlo **Set as Local Server** alebo **Set as Local Client**, pomocou ktorého vieme prepnúť režim socketového pripojenia, ktoré aplikácia používa. Buď to funguje ako záložný server alebo sa k nemu vie pripojiť. Na pripojenie a vyhľadanie záložného serveru v lokálnej sieti slúži tlačidlo **Find Local Server**. Pod tlačidlami je dostupný text, ktorý opisuje stav, v ktorom sa aplikácia aktuálne nachádza na prezentačné účely prototypu. Môžeme tu vidieť IP adresy, stav socketového spojenia, stav Wi-Fi pripojenia a stav spojenia s primárnym serverom.



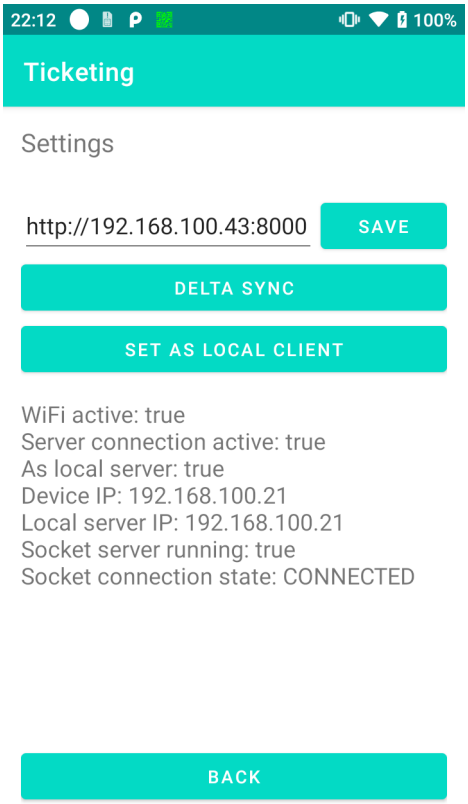

Obr. A.5: Aplikácia pripojená ku záložnému serveru

Po validácii lístka aplikácia zobrazí informačnú správu s výsledkom overenia A.6. Aplikácia vie rozlíšiť platnosť lístka, stav a hodnotu kreditu. Všetky informácie sa vypíšu aj v textovej podobe.




Obr. A.6: Validácia lístka

Na obrázku A.7 vidno aplikáciu v stave záložného servera. IP adresa mobilného zariadenia a IP adresa záložného servera je z toho dôvodu rovnaká.




Obr. A.7: Pripojenie na primárny server

# B Systémová príručka

## Príprava prostredia serverovej služby

Serverová časť práce sa nachádza v priečinku **server** a pozostáva zo zdrojových súborov využívajúcich Node.js aplikačný rámec.

Požiadavkou pre spustenie je nainštalovanie aplikácie *node* na hostiteľskom počítači.

Prvotnú inicializáciu vykonáme nasledovne:

```
npm install
```

Spustenie servera vykonáme príkazom:

```
npm start
```

Spustenie servera prebehne na preddefinovanom porte *8000*. Port je definovaný v súbore *server.js* v konštante *HTTP_PORT*. V prípade jeho zmeny je potrebné ho zmeniť aj v súbore *frontend/index.html*.

## Príprava prostredia mobilnej aplikácie

Mobilná aplikácia sa nachádza v priečinku **app** a pozostáva zo zdrojových súborov napísaných v jazyku Kotlin. Pre prácu s projektom odporúčame používať vývojové prostredie **Android Studio**, ktoré prevedie prvotnú inicializáciu všetkých potrebných súborov.