Masarykova univerzita
Fakulta informatiky

Mobilná aplikácia pre ovládanie systému výpožičky a inventarizácie pre laboratórium Siemens

Diplomová práca

Bc. Štefan Matta

Brno, jeseň 2018

Namiesto tejto stránky vložte kópiu oficiálneho podpísaného zadania práce a prehlásenie autora školského diela.

Prehlásenie

Prehlasujem, že táto diplomová práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Bc. Štefan Matta

Vedúci práce: doc. Ing. RNDr. Barbora Bühnová, Ph.D.

Poďakovanie

Moje poďakovanie patrí doc. Ing. RNDr. Barbore Bühnovej, Ph.D. za vedenie mojej diplomovej práce a Ing. Tomášovi Gavendovi, Ph.D. za pravidelné konzultácie a pripomienky.

Zhrnutie

Táto diplomová práca sa zaoberá analýzou súčasného riešenia inventarizačného systému v spoločnosti *Siemens* a taktiež návrhom a implementáciou dátovej služby a mobilnej aplikácie na platforme *android*. Práca obsahuje aj analýzu výsledkov z užívateľského testovania aplikácie a nasadenie aplikácie a dátovej služby vo firme *Siemens*.

Kľúčové slová

Android, mobilná aplikácia, Kotlin, Java, REST, inventarizácia

Obsah

Zoznam tabuliek

Zoznam obrázkov

1 Úvod

Mobilné zariadenia alebo *smartfóny* sa stali pre mnohých neoddeliteľnou súčasťou ich životov. Niet sa čomu diviť, tieto zariadenia majú obrovské využitie či už pre zábavu alebo prácu. Od príchodu prvého *smartfónu IBM Simon* z roku 1994 prešli obrovským technickým pokrokom. V súčasnosti skoro sto percent predaných zariadení má operačný systém od jednej z dvoch najväčších mobilných platforiem, a to *android* alebo *iOS*. Tieto platformy poskytujú vývojárom možnosti vývoja mobilných aplikácií a užívateľom možnosť rozšíriť funkcionalitu svojho zariadenia. Aktuálne je spolu dostupných skoro 6 miliónov aplikácií pre obe platformy a čísla sa z roka na rok zvyšujú [1]. Narozdiel od stolových počítačov mobilita *smartfónov* umožňuje užívateľom zariadenie používať skoro kdekoľvek. Tento rozdiel prináša výhody pre rôzne činnosti. Príkladom môže byť aj proces inventarizácie vo firme *Siemens*, kde by prítomnosť mobilnej aplikácie umožnila vykonať záznam o inventúre, revízii alebo kalibrácii zariadenia na mieste, a tým urýchliť tento proces.

Cieľom tejto práce je analýza súčasnej webovej verzie inventarizačného systému vo firme *Siemens*. Na základe analýzy bude navrhnutá mobilná aplikácia, ktorá bude slúžiť na ovládanie súčasného inventarizačného sytému. Aplikácia bude implementovaná pre platformu *android* a prototyp aplikácie bude otestovaný zamestnancami firmy *Siemens*. Cieľom práce je taktiež nasadenie aplikácie v prostredí firmy *Siemens*.

Práca je rozdelená do siedmich kapitol. Druhá kapitola sa zaoberá analýzou súčasného riešenia webového inventarizačného systému, ktorý sa používa vo firme *Siemens*. Tento systém pomáha zamestnancom vypožičiavať si zariadenia z laboratórií, vykonávať elektrické a kalibračné revízie, a taktiež každoročne vykonávať inventúru týchto zariadení. V mnoho smeroch zjednodušil prácu zamestnancov a nahradil neefektívne procesy, ktoré sa používali pred jeho nasadením. Tento systém so sebou ale nesie zopár nedokonalostí, ktoré mu bránia v tom, aby sa dal jednoduchšie používať. Ako bolo spomenuté, ide o webovú aplikáciu, tá však nie je prispôsobená pre mobilné zariadenia, a to napríklad v prípade inventarizácie prináša problém, kedy je nutné

ručne zadávať hodnoty do systému, čo z užívateľského hľadiska nie je prívetivé.

Tretia kapitola sa zaoberá návrhom *android* aplikácie, ktorá by tieto nedokonalosti riešila. Súčasné riešenie však neposkytuje žiadnu dátovú službu, ktorú by táto aplikácia mohla využívať na číťanie a modifikáciu dát. Preto hlavnou súčasťou aplikácie je aj návrh dátovej služby, ktorá využíva existujúci dátovy model webovej verzie. Cieľom tejto kapitoly je aj výber technológii a *frameworkov*, ktoré budú použíté pre vývoj aplikácií. Kapitola popisuje architektonický návrh riešenia dátovej služby, *android* aplikácie a ich spoločnej komunikácie.

Štvrtá kapitola sa zaoberá konkrétnou implementáciou a popisuje spôsob akým bola vytvorená. Piata kapitola sa venuje testovaniu použiteľnosti mobilnej aplikácie zamestnancami firmy *Siemens*. Testovanie slúžilo na overenie prívetivosti a vylepšenie aplikácie. Záver práce sa zaoberá nasadením dátovej služby a mobilnej aplikácie v prostredí firmy *Siemens*.

2 Analýza

Táto kapitola sa zaoberá analýzou súčasného riešenia inventarizačného systému, ktorý v súčasnosti používa firma *Siemens*. Výsledok tejto analýzy bude použitý pre analýzu riešenia mobilnej aplikácie a jej dátovej služby.

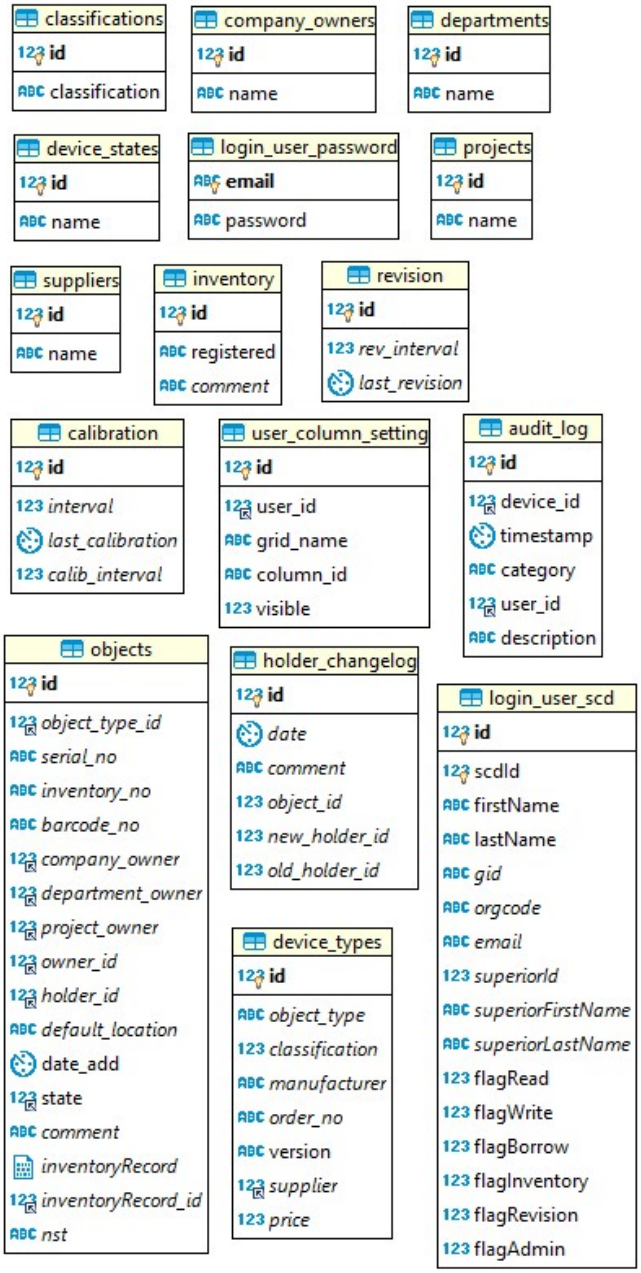
2.1 Súčasný stav

2.1.1 Siemens Inventory

V súčasnosti je inventarizačný systém vo firme *Siemens* riešený ako webová aplikácia založená na open source frameworku *Vaadin* určenom na vývoj webových aplikácií. *Vaadin* obsahuje množstvo grafických komponentov, a tak uľahčuje vývoj [2]. Bohužiaľ, webová aplikácia bola počas vývoja *android* aplikácie a dátovej služby stále vo vývoji. Preto bolo potrebné niekoľkokrát upravovať design, či dátové štruktúry. Kód aplikácie nebol verzovaný žiadnym verzovacím systémom a bolo obtiažne zistiť zmeny v implementácii medzi jednotlivými verziami. Nasledujúce podkapitoly bližie analyzujú jednotlivé časti aplikácie.

2.1.2 Ukladanie dát a dátový model

Dátový model inventarizačného systému bol vyvíjaný tak, aby zachoval rovnakú dátovú štruktúru pre spätnú kompatibilitu s predchádzajúcim systémom. To spôsobilo, že bolo potrebné robiť ústupky pri návrhu nových tabuliek, a tak má dátový model v súčasnosti ďaleko od ideálneho návrhu, kedže niektoré tabuľky nespĺňanú ani *2. normálovú formu* ([3, Kapitola 3.4]). Databázové tabuľky neobsahuju ani cudzie kľúče aj napriek tomu, že sú v relácii. Príkladom je tabuľka *objects* (obrázok 2.1), ktorá má logicky asociáciu na tabuľky ako *company_owners, departments, device_types*, no tieto asociácie nie sú definované cudzím kľúčom. Pre persistenciu bola použitá databáza *MySQL*. Ide o *open source SQL* relačný databázový server podporovaný viacerými platformami.



Obr. 2.1: Databázové schémy tabuliek

2.1.3 Backend aplikácie

Entity

Pre reprezentáciu dát z databázy je vytvorená ich objektová podoba v kóde. Každá tabuľka z obrázka 2.1 je namapovaná na entitu vytvorenú pomocou *Java JPA* (*Java Persistence API*) a *Hibernate*. Tieto entity sa nachádzajú v balíku *cz.siemens.inventory.model*. Je nutné poznamenať, že niektoré entity obsahujú aj inštancie *DAO* objektov priamo v sebe, čo nie je správny prístup. Nespĺňajú tak *SOLID* princíp jedinej zodpovednosti, kedže entita reprezentujúca dáta je zodpovedná aj za persistenciu a vytvára to závislosť entity na *DAO* objekte.

Vrstva prístupu k databáze - *Data Access Layer (DAO)*

DAO vrstva zodpovedá za persistenciu entít a selekciu dát z databázy. Predstavuje aj určitú mieru abstrakcie, pretože nevystavuje navonok detaily použitej databázy. Existujú *frameworky* napríklad *Spring framework*, ktorý sa postará o správu databázových transakcií, vytvorenie DAO vrstvy a operácie nad dátami v princípe len rozšírením rozhrania. V tomto prípade ale bol použitý manuálny spôsob operácie nad dátami (príklad 1). V príklade 2 je implementácia *DAO* vrstvy a jej *CRUD* operácií pre entitu *DeviceInternal*.

Príklad 1 *Ukážka implementácie create operácie v DAO vrstve*

```
public class GenericDao<E extends Object> {
    private Class<E> entityClass;
    protected SessionFactory sessionFactory;

    public void create(E entity) {
        Session session = sessionFactory.openSession();
        session.beginTransaction();
        session.save(entity);
        session.getTransaction().commit();
        session.close();
    }
}
```

Príklad 2 *Ukážka implementácie DAO vrstvy pomocou Spring Framework Rozšírenie zozhrania JpaRepository stačí na vytvorenie CRUD operiácii DAO vrstvy pre entitu DeviceInternal.*

```
public interface DeviceDao
        extends JpaRepository<DeviceInternal, Long> {
}
```

Takéto riešenie je čitateľnejšie a jednoduchšie na údržbu.

Existujúce riešenie sa však nedá využiť pri implementácií nového riešenia, a to z dôvodu, že aplikácia neposkytuje žiadnu knihovňu, ktorá by umožnovala akokoľvek rozšíriť alebo použiť existujúce riešenie. Na základe vyššie spomenutých problémov je vrstva *dátového prístupu* pre dátovú službu vytvorená refaktorovaním zdrojových kódov existujúceho riešenia a je vytvorená knihovňa, ktorá zjednoduší budúci vývoj či rozšírenie aplikácie.

2.1.4 Frontend aplikácie

Vaadin framework podporuje vytváranie frontendovej časti aplikácie na backende. Neboli potrebné žiadne značkovacie jazyky ako *HTML* či *XML*. Vytváranie jednotlivých obrazoviek prebieha použitím preddefinovaných *Vaadin* komponentov, čím sa uľahčuje implementácia a je to jeden z dôvodov, prečo bol tento *framework* zvolený ([3, Kapitola 3.1.3]). Design jednotlivých obrazoviek je v princípe jednoduchý a predstavuje užívateľsky prívetivý pohľad do databázy. Nevýhodou ale je, že časť funkcionality a vzhľad nie je optimalizovaný na mobilné zariadenia. Súčasné riešenie poskytuje možnosť rozpoznania *QR* kódov, tie sa však musia do aplikácie nahrať ručne. Toto riešenie spôsobuje problém pri vykonávaní inventúry, kde je potrebné spracovať veľa záznamov za sebou a samotné nahrávanie a spracovanie *QR* kódu trvá určitú dobu.

2.2 Užívateľské požiadavky

Požiadavky sú popisy systémových služieb, ktoré musí systém poskytovať a obmedzenia s ktorými musí fungovať. Úroveň požiadaviek sa môže pohybovať od vysoko abstraktných až po detailné špecifikácie prostredníctvom matematických funkcií. Požiadavky delíme na funkčné a nefunkčné. Funkčné požiadavky definujú služby, ktoré ma systém poskytovať, ako má systém reagovať na určité vstupy a ako sa má správať v konkrétnych situáciách. Nefunkčné požiadavky sú vlastnosti a obmedzenia služieb poskytované systémom ako napríklad bezpečnosť a spoľahlivosť [4]. V nasledujúcich kapitolách môžete vidieť funkčné a nefunkčné požiadavky pre mobilnú aplikáciu.

2.2.1 Funkčné požiadavky

FR1 Prihlasovanie a odhlasovanie

- **FR1.1** do aplikácie ma prístup iba prihlásený užívateľ
- **FR1.2** užívateľ má možnosť sa z aplikácie odhlásiť

FR2 Zobrazenie a úprava zariadenia

- **FR2.1** užívateľ má možnosť prezerať a upravovať zariadenia
- **FR2.2** užívateľ má možnosť priradiť alebo zmeniť *QR* kód zariadenia pomocou skenovania nového *QR* kódu aplikáciou
- **FR2.3** na úpravu zariadenia sú potrebné dostatočné práva (*write*)

FR3 Zobrazenie požičaných zariadení

- **FR3.1** užívateľ má možnosť prezerať zariadenia, ktoré má práve požičané
- **FR3.2** na zobrazenie požičaných zariadení sú potrebné dostatočné práva (*borrow*)

FR4 Požičanie a vrátenie zariadenia

- **FR4.1** užívateľ má možnosť si požičať a vrátiť zariadenia
- **FR4.2** užívateľ má možnosť nájsť zariadenie v aplikácii pomocou manuálneho zadania sériového čísla, čísla *QR* kódu alebo pomocou skenovania *QR* kódu aplikáciou
- **FR4.3** história zmien aktuálneho vlastníka musí byť evidovaná
- **FR4.4** na požičanie a vrátenie zariadení sú potrebné dostatočné práva (*borrow*)

FR5 Elektrická revízia zariadenia

- **FR5.1** užívateľ má možnosť vykonať záznam o elektrickej revízii
- **FR5.2** užívateľ má možnosť vidiet zoznam zariadení a informácie poslednej a budúcej revízii
- **FR5.3** užívateľ má možnosť nájsť zariadenie v aplikácii pomocou manuálneho zadania sériového čísla, čísla *QR* kódu alebo pomocou skenovania *QR* kódu aplikáciou
- **FR5.4** história zmien revízie zariadenia musí byť evidovaná
- **FR5.5** na vykonanie záznamu o elektrickej revízii sú potrebné dostatočné práva (*revision*)

FR6 Kalibrácia zariadenia

- **FR6.1** užívateľ má možnosť vykonať záznam o kalibrácii zariadenia
- **FR6.2** užívateľ má možnosť vidiet zoznam zariadení a informácie poslednej a budúcej kalibrácii
- **FR6.3** užívateľ má možnosť nájsť zariadenie v aplikácii pomocou manuálneho zadania sériového čísla, čísla *QR* kódu alebo pomocou skenovania *QR* kódu aplikáciou
- **FR6.4** história zmien kalibrácie zariadenia musí byť evidovaná
- **FR6.5** na vykonanie záznamu o kalibrácii sú potrebné dostatočné práva (*revision*)

FR7 Inventúra

- **FR7.1** užívateľ má možnosť vykonať inventúru zariadení
- **FR7.2** užívateľ má možnosť vidiet zoznam zariadení a ich stav inventúry
- **FR7.3** užívateľ má možnosť nájsť zariadenie v aplikácii pomocou manuálneho zadania sériového čísla, čísla *QR* kódu alebo pomocou skenovania *QR* kódu aplikáciou
- **FR7.4** história zmien inventúry zariadenia musí byť evidovaná
- **FR7.5** na vykonanie záznamu inventúry sú potrebné dostatočné práva (*inventory*)

FR8 Typy zariadení, Dodávatelia, Oddelenia, Vlastníci a Projekty

- **FR8.1** užívateľ má možnosť vidieť zoznam všetkých typov zariadení, dodávateľov, oddelení, vlastníkov a projektov (ďalej len entity)
- **FR8.2** užívateľ má možnosť pridávať a modifikovať entity
- **FR8.3** na pridanie alebo modifikovanie entít sú potrebné dostatočné práva (*write*)

FR9 Profil

- **FR9.1** užívateľ má možnosť vidieť svoje oprávnenia
- **FR9.2** užívateľ má možnosť vidieť meno svojho nadriadeného

2.2.2 Nefunkčné požiadavky

- Mobilná aplikácia musí byť implementovaná pre platformu *android*
- Aplikácia musí využívať súčasný dátový model a databázu MySQL
- Aplikácia nemôže modifikovať existujúce databázové objekty
- Design mobilnej aplikácie musí byť zhodný s existujúcou webovou aplikáciou

3 Návrh

Pred začatím implementácie je potrebné si vytvoriť celkový pohľad na projekt a definovať jeho dizajn. Z požiadaviek je jasné, že je potrebné vytvoriť aplikáciu na platformu *android*. Čo však jasné nie je, je spôsob akým bude aplikácia komunikovať s databázou. Preto je nutné navrhnúť dátovú službu, ktorá bude prijímať a poskytovať dáta *android* aplikácii.

3.1 Výber technológií

Pri výbere technológií bolo potrebné zvážiť aj možnosti budúceho rozširovania funkcionality zamestnancami firmy *Siemens*. Je logické prispôsobiť technológie použité pre tento projekt technológiám, s ktorými majú ich vývojári skúsenosti. Konkrétne ide o jazyk *Java* a *Spring framework*.

Android

Užívateľské požiadavky jednoznačne definovali platformu, ktorá bude použitá pre vývoj mobilnej aplikácie. Existuje ale viacero možností v akom jazyku vyvíjať *android* aplikáciu. Oficiálnym jazykom pre vývoj na platforme *android* je *Java* [5] a ak vylúčime ostatné programovacie jazyky, ktoré nie sú natívne pre platformu *android* zostáva už len jazyk *Kotlin*, ktorý taktiež beží nad *JVM (Java Virtual Machine)*. Keďže *Kotlin* vznikol v roku 2011 odráža sa to na jednoduchšom zápise, bezpečnejšom zaobchádzaní s *null* hodnotami a menšom množstve štandardizovaného (*boilerplate*) kódu. Výhodou pri použití programovacieho jazyka *Kotlin* je, že je v projekte možné definovať aj *Java* triedy a použiť ich v *Kotlin* kóde [6]. Viac o platforme *android* sa dozviete v kapitole 3.2.1.

Programovací jazyk

Ako bolo v úvode tejto kapitoly spomenuté, programovací jazyk *Java* bol zvolený pre vývoj tejto aplikácie. Viac o programovacom jazyku *java* sa dozviete v kapitole 3.3.1.

Výber frameworku

Hlavným *frameworkom* z dôvodu údržby a ďalšieho vývoja dátovej služby je *Spring framework*, ktorý obsahuje funkcionalitu pre komunikáciu s databázou, vkladanie závislostí, vytváranie a zabezpečenie *webových* služieb. Vďaka jeho popularite má veľkú komunitu užívateľov, čo je výhodou pri riešení problémov počas vývoja.

3.2 Použité technológie

3.2.1 Android

Platforma *android* je najpoužívanejšou platformou pre mobilné zariadenia s vyše 2 miliardami aktívnych užívateľov mesačne [7]. Prvýkrát bola oficiálne predstavená v roku 2007 spoločnosťou *Google* a dnes už existuje v svojej 9. verzii. Popularita má vplyv aj na počet dostupných aplikácií. V súčasnosti (*október 2018*) si môžeme vybrať z 3.6 milióna aplikácií, ktoré sú dostupné pre platformu *android* [1]. Pre vývoj na tejto platforme je potrebný *Android Software Development Kit (SDK)*, čo je súbor nástrojov a *API* pre vývoj aplikácií.

3.2.2 Spring framework

Spring framework patrí k najpopulárnejším *open-source frameworkom* pre vývoj aplikácií na platforme *Java Enterprise Edition* [8]. Prvá verzia *frameworku* z roku 2002 bola publikovaná Rodom Johnsonom pod názvom *Expert One-on-One J2EE Design and Development* [9]. V súčasnej piatej verzii obsahuje množstvo koponentov pre uľahčenie vývoja aplikácií [10]. Medzi najkľúčovejšiu funkcionalitu patrí:

- Inversion of Control - vkladanie závislostí
- Aspektovo orientované programovanie (AOP)
- Dátový prístup - transakčný manažment, podpora *JDBC*, či objektovo-relačného mapovania
- Autentifikácia a autorizácia - konfigurovateľná a extendovateľná podpora
- Testovanie - podpora pre prípravu unit a integračných testov

3.2.3 Spring Boot

Spring Boot je riešenie *Spring frameworku* na vývoj samostatných (*standalone*) aplikácií podľa paradigmu *Convention-over-configuration*. *Spring Boot* je v svojej podstate taktiež *framework*, ktorý je ale dopredu predkonfigurovaný s odporúčanými konfiguráciami *Springu* a knihovňami tretích strán pre rýchlejšie použitie [11].

3.2.4 REST

Representational state transfer (REST) je súbor architektonických princípov prvýkrát definovaný v dizertačnej práci Roya Fieldinga z roku 2000 [12]. *REST-ové* služby umožňujú prístup a modifikáciu zdrojov webových služieb pomocou uniformného a preddefinovaného množstva bezstavových operácií. Práve bezstavovosť tohto architektonického štýlu zjednodušuje implementáciu *webových* služieb, zrýchľuje obsluhovanie jednotlivých požiadaviek a uvoľnovanie zdrojov [13]. *REST* používa pre identifikáciu jednotlivých zdrojov reťazec znakov inak známy ako *Uniform Resource identifier (URI)* [14].

3.2.5 Swagger

Spring framework umožňuje vytváranie *REST* koncových bodov, ktoré spracúvavajú požiadavky od klienta. Týmto spôsobom by bolo potrebné manuálne vytvoríť entity, kontroléry obsluhujúce koncové body a validáciu vstupných parametrov. Použitím *frameworku Swagger* je však možné zredukovať čas na vývoj a prípadné chyby. Tento *framework* pokrýva celý cyklus vývoja *API* od návrhu a implementácie až po dokumentáciu a testovanie. Umožňuje vytváranie entít a rozhrania kontrolérov, ošetruje nesprávne vstupy od klienta a generuje *HTML* dokumentáciu.

3.2.6 OAuth 2.0

OAuth 2.0 je autorizačný *framework* definovaný v dokumente *RFC 6749* [15], ktorý nahradzuje predchádzajúcu verziu *OAuth 1.x* a umožňuje užívateľom alebo tretej strane limitovaný prístup k *HTTP* službe alebo jej dátam bez nutnosti odhaľovať prihlasovacie údaje [16]. Takýto prístup je v *OAuth 2.0* umožnený pomocou *prístupových tokenov* (*Access*

Token) štandardne vo formáte *JSON Web Token (JWT)*, ktorý obsahuje informácie o získaných povoleniach jeho držiteľa. *OAuth 2.0* definuje nasledujúce 4 role [17]:

- *Majiteľ zdrojov (Resource Owner)* - entita vlastniaca údaje na serveri a rozhodujúca o pridelení úrovne práv
- *Server so zdrojmi (Resource server)* - server obsahujúci chránené údaje, schopný akceptovať a odpovedať na požiadavky o prístup pomocou *prístupových tokenov*
- *Klient (Client)* - aplikácia vytvárajúca požiadavky o prístup k chráneným údajom v mene *Majiteľa zdrojov*
- *Autorizačný server (Authorization server)* - server, ktorý autentifikuje *Majiteľa zdrojov* a vydáva *prístupové tokeny klientovi* po úspešnom prijatí autorizácie

3.3 Použité programovacie jazyky

Výsledkom kapitoly 3.1 je výber dvoch programovacích jazykov. Kde jazyk *Java* bude použitý pre development dátovej služby a jazyk *Kotlin* pre vývoj *android* aplikácie. Táto sekcia v krátkosti predstavuje tieto programovacie jazyky.

3.3.1 Java

Java je jeden z najznámejších a najpoužívanejších programovacích jazykov súčasnosti. Bol vytvorený v roku 1995 spoločnosťou *Sun Microsystems*. Ide o silne typovaný, objektovo-orientovaný programovací jazyk, ktorý môže byť kompilovaný do platformovo nezávislých *bytekód* inštrukcií a bežať na *Java Virtual Machine (JVM)* alebo byť kompilovaný priamo do strojového jazyka konkrétneho počítača. Vďaka tomu je možné použiť jazyk *Java* na akejkoľvek platforme, ktorá obsahuje *JVM* [18].

3.3.2 Kotlin

Kotlin je moderný, staticky typovaný programovací jazyk vyvíjaný firmou *JetBrains* a bol prvýkrát predstavený v roku 2011. Rovnako ako *Java* beží na *JVM* a môže byť taktiež kompilovaný do jazyka *JavaScript*. Od Októbra 2017 je *Kotlin* plne podporovaný spoločnosťou *Google* a označený za prvotriedny programovací jazyk pre platformu *android* [19]. Medzi jeho výhody oproti *Jave* patrí napríklad, bezpečnosť pri zaobchádzaní s *null* hodnotami, kratší zápis tried vďaka implicitným metódam *get*, *set* a 100% interoperabilita s existujúcim kódom v *Jave* a mnoho ďalších [20].

3.4 Architektúra

Vybrané technológie poskytujú dostatočný obraz pre návrh architektúry pre výsledný systém, kde budú komunikovať tri entity podľa schémy nižšie (obrázok 3.1). Prvou entitou je zariadenie s operačným systémom *android*, ktoré bude obsahovať nainštalovanú *Inventory Application* aplikáciu. Druhou entitou je webový server *Tomcat*, ktorý bude obsahovať nasadenú službu *Inventory Service*. Poslednou entitou je databázový server hostiaci *MySQL* databázový systém. Všetky tieto entity sa budú nachádzať v rámci rovnakej siete.

3.4.1 Komunikácia medzi android aplikáciou a dátovou službou

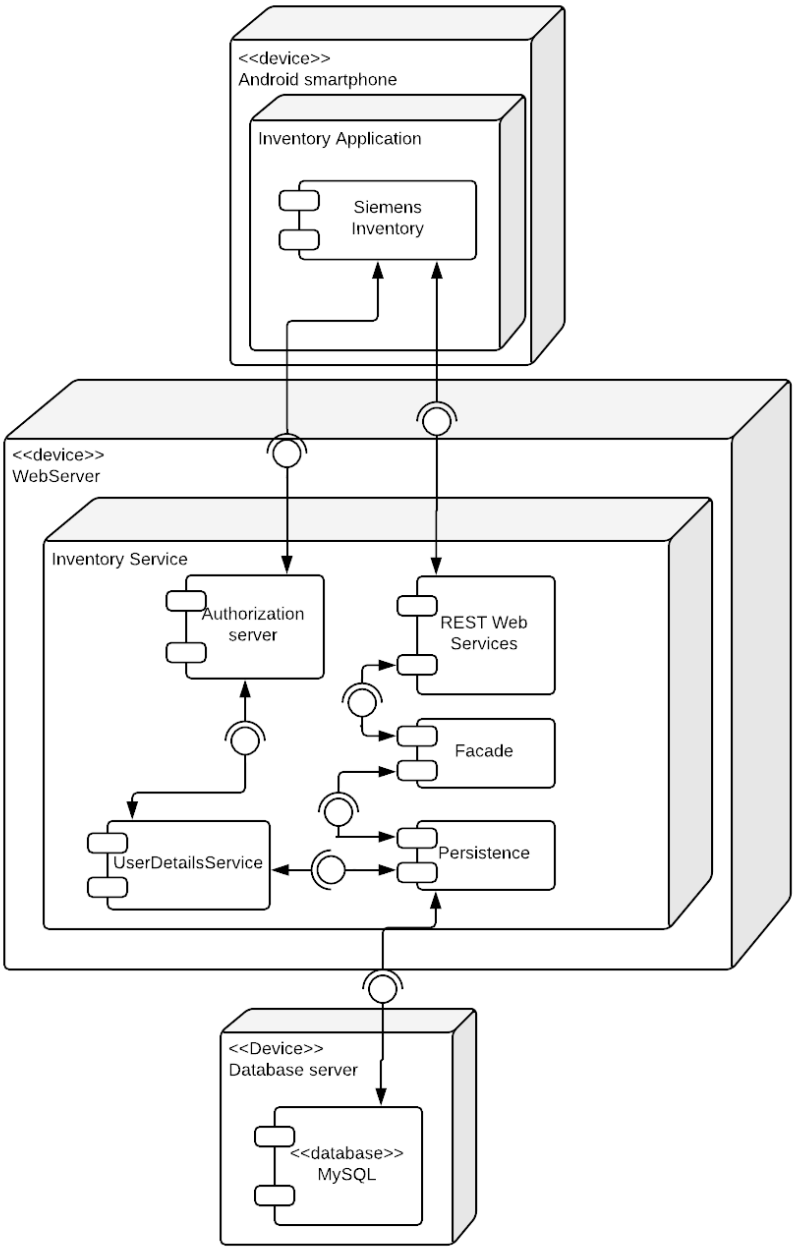
Komunikácia medzi týmito entitami bude prebiehať pomocou definovaného *REST* rozhrania, ktorého zverejnenie je zodpovednosťou dátovej služby. Táto služba bude prijímať, spracovávať a odpovedať na požiadavky vo formáte *application/json* vystavené klientom v prípade, že je klient autentizovaný a dostatočne autorizovaný. Spomínaná autorizácia a autentizácia bude prebiehať pomocou *OAuth 2.0*, kde klient (*android* aplikácia) požiada autorizačný server *Inventory Service* aplikácie o *prístupový token* správnymi prihlasovacími údajmi. Následne týmto *prístupovým tokenom* kontaktuje *REST*-ové rozhranie. Dátová služba pomocou *HTTP stavových kódov* a tela odpovede notifikuje o úspešnosti požiadavky.

3.4.2 Spracovanie požiadaviek dátovou službou

Služba je zodpovedná za spracovanie prichádzajúcich autorizovaných požiadaviek. Prvotné spracovanie začína v *REST kontroleroch* obsluhujúcich jednotlivé koncové body, kde sa skontroluje, či formát *JSON* požiadavky odpovedá definovanému konktraktu služby. Následne sa požiadavka predá *fasáde*, kde sa skontroluje sémantická korektnosť požiadavky a vykoná sa mapovanie z externého formátu dátových entít na interný a opačne. Na základe charakteristiky požiadavky je kontaktovaná *persistence* vrstva za účelom uloženia zmien, prípadne získania dát.

3.4.3 Komunikácia medzi dátovou službou a databázovým serverom

Persistence vrstva obsahuje objektové reprezentácie databázových tabuliek a objekty databázového prístupu (*Database Access Object - DAO*), ktoré sú zodpovedné za vykonávanie operácií nad databázou. Prepojenie s databázou je riešené v rámci *Spring frameworku* a jeho konfiguračného súbora viac v kapitole 4.1.3.



Obr. 3.1: Architektúra systému

3.5 Dátová služba

Hlavným účelom tejto *REST* služby bude poskytovanie dát pre *android aplikáciu*. Keďže existujúce riešenie vo firme *Siemens* nevystavuje žiadne použitelné knižnice, je potrebné vytvoriť jednotlivé vrstvy znova. Základným prvkom služby je *Spring framework*, ktorý je doplnený *frameworkom Swagger* a zabezpečenie služby je implementované použitím autorizačného *frameworku OAuth 2.0*. Projekt bol vytvorený v jazyku *Java* s buildovacím nástrojom *maven* a obsahuje 2 moduly, z ktorých boli vytvorené knižnice.

- *Persistence* - formát *jar*
- *REST* - formát *war*

3.5.1 Knižnica Persistence

Ide o *maven* modul aplikácie obsahujúci balíky *entity* a *dao*. Pre budúci vývoj bola táto vrstva oddelená do vlastnej *jar* knižnice, a tým bolo umožnené ju extendovať alebo využiť v inej aplikácii.

Entity

Balík entity obsahuje objektovú reprezentáciu databázových tabuliek potrebných pre samotnú aplikáciu podľa špecifikácie *JPA*. Tá bola pre čitateľnosť doplnená ešte *frameworkom Lombok*, ktorý použitím anotácií skracuje zápis jednotlivých funkcií a zvyšuje čitateľnosť. Definíciu entity je možné vidieť na príklade 3.

DAO

Tento balík využíva *framework Spring Data JPA* na definovanie *DAO* tried. Spomínaný *framework* vystavuje rôzne rozhrania a medzi nimi je potrebné spomenúť rozhranie *JpaRepository*, ktoré poskytuje základné aj rozšírené metódy na čítanie a zápis dát z databázy. Toto rozhranie je možné ďalej rozširovať o ďalšie metódy podľa biznis požiadaviek. Rozširovanie je možné implementovať rôznymi spôsobmi. V príklade nižšie sú uvedené 2 možnosti rozširovania *DAO* metód. Jednou z nich

je *Query Methods*, kde *Spring framework* vygeneruje *SQL query* na základe pomenovania *DAO* metódy (*getDeviceByBarcodeNumber* príklad 4). Druhým spôsobom je ručné definovanie *SQL query* pomocou anotácie *@Query* (*getDevicesBorrowedByUser* príklad 4).

Príklad 3 *Ukážka definície entity Supplier*

```
@Data
@NoArgsConstructor
@Entity
@Table(name = "suppliers")
public class Supplier {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  @Column(name = "name")
  private String name;
}
```

Kde anotácia:

- *@Data - vytvára potrebné getter, setter, toString, equals a hashCode metódy*
- *@NoArgsConstructor - vytvára bezparametrický konštruktor*
- *@Entity - špecifikuje, že trieda reprezentuje entitu*
- *@Table - špecifikuje databázovú tabuľku pre túto entitu*
- *@Id - špecifikuje, že premenná reprezentuje primárny kľúč entity*
- *@GeneratedValue - špecifikuje stratégiu generovania primárneho kľúča entity*
- *@Column - špecifikuje mapovanie premennej na stĺpec v tabuľke*

Príklad 4 *Ukážka definície triedy DeviceDao*

```
public interface DeviceDao
    extends JpaRepository<DeviceInternal, Long> {

  Optional<DeviceInternal>
      getDeviceByBarcodeNumber(String barcodeNumber);

  @Query("SELECT dev FROM DeviceInternal dev " +
         "WHERE dev.holder.id=:id")
  List<DeviceInternal>
      getDevicesBorrowedByUser(
        @Param("id") Long userId);
}
```

Kde anotácia:

- *@Query - definuje SQL query, ktoré sa aplikuje pri zavolaní metódy*
- *@Param - vloží hodnotu v parametri userId do query parametra :id*

3.5.2 Knižnica Rest

Táto knižnica využíva knižnicu *Persistence*, vytvára *REST* dátovú službu a balí ju do formátu *war*, ktorý slúži na nasadenie aplikácie na server *Tomcat* vo firme *Siemens*. Ide o *Spring Boot* aplikáciu, ktorá vystavuje *REST API endpointy* pre komunikáciu s konzumujúcim klientom.
Na dosiahnutie spomínanej funkcionality je potrebné vytvoriť triedy reprezentujúce entity, ktoré budeme zasielať a prijímať od klienta a kontroléry, ktoré budu spracovávať požiadavky od klienta. Táto komponenta je zodpovedná aj za zabezpečenie celej služby.

DTO entity a definícia API

DTO (Data Transfer Object) je typ objektu, ktorý zapuzdruje dáta vymienané medzi dvoma systémami [21]. Tieto objekty spolu s *API* definujú kontrakt pre klienta, ktorý chce využívať danú službu. Za týmto účelom bol použitý *framework Swagger* [22], ktorý umožňuje definíciu

dto entít a *API*. Ďalšou jeho výhodou je, že umožňuje vygenerovanie dokumentácie na základe týchto definícii a validuje prichádzajúce požiadavky podľa špecifikácie entít a *API*.

Príklad 5 *Ukážka definície entity Supplier a jej API (supplier.yaml)*

```
basePath: /inventory-service/api/v1
paths:
  /suppliers:
    get:
      summary: Gets all Suppliers
      operationId: getAllSuppliers
      produces:
        - application/json
      responses:
        '200':
          schema:
            type: array
            items:
              ref: '#/definitions/Supplier'
definitions:
  Supplier:
    type: object
    properties:
      id:
        type: integer
        format: int64
      name:
        type: string
    required:
      - name
```

Príklad 5 definuje entitu Supplier a jej *Http GET endpoint '/inventory-service/api/v1/suppliers'*, ktorý poskytuje list všetkých supplierov.
Z tejto definície vo formáte *yaml* sa pri kompilácií projektu vytvorí trieda *Supplier.java* a rozhranie *SuppliersApi.java*.

REST Kontroléry

Konkrétna implementácia *Api.java tried sa nachádza v balíku *controllers*. Každý kontrolér implementuje triedu **Api.java* vygenerovanú *frameworkom Swagger*. Pomocou *Spring Dependency Injection* sú inicializované potrebné fasády v kontroleri. Takáto *Dependency Injection* je vykonaná pomocou konštruktora a anotácie *@Autowired*. *Spring framework* na základe rozhrania nájde triedu, ktorá toto rozhranie implementuje, vytvorí jej inštanciu a vloží ju ako parameter do konštruktora. Triedu tak zbavuje zodpovednosti za vytváranie objektov (príklad 6).

Príklad 6 *Ukážka vkladania závislostí v Spring frameworku*

```
public class SupplierController
      implements SuppliersApi {

    private SupplierFacade supplierFacade;

    @Autowired
    public SupplierController(
            SupplierFacade supplierFacade) {
        this.supplierFacade = supplierFacade;
    }
}
```

Mapér

Vytvorenie *dto* a *JPA* entít môže vytvoriť určitú nekonzistenciu v prípade, že nechceme zverejniť určité parametre *JPA* entity. Klasickým príkladom je *JPA* entita *User*, ktorá by mohla obsahovať parametre ako napríklad *password*, ktoré nechceme zasielať mimo aplikácie. Prípadne by mohlo ísť o mapovanie medzi rôznymi formátmi dátumov. Triedy v balíku *mapper*, teda riešia mapovanie medzi *dto* a *JPA* entitami.

Príklad 7 *Ukážka mappera pre entitu Supplier*

```
@Service
public class SupplierMapperImpl
    implements SupplierMapper {

  @Override
  public Supplier mapToInternal(SupplierDTO object) {
    if (object == null) {
      return null;
    }
    Supplier supplier = new Supplier();
    supplier.setId(object.getId());
    supplier.setName(object.getName());
    return supplier;
}
```

Kde anotácia:

- *@Service - indikuje, že táto trieda môže byť použitá pre Dependency Injection*

Fasáda

Posledným významným prvkom pre manipuláciu s dátami sú fasády. Tie za pomoci *mapérov* slúžia ako adaptér medzi *REST API* a *dao* vrstvou, ďalej validujú a manipulujú s dátami. Fasáda taktiež vytvára a ukončuje transakcie v databáze. Pre svoju funkcionalitu potrebuje

dva komponenty *mapper* a *dao*, ktoré sú vložené pomocou závislostí do konštruktora.

Príklad 8 *Ukážka fasády pre entitu Supplier*

```
@Service
@Transactional
public class SupplierFacadeImpl
      implements SupplierFacade {

  private SupplierDao supplierDao;
  private SupplierMapper supplierMapper;

  @Autowired
  public SupplierFacadeImpl(SupplierDao supplierDao,
      SupplierMapper supplierMapper) {
    this.supplierDao = supplierDao;
    this.supplierMapper = supplierMapper;
  }

  @Override
  public List<SupplierDTO> getSuppliers() {
    return supplierMapper
      .mapToExternal(supplierDao.findAll());
  }
}
```

Kde anotácia:

- *@Transactional - vytvára databátovú transakciu pre každé zavolanie metódy*

Zabezpečenie

Zabezpečenie dát je v súčasnosti veľmi často riešený problém, kvôli ktorému vznikol aj nový právny rámec ochrany osobných údajov, inak známy pod skratkou *GDPR (General Data Protection Regulation)*. Toto

nariadenie nariadzuje zaviesť technické, organizačné a procesné opatrenia za účelom ochrany dát [23]. Preto nie je možné brať zabezpečenie dát na ľahkú váhu, aj keď ide o aplikáciu na interné použitie. Aplikácia definuje niekoľko rolí, ktoré užívateľ môže vlastniť. Ide o role:

- *Read* - sprístupňuje čítanie dát (mimo rolí Borrow, Inventory, Revision, Admin)
- *Write* - sprístupňuje zapisovanie a modifikáciu dát (mimo rolí Borrow, Inventory, Revision, Admin)
- *Borrow* - sprístupňuje čítanie a modifikáciu dát pre Borrow funcionalitu
- *Inventory* - sprístupňuje čítanie a modifikáciu dát pre Inventory funcionalitu
- *Revision* - sprístupňuje čítanie a modifikáciu dát pre Revision funcionalitu
- *Admin* - použité len v predchádzajúcej aplikácií, obsahuje všetky role

Pre výsledné zabezpečenie je potrebné použiť zabezpečovaciu techniku, ktorá podporuje ako autentifikáciu, tak autorizáciu. Takáto technika je *OAuth 2.0*. V rámci *Spring frameworku* existujú pripravené triedy na konfiguráciu autorizačného a resource servera pre správne fungovanie zabezpečenia. *OAuth 2.0* podporuje viac spôsobov prihlasovania. Kedže užívatelia sa budú prihlasovať pomocou mena a hesla, bola zvolená prihlasovacia technika (*grant type*) *password*. Pre kontaktovanie autentifikačného servera je potrebné, aby hlavička obsahovala aj *client-id* a *client-secret*. Sú to tajné hodnoty, ktoré klient pre kontaktovanie servera musí poznat. Ak autentifikácia prebehne v poriadku, klient získa *JWT (JSON Web Token)*, s ktorým sa pri ďalšej komunikácií so serverom autorizuje.

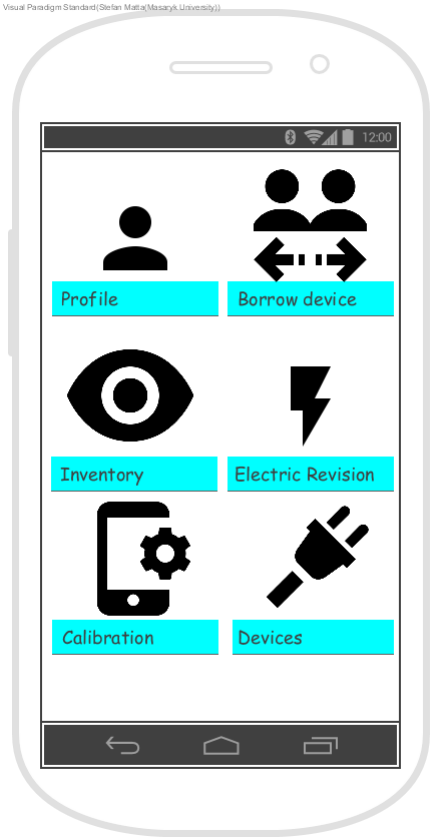
3.6 Mobilná aplikácia

3.6.1 Návrh grafického rozhrania

Pred začatím vývoja *android* aplikácie bolo potrebné vytvoriť návrh grafického rozhrania, s ktorým budú užívatelia interagovať. Ako nástroj na návrh grafického rozhrania som zvolil *Visual Paradigm 15.1*.

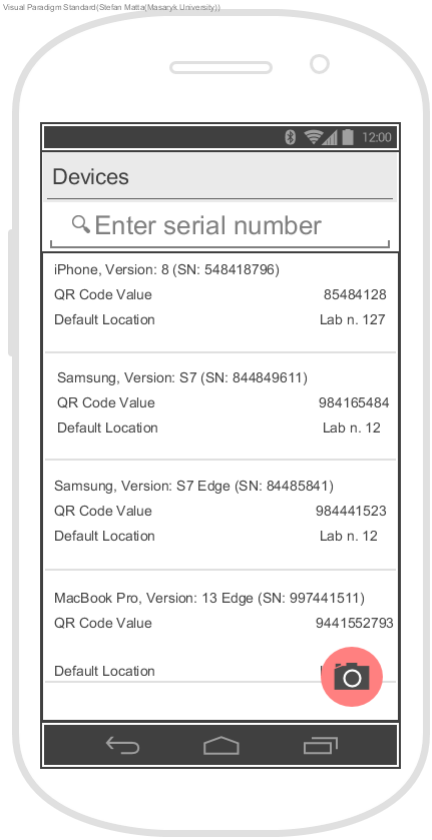
Prvou obrazovkou, s ktorou užívatelia prídu do kontaktu je prihlasovacia obrazovka (obrázok 3.2). Tá ma jednoduchý dizajn a obsahuje len formulár pre zadanie emailu a hesla. V neskoršej verzii bolo pridané tlačidlo na konfiguráciu pripojenia na dátovú službu. Prihlasovací formulár obsahuje aj validácie hodnôt v jednotlivých poliach. Po úspešnom prihlásení sa užívateľovi zobrazí menu aplikácie (obrázok 3.3). Menu obsahuje dlaždice, ktoré sa skladajú z grafickej a textovej časti. Obrázky a texty použité pre tieto dlažice bolo potrebné zvoliť tak, aby sa zhodovali s webovou verziou.

Obr. 3.2: Návrh prihlasovacej obrazovky



Obr. 3.3: Návrh menu aplikácie

Väčšina obrazoviek vo webovej verzii aplikácie pracuje so zoznamom zariadení. Preto bolo nutné zjednotiť dizajn zoznamov na rôznych aktivitách. Výsledkom je dizajn na obrázku 3.4, kde v hornej časti obrazovky sa nachádza názov aktivity, ktorú užívateľ práve vykonáva. Pod názvom je vyhľadávacie pole, kde užívateľ môže zadať hodnoty čiarového kódu, prípadne sériového čísla. Ďalej nasleduje samotný zoznam, ktorý zobrazuje informácie o zariadení dôležité pre danú aktivitu. Užívateľ má taktiež možnosť automatického skenovania čiarového kódu zariadenia po kliknutí na tlačidlo fotoaparátu v pravom dolnom rohu. Takáto predloha dizajnu bola použitá v aktivitách *Devices*, *Inventory*, *Electric Revision*, *Calibration* a *Device Types*.



Obr. 3.4: Návrh obrazovky zariadení

Obr. 3.5: Návrh obrazovky pridávania nového zariadenia

Častou aktivitou je vytváranie, prípadne editácia zariadenia. Tieto akcie sú dostupné z aktivít ako *Devices, Inventory, Electric Revision* a *Calibration*. Z hľadiska dizajnu je obrazovka rozdelená na pravú a ľavú časť, kde ľavá strana obsahuje názvy parametrov zariadenia a pravá strana textové polia a rozbaľovacie zoznamy možných hodnôt. V prípade hodnoty parametra *QR Code Value* má užívateľ možnosť aj automatického skenovania *QR* kódu.

3.6.2 Návrh štruktúry aplikácie

Pre dosiahnutie čitateľnosti kódu a projektu je potrebné organizovať zdrojový kód do balíkov [24]. Rozdelenie je možné vykonať rôznymi spôsobmi ako napríklad podľa kategórie alebo funkcionality. V prvom prípade sa zdrojový kód delí do balíkov ako *activities* - obsahuje zdrojové kódy, ktoré obsluhujú aktivity alebo *adapters* - obsahuje zdrojové kódy, ktoré obsluhujú zoznamy. V druhom prípade, je kód rozdelený podľa funkcionality, čo znamená, že je vytvorený balík podľa názvu aktivity a všetky zdrojové kódy súvisiace s danou aktivitou sa v ňom nachádzajú.

Prvotne som zvolil štrukturovanie kódu podľa kategórie, čo sa však v neskoršom štádiu vývoja ukázalo ako málo čitateľné z dôvodu, že balíky aj napriek rozdeleniu do kategórií obsahovali veľké množstvo tried. Preto som zvolil druhú možnosť - štrukturovanie podľa funkcionality. Príkladom takejto organizácie je balík *devicetype*. Ten obsahuje nasledovné súbory:

- DeviceTypeActivity.java

  Trieda obsluhujúca aktivitu zobrazenia, editácie a vytvárania entity *Device Type*

- DeviceTypeListAdapter.java

  Adaptér pre zoznam použitý v aktivite *Device Types*

- DeviceTypeServiceApi.java

  Trieda obsluhujúca komunikáciu s *REST* službou pre entitu *Device Type*

- DeviceTypeListActivity.java

  Trieda zobrazujúca zoznam objektov typu *Device Type*

3.6.3 Návrh komunikácie s dátovou službou

Pre kontaktovanie koncových bodov dátovej služby je dôležité vedieť typ objektu, ktorý požaduje prípadne vracia, *http* metódu a *url* adresu koncového bodu. Tieto informácie je vďaka knižnici *retrofit* možné reprezentovať objektovo. Táto knižnica potrebuje dva základné prvky, a to triedu reprezentujúcu dátovú entitu (napríklad trieda *Device.kt*) a rozhranie reprezentujúce koncový bod (napríklad rozhranie *DeviceTypeServiceApi.kt*).

Príklad 9 *Ukážka definície rozhrania DeviceTypeServiceApi.kt*

```
interface DeviceTypeServiceApi {

  @GET("device-types/")
  fun getDeviceTypes(): Call<List<DeviceType>>

  @GET("device-types/{devTypeId}")
  fun getDeviceType(@Path("devTypeId") deviceTypeId: Long)
      : Call<DeviceType>

  @POST("device-types/")
  fun createDeviceType(@Body device: DeviceType?)
      : Call<DeviceType>
}
```

Kde:

- *anotácie @GET a @POST - špecifikujú http metódu a url koncového bodu*
- *návratové typy funkcii reprezentujú návratové typy, ktoré dátová služba vracia*
- *parametre funkcii reprezentujú typy, ktorá dátová služba príjma buď v tele http požiadavky (anotácia @Body) alebo v url parametri (anotácia @Path)*

Volanie takejto funkcie s príslušným parametrom vytvorí *http* požiadavku na zvolenú dátovú službu. Spracovanie odpovede prebieha pomocou definovania tzv. funkcií spätného volania (*z angl. callback*). Pre spracovanie je nutné vytvoriť dve *callback* funkcie, a to funkciu *onResponse*, ktorá spracováva odpoveď servera a funkciu *onFailure*, ktorá je zavolaná v prípade vzniknutých problémov ako napríklad straty spojenia.

Keďže pre zabezpečenie dátovej služby bolo zvolené *OAuth*, tak pri prihlásení užívateľa je potrebné požiadať autorizačný server o *oauth token* správnymi prihlasovacími údajmi. Tento *token* sa následne zasiela serveru s každou ďalšou požiadavkou v jej *http* hlavičke. Knižnica *retrofit* to umožňuje za vytvorením vlastnej implementácie triedy typu *Interceptor*, ktorá pred zavolaním koncového bodu umožňuje vykonať naprogramovanú funkcionalitu ako napríklad v tomto prípade - vloženie *oauth tokenu* do hlavičky *http* požiadavky.

4 Implementácia

Implementácia prebiehala na základe návrhu z kapitoly 3. Keďže pre vývoj *android* aplikácie a jej testovanie bolo potrebné mať zdroj dát, implementácia dátovej služby bola uprednostnená. Ako nástroje pre vývoj boli zvolené *IntelliJ IDEA* - pre vývoj dátovej služby, *DBeaver* pre manipuláciu s dátami v databáze, *Postman* - pre vytváranie požiadaviek na službu a *Andriod studio* pre vývoj *android* aplikácie. Nasledujúce kapitoly popisujú implementáciu najdôležitejších komponentov.

4.1 Softvér dátovej služby

4.1.1 Implementácia služby

Prvotným krokom bola inštalácia lokálneho *MySQL* servera a vytvorenie databázových objektov pomocou *SQL skriptu*. Skript bol získaný exportovaním schémy z existujúceho databázového systému firmy *Siemens*. Nasledovalo vytvorenie *spring* projektu dátovej služby definovanej v návrhu pomocou nástroja *Spring Initializr* [25].

Prvým implementovaným modulom bol modul *persistence*. Triedy v jeho balíku nazvanom *entity* boli z väčšiny použité z existujúceho webového riešenia, avšak ich bolo potrebné značnou mierou refaktorovať kvôli nedostatkom spomenutým v kapitole 2.1.3. Pre tieto entity boli vytvorené rozhrania v balíku *dao*, ktoré poskytujú funkcie pre čítanie, mazanie a modifikovanie dát v databáze. Pre tento *maven* modul bola v súbore *pom.xml* nastavená hodnota parametra *packaging* na hodnotu *jar* pre vytvorenie znovupoužiteľnej knižnice.

Modul *rest* obsahuje okrem samotnej služby aj špecifikácie dátových entít a koncových bodov pre *framework swagger*. Na základe týchto špecifikácií sa pri kompilácii vygeneruje v balík *gen* obsahujúci triedy dátových entít a rozhraní *rest* kontrolérov, ktoré boli použité pri ich implementácii. Okrem zdrojového kódu sa v zložke *resources/static/documentation* vygeneruje dokumentácia k *rest* rozhraniu. Parameter *packaging* v súbore *pom.xml* bol v tomto prípade nastavený na hodnotu *war* z dôvodu, že tento artefakt sa bude nasadzovať na server *Tomcat*.

4.1.2 Implementácia *OAuth*

Spring framework pre jednoduchšiu implementáciu zabezpečenia poskytuje balíky ako napríklad *spring-security-core*. Vložením tohto balíka medzi závislosti v súbore *pom.xml spring framework* predvolene použije *Basic* autentifikáciu, toto nastavenie je však možné ďalej konfigurovať. Konfigurácia *oauth* zabezpečenia prebieha v triede *SecurityConfig.java*, kde je definovaný typ hašovacej funkcie, služba poskytujúca údaje o užívateľoch a ich právach - *UserDetailsService* a typ uloženia *jwt tokenov*.

Implementácia *UserDetailsService*

Spring framework nepozná vnútornú reprezentáciu užívateľov, ich prihlasovacích údajov a oprávnení, preto poskytuje rozhranie, ktorého implementácia sa použije pri prihlasovaní užívateľa. Týmto rozhraním je *UserDetailsService*, ktorého jedinou funkciou je *loadUserByUsername*.

```
UserDetails loadUserByUsername(String username)
                throws UsernameNotFoundException;
```

Úlohou implementácie je zistenie, či sa užívateľ nachádza v systéme a ak áno, vrátiť *UserDetails* objekt obsahujúci jeho prihlasovacie údaje a práva v systéme, ktoré sa porovnajú s hodnotami zadanými užívateľom. V prípade, že užívateľ neexistuje, je potrebné vyhodiť výnimku *UsernameNotFoundException*. Implementácia tejto triedy sa nachádza v *UserDetailsServiceImpl.java*.

Implementácia autorizačného servera

Pre správne fungovanie *oauth* autorizácie je dôležitý aj autorizačný server. *Spring framework* umožňuje pomocou rozhrania *AuthorizationServerConfigurerAdapter* nakonfigurovať vlastný autorizačný server, ktorého implementácia je v triede *AuthorizationServerConfig*. Pre server je potrebné inicializovať parametre ako *clientId, clientSecret, grantType a scopes*. Tieto hodnoty sú načítané z konfiguračného súbora. Bližší popis konfigurácie je v sekcii *OAuth* 4.1.3.

4.1.3 Konfigurácia

Spring framework poskytuje možnosť centrálnej konfigurácie pomocou *application.properties* súbora. Týmto spôsobom je možné konfigurovať nie len parametre *Spring frameworku*, ale aj iných komponentov ako napríklad *Hibernate, OAuth, či Logovania*.

Databázové pripojenie

Pre pripojenie k databáze je potrebné nastaviť hodnoty pre parametre *$url*, *$username* a *$password*.

```
spring.datasource.url=$url
spring.datasource.username=$username
spring.datasource.password=$password
```

OAuth

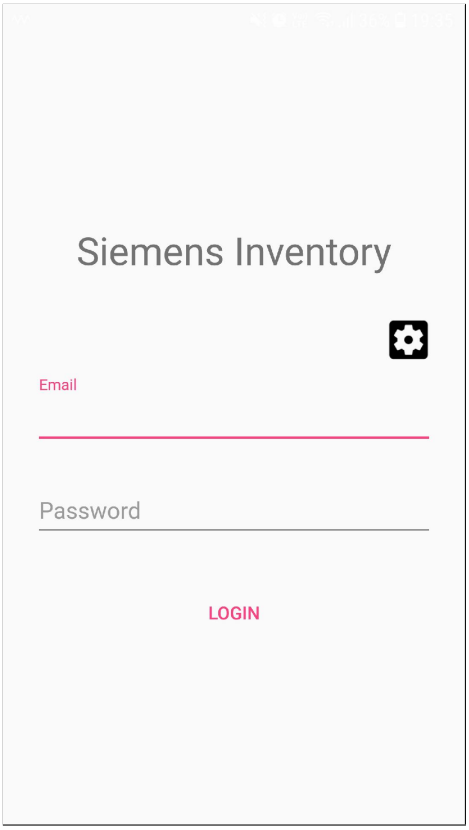
Konfigurácia *OAuth* obsahuje definície rolí a povinných parametrov ako *$security-realm*, *$signing-key*, *$client-id a $client-secret*.

```
security.security-realm=$security-realm
security.signing-key=$signing-key
security.jwt.client-id=$client-id
security.jwt.client-secret=$client-secret
security.jwt.grant-type=password
security.jwt.scope-read=read
security.jwt.scope-write=write
security.jwt.scope-admin=admin
security.jwt.scope-borrow=borrow
security.jwt.scope-revision=revision
security.jwt.scope-inventory=inventory
```

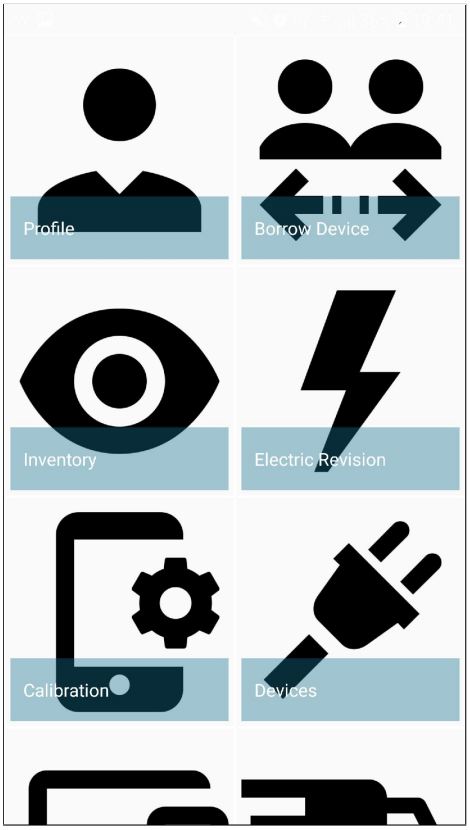
4.2 Softvér android aplikácie

4.2.1 Implementácia grafického rozhrania

Implementácia grafického rozhrania prebieha v *xml* súboroch, kde sa definujú a inicializujú atribúty použitých grafických prvkov. Tieto prvky sú naviazané na dátový model pomocou kontrolérov (aktivít) a spolu tak tvoria architektonický model zvaný *Model-View-Controller* skrátene *MVC* [26]. Obrázok 4.1 zobrazuje implementáciu obrazovky prihlásenia podľa návrhu z obrázka 3.2. Implementácia bola doplnená o tlačidlo nastavenia aplikácie. Na obrázku 4.2 je zobrazené rolovacie menu aplikácie podľa návrhu na obrázku 3.3.

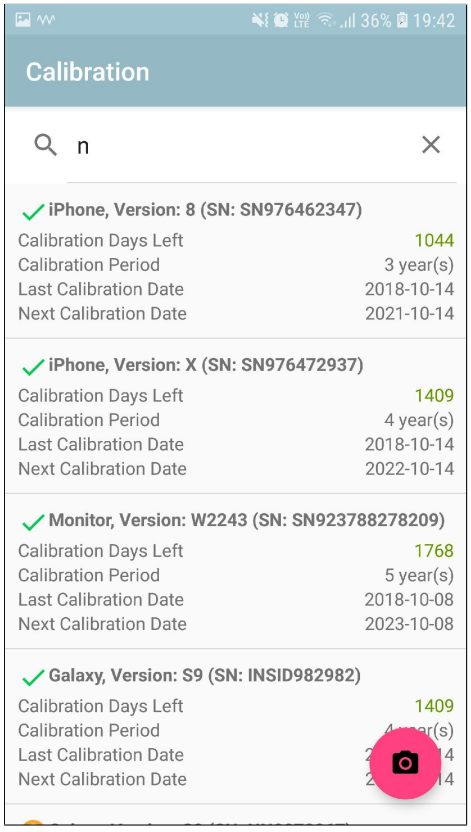


Obr. 4.1: Obrazovka prihlasovania

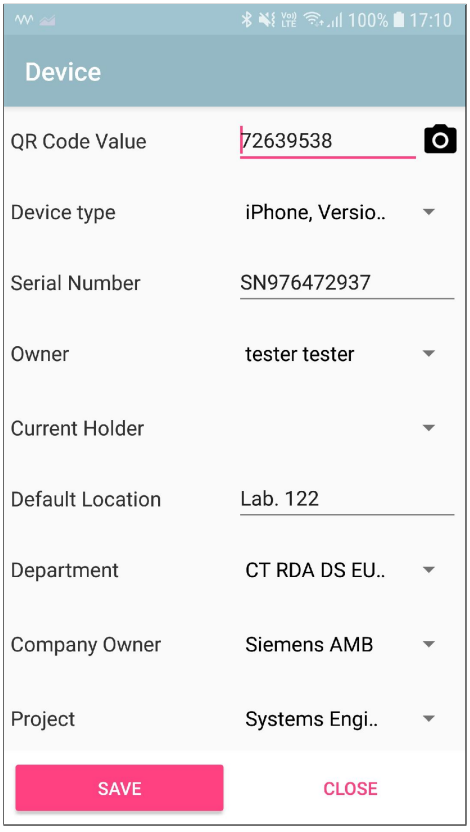


Obr. 4.2: Menu aplikácie

Na obrázku 4.3 je možné vidieť usporiadanie (*z angl. layout*) elementov pre filtrovanie listu. Rovnaký *layout* bol použitý v aktivitách ako Calibration, Electric Revision, Inventory a Devices. V hornej časti obrazovky sa nachádza vyhľadávacie pole, ktoré pri zmene hodnoty filtruje obsah listu a jeho položiek pod ním. V dolnej časti obrazovky sa nachádza tlačidlo na skenovanie čiarového kódu, ktoré umožňuje vyhľadať zariadenie v systéme bez potreby manuálneho zadávania hodnoty do vyhľadávacieho poľa. Táto obrazovka je implementáciou návrhu z obrázka 3.4. Na obrázku 4.4 je obrazovka zariadenia, ktorá je prístupná z viacerých aktivít v rôznych kontextoch (viac o kontextoch v podkapitole 4.2.2). Spodná časť obrazovky obsahuje tlačidlá relevantné pre daný kontext.



Obr. 4.3: Obrazovka kalibrácie



Obr. 4.4: Obrazovka zariadenia

4.2.2 Implementácia logiky aplikácie

Implementácia nastavení aplikácie

Predpokladá sa, že *android* aplikácia a dátová služba budú nasadené aj pre iné pobočky firmy *Siemens*. Z toho dôvodu je potrebné, aby parametre spojenia s dátovou službou boli konfigurovateľné. O túto konfiguráciu sa stará aktivita *SettingsActivity*, v ktorej užívateľ môže nastaviť parametre pripojenia k dátovej službe ako *url* a *oauth* parametre.

Implementácia skenovania čiarových kódov

Možnosť skenovania čiarových a QR kódov bola jedna z najhlavnejších funkčných požiadaviek pre výslednú aplikáciu. Táto funkcionalita je implementovaná za použitia *open-source* knižnice *ZXing* [27], ktorá slúži na spracovanie lineárnych alebo *2D* čiarových kódov z obrázkov. Aktivita zodpovedná za získanie hodnoty čiarového kódu je *ScanActivity*.

Implementácia aktivity Device

Aktivita *device* môže byť volaná z rôznych kontextov. Je to z dôvodu, že grafické rozhranie a funkcionalita pre obrazovku vytvárania, editácie, zobrazenia, inventarizácie, revízie a kalibrácie zariadenia sa z väčšiny zhoduje. Preto je zodpovedná za správnu inicializáciu grafického rozhrania pre žiadaný kontext. V tomto prípade bola použitá dvojsmerná dátová väzba (*z angl. two-way data binding*) [28], ktorá umožňuje prepojiť model s grafickým rozhraním. Týmto spôsobom sú zmeny na dátovom modeli automaticky reflektované na grafickom rozhraní a opačne.

4.2.3 Implementácia komunikácie s REST službou

Pre zjednodušenie implementácie bola použitá knižnica *retrofit*, ktorá na základe definície rozhrania umožňuje vytvárať *rest* volania. Pre vytvorenie týchto rozhraní je nutné vytvoriť dátové triedy reprezentujúce prenášané dáta. Tieto triedy sa nachádzajú v balíku *api.entity*.

Prvotnou akciou potrebnou pre správnu kominukáciu s dátovou službou je prihlásenie. Rozhranie implementujúce túto požiadavku je *LoginServiceApi*, ktoré definuje metódu *getNewAccessToken* takto:

Príklad 10 *Metóda zodpovedná za získanie oauth tokenu*

```
@FormUrlEncoded
@POST("/oauth/token")
fun getNewAccessToken(@Header("authorization") auth:String,
                      @Field("username") username:String,
                      @Field("password") password:String,
                      @Field("grant_type") type:String)
                      : Call<AccessToken>
```

Kde parameter:

- ***auth*** *obsahuje zašifrovanú hodnotu clientid a clientsecret pomocou Base64*
- ***username*** *obsahuje email zadaný užívateľom pri prihlásení*
- ***password*** *obsahuje heslo zadané užívateľom pri prihlásení*
- ***grant_type*** *predstavuje typ oauth grantu*

Volanie tejto metódy zašle *post* požiadavku na koncový bod dátovej služy obsluhujúci *oauth* autorizáciu. Po úspešnom prihlásení obsahuje návratová hodnota *oauth token*, ktorý je ďalej nutné zasielať s každou požiadavkou. Z tohto dôvodu bola vytvorená trieda *ServiceApiGenerator*. Tá pri vytváraní inštancie rozhrania pre *retrofit* definuje metódu, ktorá pred zaslaním požiadavky vloží do jej *http* hlavičky *oauth token* získaný pri prihlásení.

5 Testovanie

Testovanie je neoddeliteľnou súčasťou vývoja softvéru. Úlohou tohto procesu je overovanie kvality produktu, a preto by malo byť zaradené do *životného cyklu vývoja softvéru (z angl. Software Development Life Cycle)* [29]. Z hľadiska zamerania môžeme testovanie rozdeliť na funkčné a nefunkčné testovanie. Zatiaľ čo funkčné testovanie sa zameriava na overenie, že výsledný produkt splňuje funkčné požiadavky, cieľom nefunkčného testovania je overiť nefunkčné aspekty aplikácie ako bezpečnosť, výkonnosť alebo použiteľnosť [30][31]. Práve testovaním použiteľnosti výslednej *android* aplikácie sa zaoberá nasledujúca podkapitola.

5.1 Testovanie použiteľnosti

Testovanie použiteľnosti (*z angl. Usability testing*) je technika testovania softvéru určená na evaluáciu produktu zapojením zvyčajne malej skupiny koncových užívateľov. Cieľom tohto typu testovania je na základe užívateľských interakcií so systémom zistiť jednoduchosť ovládania systému, jednoduchosť pochopenia a zoznamovania sa so systémom a celková spokojnosť užívateľa [32].
Štrukturovaný prístup k takémuto typu testovania zahŕňa:

- Výber/identifikácia koncových užívateľov

  Cieľom tohto kroku je vybrať užívateľov, ktorí v súčasnosti používajú podobný produkt alebo budú testovaný produkt používať v budúcnosti.

- Výber úloh pre užívateľov

  Primárne je potrebné vybrať úlohy, ktoré by mohli odhaliť problémy spojené s použiteľnosťou systému, ďalej úlohy, ktoré reprezentujú reálne použitie aplikácie a úlohy vyplývyjúce z požiadaviek na produkt.

- Vykonanie testovania

  Počas samotného testovania užívatelia vykonávajú zadefinovné úlohy na testovanom systéme. Dôležitou súčasťou tohto

kroku je vytváranie poznámok napríklad o spôsobe, akým užívateľ interaguje so systémom alebo o type pomôcok, ktoré boli užívateľovi poskytnuté pre úspešné dokončenie úlohy. Tieto údaje neskôr pomôžu pri vykonávaní analýzy.

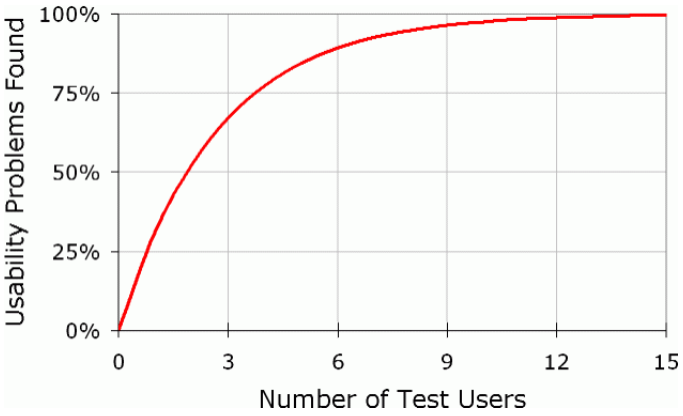
- Analýza výsledkov

  V tomto štádiu je potrebné posúdiť úspešnosť jednotlivých úloh, agregovať výsledky z testovania jednotlivých užívateľov a vyvodiť závery pre vylepšenie produktu.

V nasledujúcich podkapitolách predstavím konkrétne implementácie týchto krokov.

5.1.1 Výber koncových užívateľov

Výber užívateľov pre testovanie mobilnej aplikácie bolo na zástupcovi spoločnosti *Siemens*. Ten na základe zamerania aplikácie vybral troch užívateľov, ktorí mali možnosť s webovou verziou aplikácie pracovať už v minulosti alebo budú používať aplikáciu navrhnutou v rámci tejto diplomovej práce v budúcnosti. Štúdia firmy *Nielsen Norman Group* [33] sa zaoberala otázkou, koľko užívateľov je potrebných pre užívateľské testovanie. Na základe 83 projektov zistili, že 5 užívateľov odhalí až 85% chýb programu (graf 5.1), čo by pri množstve 3 užívateľov predstavovalo približne 66%.



Obr. 5.1: Percentuálne množstvo nájdených chýb na počet užívateľov [34]

5.1.2 Výber úloh pre užívateľov

Funkčné požiadavky na systém je možné rozdeliť do štyroch logických okruhov:

- Všeobecné operácie so systémom
- Požičiavanie zariadení
- Inventarizácia zariadení
- Vykonávanie revízie a kalibrácie

Tieto jednotlivé okruhy majú svoje špecifiká ako napríklad typ užívateľa, ktorý ich bude vykonávať, úrovne oprávnení takéhoto používateľa a tak ďalej. Úlohy obsahujú aj doplňujúce otázky, ktorých účelom je zistenie či užívateľ správne chápe údaje, ktoré sa mu zobrazujú na obrazovke. Každá sada úloh začína prihlasovaním do aplikácie a končí záverečnými otázkami. Tieto úlohy sa nachádzajú v prílohách B, C, D a E.

Všeobecné operácie so systémom

V tejto sade je od užívateľa vyžadované, aby pridal do systému nové položky ako napríklad nové oddelenie, typ zariadení alebo skontroloval svoje užívateľské práva. Táto sada je zameraná na funkčné požiadavky FR1, FR8 a FR9. Následne mu je predstavených šesť úloh, ktoré sú zamerané na:

- Úloha 1 - Prihlásenie sa do aplikácie
- Úloha 2 - Zistenie užívateľských práv a iných údajov
- Úloha 3 - Pridanie nového dodávateľa
- Úloha 4 - Pridanie, odoberanie a modifikácia oddelenia
- Úloha 5 - Pridanie nového typu zariadenia
- Úloha 6 - Odhlásenie sa z aplikácie

Požičiavanie zariadení

Táto sada úloh sa zameriava na funkcionalitu požičiavania a vrátenia požičaných zariadení definovanej v požiadavke FR4 a FR3. Užívateľ

je uvedený do situácie, kde je od neho potrebné, aby si vypožičal zariadenia. Následne sú mu predstavené štyri úlohy, ktoré sú zamerané na:

- Úloha 1 - Vypožičanie dostupného zariadenia
- Úloha 2 - Vypožičanie rezervovaného zariadenia
- Úloha 3 - Prezretie zoznamu práve vypožičaných zariadení
- Úloha 4 - Vrátenie požičaného zariadenia

Inventarizácia zariadení

Cieľom tejto sady úloh je overenie správnosti implementácie inventarizačnej funkcionality definovanej v požiadavke FR7 a funkcionality úpravy zariadenia FR2. Užívateľ je v úvode oboznámený so situáciou, ktorá ho informuje, že jeho úlohou je vykonať inventarizáciu pre štyri zariadenia. Následne sú mu predstavené štyri úlohy, ktoré sú zamerané na:

- Úloha 1 - Vykonanie inventarizácie
- Úloha 2 - Vykonanie inventarizácie pre neexistujúce zariadenie
- Úloha 3 - Vykonanie inventarizácie pre nové zariadenie
- Úloha 4 - Vykonanie inventarizácie a zmena parametrov zariadenia

Vykonávanie revízie a kalibrácie

Užívateľ je v tejto sade zodpovedný za vykonanie elektrickej revízie a kalibrácie zariadení. Cieľom je overiť funkčné požiadavky FR5 a FR6. Následne sú mu predstavené tri úlohy, ktoré sú zamerané na:

- Úloha 1 - Zaznamenanie elektrickej revízie zariadenia
- Úloha 2 - Zaznamenanie elektrickej revízie so zmenou parametrov
- Úloha 3 - Zaznamenanie kalibrácie zariadenia so zmenou parametrov

5.1.3 Vykonanie testovania

Testovanie prebiehalo v predom stanovenom termíne vo firme *Siemens* a bolo rozdelené do hodinových časových slotov pre každého užívateľa. V úvode bol každý užívateľ oboznámený s účelom tohto testovania a boli požiadaní o to, aby komentovali jednotlivé kroky, ktoré vykonávajú. Užívatelia dostali mobilné zariadenie, na ktorom bola nainštalovaná *android* aplikácia *Siemens Inventory*, ktorá bola predmetom testovania. Zároveň boli jednotlivé kroky užívateľa zaznamenané pomocou aplikácie *AZ Screen Recorder*, ktorá vyhotovila video záznam z testovania. Tento videozáznam bol vyhotovený so súhlasom užívateľa. Užívateľom boli priebežne kladené aj doplňujúce otázky, ktoré zisťovali či dostatočne rozumejú údajom zobrazeným v aplikácii. Taktiež záver každej sady úloh obsahoval otvorené otázky zamerané na hodnotenie aplikácie a funkcionality, ktorú práve používali.

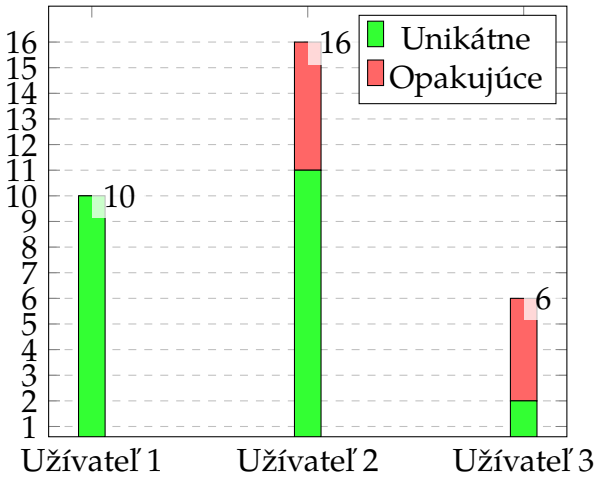
5.1.4 Analýza výsledkov

Analýza výsledkov zahŕňa agregáciu dát získaných počas vykonávania testovania. Medzi tieto dáta patria video nahrávky interakcie s mobilnou aplikáciou a poznámky vytvorené počas testovania. Obsahom poznámok boli komentáre od užívateľov k funkcionalite, ktorú práve používali, kdežto videonahrávky slúžili pre detailnejšiu analýzu problému.

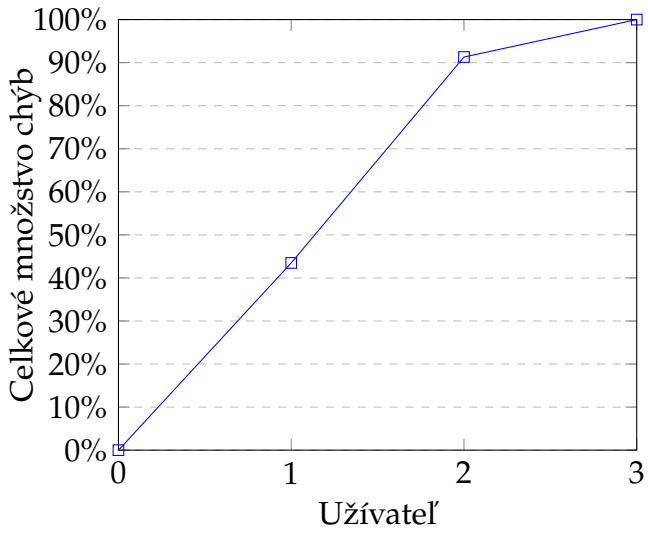
Najhlavnejším cieľom tohto testovania bolo zistiť chyby aplikácie v dizajne alebo funkcionalite aplikácie. Takto sa podarilo objaviť až 23 unikátnych chýb pomocou troch užívateľov. Celý zoznam nájdených chýb je v prílohe F. Obrázok 5.2 bližšie ukazuje pomer nájdených chýb medzi užívateľmi, kde zelená farba predstavuje novo objavené chyby užívateľom a červená farba zobrazuje chyby, ktoré boli identifikované predchádzajúcimi užívateľmi.

Ďalším cieľom testovania bol zber nových funkčných požiadaviek na prípadné budúce vylepšenie aplikácie. Celý zoznam požiadaviek definovaných užívateľmi je v prílohe G.

Zdrojom hodnôt pre nasledujúce grafy 5.2 a 5.3 je tabuľka F.1.

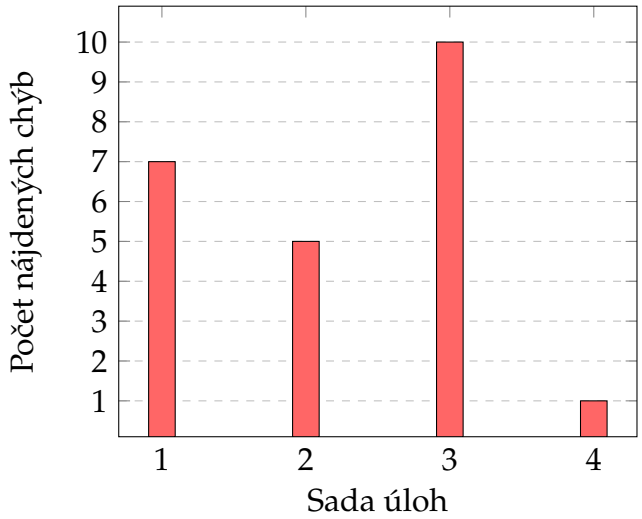


Obr. 5.2: Pomer nájdených chýb



Obr. 5.3: Percentuálne množstvo nájdených chýb

Z grafu je viditeľné, že zatiaľ čo prírastok prvých dvoch užívateľov predstavoval zhruba 43% a 48% pri treťom užívateľovi tento prírastok klesol už len na približne 9%.

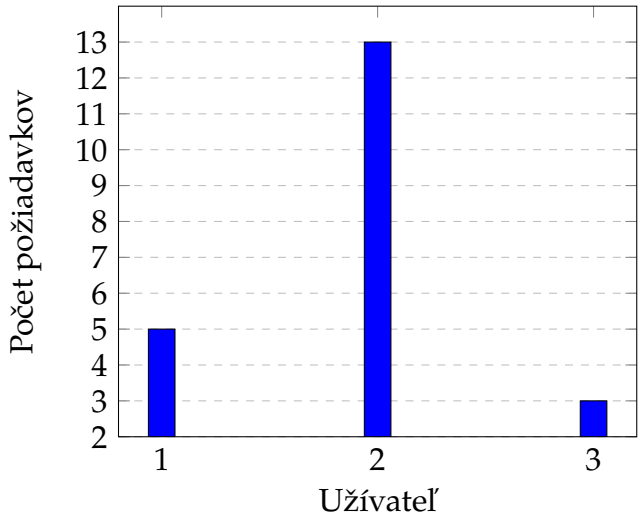


Obr. 5.4: Množstvo nájdených chýb na sadu úloh

V kapitole 5.1.2 bolo uvedené akým spôsobom je rozdelená funkcionalita aplikácie medzi 4 sady úloh. V grafe 5.4 je možné vidieť distribúciu chýb medzi týmito sadami. Sada s najväčším počtom chýb bola zameraná na inventarizáciu, kde prevládali chyby spôsobené nesprávnou terminológiou a chybami v dizajne, ktoré sťažovali prácu užívateľom. Sada s druhým najväčším počtom chýb bola zameraná na všeobecnú prácu s aplikáciou ako prihlasovanie a odhlasovanie, kontrola osobných údajov a vytváranie jednoduchých položiek. Zvláštnosťou je sada 4, ktorá podľa grafu 5.4 objavila iba 1 chybu, to je však spôsobené tým, že s kalibrácia, revízia a inventarizácia zdieľajú jednu obrazovku a chyby boli nájdené už v predchádzajúcej sade. Zdrojom hodnôt pre graf 5.4 je tabuľka F.2.

5.1.5 Nové užívateľské požiadavky

Účelom užívateľského testovania bolo aj zistenie nových funkčných požiadaviek na aplikáciu priamo od užívateľov. Každá sada úloh obsahovala v závere otázku, ktorej cieľom bolo zistiť, aká funkcionalita by im pri vykonávaní práce chýbala. Na základe toho, boli zadefinované nové užívateľské požiadavky. Jednotlivé počty požiadaviek medzi užívateľmi sú zobrazené v grafe 5.5. Tabuľka G.1 zobrazuje, ktoré požiadavky boli vyžiadané ktorým užívateľom.



Obr. 5.5: Funkčné požiadavky na užívateľa

Medzi požiadavky, ktoré boli žiadané aspoň dvoma užívateľmi patria:

NFR1 Prehľadávať a filtrovať zoznam zariadení pri vypožičiavaní

NFR3 Pridať údaje *Holder, Owner* a *Location* do položky listu zariadení

NFR6 Automatické otvorenie klávesnice pri vytváraní nových entít

NFR9 Zobraziť mnou vlastnené zariadenia

Kompletný zoznam požiadaviek je v prílohe G.

6 Nasadenie mobilnej aplikácie a služby

Pred samotným nasadením mobilnej aplikácie a dátovej služby je potrebné vytvoriť inštalačné súbory a artefakty. Výsledné artefakty sa nachádzajú v prílohe A.

Vytvorenie inštalačného súbora android aplikácie

Pomocou nástroja *Android Studio* je možné vytvoriť inštalačný *apk* súbor zo zdrojového kódu.

Vytvorenie nasaditeľného artefaktu dátovej služby

Artefakt dátovej služby vzniká počas kompilácie projektu a nachádza sa v zložke *rest/target* ako súbor *inventory-api-1.0.0.war*. Počas tejto kompilácie vzniká v zložke *persistence/target* knižnica *inventory-persistence-1.0.0.jar* použiteľná pre zaistenie integrácie s webovou verziou.

Inštalácia android aplikácie

Pre inštaláciu mobilnej aplikácie je potrebné uložiť *apk* súbor do mobilného telefónu a spustiť inštaláciu. Po úspešnej inštalácii je aplikácia dostupná pod názvom *Siemens Inventory*.

Nasadenie dátovej služby aplikácie

Nasadenie na server *tomcat* sa vykonáva umiestnením *war* artefaktu do zložky *webapps*, kde sa obsah súbora extrahuje a spustí sa *Inventory Service* aplikácia. Po extrahovaní do zložky *inventory-api-1.0.0* je nutné upraviť parametre súbora *WEB-INF/classes/application.properties* ako je uvedené v kapitole 4.1.3 a reštartovať server *tomcat*.

6.1 Nasadenie vo firme Siemens

Pre nasadenie vo firme *Siemens* boli vytvorené artefakty popísané vyššie. Celý proces nasadenia a konfigurácie bol vysvetlený a predvedený povereným zamestnancom firmy. V súčasnosti (*december 2018*) sú aplikácie vo firme *Siemens* nasadené a po opravení programových chýb

nájdených počas užívateľského testovania bude pridaná nová verzia mobilnej aplikácie a dátovej služby.

7 Záver

V tejto diplomovej práci som sa zaoberal analýzou, návrhom, implementáciou a testovaním mobilnej aplikácie pre platformu *android*. Aplikácia umožňuje vykonanie inventarizácie, požičiavanie a revíziu zariadení pre laboratórium spoločnosti *Siemens*. V druhej kapitole som sa zaoberal analýzou *backendu* súčasného webového riešenia a jeho znovupoužiteľnosťou pre mobilnú aplikáciu. Na základe analýzy boli vybrané vhodné technológie a bol vytvorený návrh konceptu mobilnej aplikácie a dátovej služby poskytujúcej dáta. Štvrtá kapitola sa zaoberá implementáciou navrhnutého konceptu mobilnej aplikácie, dátovej služby a ich vzájomnou komunikáciou. Výsledkom tejto časti je *android* aplikácia a dátová služba, ktoré je možné nasadiť vo firme *Siemens*. Pre možnosť budúceho rozšírenia a zaistenia kompatibility so súčasným webovým systémom bola vytvorená knižnica, ktorá sa dá využiť pre ďalšie projekty.

Nad rámec pôvodného zadania tejto diplomovej práce som pridal testovanie použiteľnosti mobilnej aplikácie na reálnych užívateľoch z firmy *Siemens*. Pre tento účel boli vypracované štyri sady úloh na overenie prívetivosti aplikácie. Dáta z testovania boli analyzované a agregované za účelom zistenia nedostatkov aplikácie. Výsledkom testovania bol zoznam programových chýb, ktoré je potrebné opraviť pre vylepšenie funkcionality, grafického rozhrania a prívetivosti aplikácie. Okrem zoznamu chýb boli vyzbierané aj nové užívateľské požiadavky na aplikáciu, ktoré si užívatelia sami vyžiadali.

Aplikácia a dátová služba boli priebežne nasadzované vo firme Siemens, aby sa ich funkčnosť overovala priamo v produkčnom prostredí. Avšak finálne nasadenie bude prebiehať až po opravení programových chýb nájdených počas užívateľského testovania.

7.1 Výhľad do budúcna

Námetom na budúcu prácu by bola generalizácia riešenia systému inventarizácie a systému výpožičky tak, aby bola použiteľná aj mimo firmy *Siemens*. Pre vylepšenie webovej a mobilnej aplikácie by bolo možné implementovať požiadavky získané počas užívateľského testovania aplikácie.

Použitá literatúra

[1] Statista GmbH. *Number of apps available in leading app stores as of 3rd quarter 2018*. 2018. URL: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/ (cit. 25. 10. 2018).

[2] Vaadin Ltd. *Vaadin Flow*. 2018. URL: https://vaadin.com/docs/v11/flow/Overview.html (cit. 28. 09. 2018).

[3] Jan Mráček. *Výpůjční systém s podporou inventarizace*. 2017. URL: https://dspace.cvut.cz/handle/10467/69687 (cit. 28. 09. 2018).

[4] Ph.D. doc. Ing. RNDr. Barbora Bühnová. *REQUIREMENTS SPECIFICATION*. 2018. URL: https://is.muni.cz/auth/el/1433/podzim2018/PB007/um/lec/02-RequirementsSpecification.pdf (cit. 08. 12. 2018).

[5] Adam Sinicki. *I want to develop Android Apps — What languages should I learn?* 2017. URL: https://www.androidauthority.com/develop-android-apps-languages-learn-391008/ (cit. 23. 10. 2018).

[6] Janice J. Heiss. *The Advent of Kotlin: A Conversation with JetBrains' Andrey Breslav*. 2013. URL: https://www.oracle.com/technetwork/articles/java/breslav-1932170.html (cit. 23. 10. 2018).

[7] Ben Popper. *Google announces over 2 billion monthly active devices on Android*. 2017. URL: https://www.theverge.com/2017/5/17/15654454/android-reaches-2-billion-monthly-active-users (cit. 25. 10. 2018).

[8] Vince Power. *Most Popular Java Web Frameworks*. 2018. URL: https://dzone.com/articles/most-popular-java-web-frameworks (cit. 25. 10. 2018).

[9] Rod Johnson. *Expert One-on-One J2EE Design and Development*. Wrox, 2002.

[10] Inc. Pivotal Software. *Spring Framework Documentation*. 2018. URL: https://docs.spring.io/spring/docs/current/spring-framework-reference/ (cit. 25. 10. 2018).

[11] Inc. Pivotal Software. *Spring Boot*. 2018. URL: https://spring.io/projects/spring-boot (cit. 25. 10. 2018).

[12] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Wrox, 2000.

[13] Jiří KADLEC. „REST a webové služby v jazyce JAVA“. Diplomová práce. Masarykova univerzita, Fakulta informatiky, Brno, 2010 [cit. 2018-10-27]. URL: https://is.muni.cz/th/y3qa7/.

[14] RESTfulAPI.net. *What is REST*. 2018. URL: https://restfulapi.net/ (cit. 25. 10. 2018).

[15] Ed. D. Hardt. *The OAuth 2.0 Authorization Framework*. 2012. URL: https://tools.ietf.org/html/rfc6749 (cit. 25. 10. 2018).

[16] Aaron Parecki. *OAuth 2.0*. 2018. URL: https://oauth.net/2/ (cit. 25. 10. 2018).

[17] Peter Špireng. *Ako dôverovať cudzím alebo protokol OAuth 2.0*. 2014. URL: https://robime.it/ako-doverovat-cudzim-alebo-protokol-oauth-2-0/ (cit. 25. 10. 2018).

[18] Andrew Binstock. *Java's 20 Years Of Innovation*. 2015. URL: https://www.forbes.com/sites/oracle/2015/05/20/javas-20-years-of-innovation/ (cit. 25. 10. 2018).

[19] Frederic Lardinois. *Google makes Kotlin a first-class language for writing Android apps*. 2017. URL: https://techcrunch.com/2017/05/17/google-makes-kotlin-a-first-class-language-for-writing-android-apps/ (cit. 25. 10. 2018).

[20] JetBrains s.r.o. *Learn Kotlin*. 2018. URL: https://kotlinlang.org/docs/reference/ (cit. 25. 10. 2018).

[21] Martin Fowler. *Data Transfer Object*. 2018. URL: https://martinfowler.com/eaaCatalog/dataTransferObject.html (cit. 26. 10. 2018).

[22] SmartBear Software. *Swagger*. 2018. URL: https://swagger.io/ (cit. 26. 10. 2018).

[23] Eva Škorničková. *Co je GDPR?* 2018. URL: https://www.gdpr.cz/gdpr/ (cit. 26. 10. 2018).

[24] CodePath. *Organizing your Source Files*. 2018. URL: https://guides.codepath.com/android/Organizing-your-Source-Files (cit. 19. 11. 2018).

[25] Inc. Pivotal Software. *SPRING INITIALIZR bootstrap your application now*. 2018. URL: https://start.spring.io/ (cit. 21. 11. 2018).

[26] Florina Muntenescu. *Android Architecture Patterns Part 1: Model-View-Controller*. 2016. URL: https://medium.com/upday-devs/android-architecture-patterns-part-1-model-view-controller-3baecef5f2b6 (cit. 01. 12. 2018).

[27] Inc. Denso Wave. *ZXing ("Zebra Crossing") barcode scanning library for Java, Android*. 2018. URL: https://github.com/zxing/zxing (cit. 04. 12. 2018).

[28] Google LLC. *Two-way data binding*. 2018. URL: https://developer.android.com/topic/libraries/data-binding/two-way (cit. 04. 12. 2018).

[29] Guru99. *SDLC (Software Development Life Cycle) Tutorial: What is, Phases, Model*. 2018. URL: https://www.guru99.com/software-development-life-cycle-tutorial.html (cit. 01. 12. 2018).

[30] Guru99. *What is Non-functional Testing?* 2018. URL: https://www.guru99.com/non-functional-testing.html (cit. 03. 12. 2018).

[31] Guru99. *FUNCTIONAL Testing Tutorial: What is, Process, Types, Examples*. 2018. URL: https://www.guru99.com/functional-testing.html (cit. 03. 12. 2018).

[32] Joseph S. Dumas a Janice C. Redish. *A Practical Guide to Usability Testing*. Intellect Ltd, 1999.

[33] Jakob Nielsen. *How Many Test Users in a Usability Study?* 2012. URL: https://www.nngroup.com/articles/how-many-test-users/ (cit. 12. 11. 2018).

[34] Jakob Nielsen. *Why You Only Need to Test with 5 Users*. 2000. URL: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/ (cit. 12. 11. 2018).

A Elektronické prílohy

Súčasťou tejto diplomovej práce sú aj nižšie uvedené prílohy.

zdrojovy_kod_datova_sluzba.zip - Zdrojový kód dátovej služby
inventory-api-1.0.0.war - War súbor dátovej služby
inventory-persistence-1.0.0.jar - Knižnica persistence
zdrojovy_kod_android.zip - Zdrojový kód *android* aplikácie
app.apk - Inštalačný súbor *android* aplikácie

B 1. Sada úloh - Všeobecné operácie so systémom

Úvod do situácie

Vaše meno je **Tory** a pracujete pre firmu *Siemens*. Vaša firma prešla istou formou reštrukturalizácie a tak Vám pribudli nové oddelenia, dodávatelia a zariadenia. Váš nadriadený Vás požiadal o pridanie týchto nových subjektov do systému aplikácie *Siemens Inventory*.

1. Úloha

Prihláste sa do aplikácie pomocou Vašeho emailu a hesla:
email: **tory@siemens.com**
heslo: **tory123**

Čo vidíte na obrazovke?

K čomu podľa Vás slúžia jednotlivé dlaždice?

---

2. Úloha

Zistite, aké práva v tomto systéme máte a kto je Váš nadriadený.

Ktoré typy oprávnení máte?

---

3. Úloha

Pridajte do systému nového dodávateľa "*IT-4-U*".

Ktorý dodávatelia sú v systéme?

4. Úloha

Pridajte do systému nové oddelenia:

- "AI"
- "Support"

Zmažte zo systému staré oddelenie "*Assembly*".
Premenujte oddelenie "*Research*" na oddelenie "*R&D*".

---

5. Úloha

Pridajte do systému nový typ zariadenia s hodnotami:
Názov: **Arduino**
Verzia: **UNO R3**
Klasifikácia: **1**
Výrobca: **SparkFun**
Číslo objednávky: **96816**
Dodávateľ: **IT-4-U**
Cena: **411**

---

6. Úloha

Odhláste sa z aplikácie.

Záverečné otázky

- Aký bol Váš dojem z funkcionality, ktorú ste používali?
- Čo sa Vám páčilo alebo nepáčilo?
- Aká funkcionalita by Vám chýbala pri vykonávaní tejto práce?
- Aká funkcionalita by Vám nechýbala pri vykonávaní tejto práce?

- Čo by ste vylepšili na prototype?

C 2. Sada úloh - Požičiavanie zariadení

Úvod do situácie

Vaše meno je **Eddie** a pracujete pre firmu *Siemens*. Kvôli Vaším pracovným povinnostiam si potrebujete vypožičať z laboratória 2 zariadenia.

1. Úloha

Prihláste sa do aplikácie pomocou Vašeho emailu a hesla:
email: **eddie@siemens.com**
heslo: **eddie123**

---

2. Úloha

Požičajte si zariadenie č.1.

Zariadenie č.1

Obr. C.1: QR kód zariadenia č.1

Akými spôsobmi je možné vyhľadať zariadenie?

Ktorý spôsob vyhľadávania by ste ešte očakávali?

3. Úloha

Požičajte si zariadenie č.2.

Zariadenie č.2

Obr. C.2: QR kód zariadenia č.2

Aké informácie ste zistili?

---

4. Úloha

Prezrite si zoznam práve požičaných zariadení.

Sú pre Vás tieto informácie dostačujúce?

Ako zistíte ďalšie informácie o tomto zariadení?

---

5. Úloha

Zariadenie z úlohy C už ďalej nepotrebujete. Vráťte toto zariadenie pomocou aplikácie.

Záverečné otázky

- Aký bol Váš dojem z funkcionality, ktorú ste používali?
- Čo sa Vám páčilo alebo nepáčilo?

- Aká funkcionalita by Vám chýbala pri vykonávaní tejto práce?
- Aká funkcionalita by Vám nechýbala pri vykonávaní tejto práce?
- Čo by ste vylepšili na prototype?

D 3. Sada úloh - Inventarizácia zariadení

Úvod do situácie

Vaše meno je **Alex** a pracujete pre firmu *Siemens*. Váš nadriadený Vás poveril vykonaním inventúry vo Vašom laboratóriu pomocou *Siemens Inventory* aplikácie na mobilnom zariadení, ktoré Vám poskytol. Vaše laboratórium celkovo obsahuje 4 zariadenia pre ktoré je nutné vykonať inventarizáciu a zadať do aplikácie výsledný stav. K dispozícii máte sériové číslo, QR kód a stav, ktorý je potrebné zadať do systému.

1. Úloha

Prihláste sa do aplikácie pomocou Vašeho emailu a hesla:
email: **alex@siemens.com**
heslo: **alex123**

---

2. Úloha

Vykonajte inventarizáciu pre zariadenie č.1.

Zariadenie č.1

Sériové číslo: **SN1598236**
Číslo QR kódu: **536001113266**

Obr. D.1: QR kód zariadenia č.1

Stav zariadenia

Toto zariadenie ste fyzicky skontrolovali, jeho údaje sú správne a teda prešlo inventarizáciou.

Akými spôsobmi je možné vyhľadať zariadenie?

Ktorý spôsob vyhľadávania je pre Vás najprívetivejší?

---

3. Úloha

Vykonajte inventarizáciu pre zariadenie č.2.

Zariadenie č.2

Sériové číslo: **SN9703601**
Číslo QR kódu: **5366953286**

Obr. D.2: QR kód zariadenia č.2

Stav zariadenia

Toto zariadenie ste vo Vašom laboratóriu nenašli a neviete ani kde by sa mohlo nachádzať. Zaznačte tento výsledok.

---

4. Úloha

Vykonajte inventarizáciu pre zariadenie č.3.

Zariadenie č.3

Zistite, či toto zariadenie je evidované v systéme.

Stav zariadenia

Pridajte toto zariadenie do systému, ak viete nasledujúce parametre zariadenia:

Obr. D.3: QR kód zariadenia č.3

Sériové číslo: **SN9703602**
Typ zariadenia: **iPhone, Version: X**
Dátum pridania: **dnešný dátum**
QR kód: **obrázok E.3**

Skontrolujte, či sa už zariadenie nachádza v systéme a zaznačte výsledok inventarizácie.

5. Úloha

Vykonajte inventarizáciu pre zariadenie č.4.

Zariadenie č.4

Sériové číslo: **SN5428036**

Stav zariadenia

Toto zariadenie ste fyzicky skontrolovali, no všimli ste si, že nemá priradený QR kód a taktiež typ zariadenia má byť **iPhone, Version: X** a nie **iPhone, Version: 8**. Opravte chybné údaje, priraďte zariadeniu QR kód z obrázka D.4 a zaznačte výsledok inventarizácie.

Obr. D.4: Nový QR kód zariadenia č.4

Záverečné otázky

- Aký bol Váš dojem z funkcionality, ktorú ste používali?

- Čo sa Vám páčilo alebo nepáčilo?
- Aká funkcionalita by Vám chýbala pri vykonávaní tejto práce?
- Aká funkcionalita by Vám nechýbala pri vykonávaní tejto práce?
- Čo by ste vylepšili na prototype?

E 4. Sada úloh - Vykonávanie revízie a kalibrácie

Úvod do situácie

Vaše meno je **Jamie** a pracujete ako externista vykonávajúci revízie a kalibrácie elektronických zariadení pre firmu *Siemens*. Firma *Siemens* Vás požiadala o vykonanie revízie a kalibrácie pre zariadenia s blížiacim sa termínom inšpekcie.

1. Úloha

Prihláste sa do aplikácie pomocou Vašeho emailu a hesla:
email: **jamie@revtech.com**
heslo: **jamie123**

---

2. Úloha

Vykonajte revíziu zariadenia č.1.

Zariadenie č.1

Sériové číslo: **SN1598236**
Číslo QR kódu: **536001113266**

Obr. E.1: QR kód zariadenia č.1

Stav zariadenia

Toto zariadenie ste fyzicky skontrolovali, je v poriadku a prešlo revíziou. Zaznačte výsledok.

Ktorý spôsob vyhľadávania je pre Vás najprívetivejší?

---

3. Úloha

Vykonajte revíziu zariadenia č.2.

Zariadenie č.2

Sériové číslo: **SN9703601**
Číslo QR kódu: **5366953286**

Obr. E.2: QR kód zariadenia č.2

Stav zariadenia

Toto zariadenie ste fyzicky skontrolovali, prešlo revíziou, no je potrebné zmeniť periódu revízii na 1 rok. Vykonajte potrebné zmeny.

4. Úloha

Zariadenie č.3 bolo potrebné skalibrovať na správne hodnoty. Zaznamenajte kalibráciu zariadenia č.3 do systému.

Zariadenie č.3

Sériové číslo: **SN9561785**
Číslo QR kódu: **5368703626**

Obr. E.3: QR kód zariadenia č.3

Stav zariadenia

Toto zariadenie ste skalibrovali, no je potrebné zmeniť periódu kalibrácii na 1 rok. Vykonajte potrebné zmeny.

Záverečné otázky

- Aký bol Váš dojem z funkcionality, ktorú ste používali?
- Čo sa Vám páčilo alebo nepáčilo?
- Aká funkcionalita by Vám chýbala pri vykonávaní tejto práce?
- Aká funkcionalita by Vám nechýbala pri vykonávaní tejto práce?
- Čo by ste vylepšili na prototype?
- Odporúčali by ste aplikáciu inému kolegovi/kolegyni?

F Zoznam odhalených chýb

B1 Prihlasovanie rozlišuje veľkosť písmen v emaili

B2 Nesprávna notifikácia pri mazaní entity - “Entity was removed“

B3 Nefunkčné rolovanie pri vytváraní *Device type*

B4 Otočenie displeja pri vytváraní *Device type* zmaže hodnoty

B5 Zlý názov aktivity “Borrow device“ namiesto “My borrowed devices“

B6 Zoznam na výber užívateľov je pridlhý

B7 Nejasné pomenovanie hodnôt revízie "false", ünclear", ök"

B8 Chýbajúce označenie povinných polí pri revízii

B9 Povolená modifikácia poľa owner pri revízii a kalibrácii

B10 Nesprávna notifikácia pri zadaní zlých prihlasovacích údajov

B11 Pomenovanie práva “inventory-making“

B12 Novo vytvorený *Device type* nie je v zozname

B13 Pomenovanie tlačidla "close"v dialógu *Borrow*

B14 Krátke zobrazenie notifikácie “Device is borrowed by other user“

B15 Pomenovanie poľa *add date*

B16 Zlý názov aktivity “Device“ namiesto “inventory“

B17 Pole *Inventory result* je predvyplnené na predchádzajúcu hodnotu

B18 Nesprávna notifikácia pri nenájdení zariadenia

B19 Zlý názov aktivity pri vytváraní zariadenia

B20 Otvorená klávenica pri inventarizácii

B21 Premenovanie tlačidla save na submit pri inventarizacii, revízii a kalibrácii

B22 Pole *Add date* nie je vyplnené pri vytváraní zariadenia

B23 Pole *Device type* je nečitateľné pri vytváraní zariadenia

Chyba	Užívateľ 1	Užívateľ 2	Užívateľ 3
B1	✓	✗	✗
B2	✓	✗	✓
B3	✓	✓	✓
B4	✓	✗	✗
B5	✓	✓	✗
B6	✓	✗	✗
B7	✓	✓	✗
B8	✓	✓	✗
B9	✓	✗	✗
B10	✓	✓	✗
B11	✗	✓	✗
B12	✗	✓	✗
B13	✗	✓	✗
B14	✗	✓	✗
B15	✗	✓	✗
B16	✗	✓	✗
B17	✗	✓	✗
B18	✗	✓	✗
B19	✗	✓	✓
B20	✗	✓	✓
B21	✗	✓	✗
B22	✗	✗	✓
B23	✗	✗	✓
(✓) objavil (✗) neobjavil			

Tabuľka F.1: Chyby objavené užívateľmi

Chyba	Sada 1	Sada 2	Sada 3	Sada 4
B1	✓	✗	✗	✗
B2	✓	✗	✗	✗
B3	✓	✗	✗	✗
B4	✓	✗	✗	✗
B5	✗	✓	✗	✗
B6	✗	✓	✗	✗
B7	✗	✗	✓	✗
B8	✗	✗	✓	✗
B9	✗	✗	✓	✗
B10	✓	✗	✗	✗
B11	✓	✗	✗	✗
B12	✓	✗	✗	✗
B13	✗	✓	✗	✗
B14	✗	✓	✗	✗
B15	✗	✓	✗	✗
B16	✗	✗	✓	✗
B17	✗	✗	✓	✗
B18	✗	✗	✓	✗
B19	✗	✗	✓	✗
B20	✗	✗	✓	✗
B21	✗	✗	✗	✓
B22	✗	✗	✓	✗
B23	✗	✗	✓	✗
(✓) objavená (✗) neobjavená				

Tabuľka F.2: Chyby odhalené v sadách úloh

G Zoznam nových užívateľských požiadaviek

NFR1 Prehľadávať a filtrovať zoznam zariadení pri vypožičiavaní
NFR2 Odstrániť údaj *Department* z položky listu zariadení
NFR3 Pridať údaje *Holder, Owner* a *Location* do položky listu zariadení
NFR4 Pridať možnosť filtrovania užívateľov pri výbere zo zoznamu
NFR5 Označenie povinných polí
NFR6 Automatické otvorenie klávesnice pri vytváraní nových entít
NFR7 Možnosť nápovedy a vysvetliviek
NFR8 Upozornenie na čas vrátenia zariadenia
NFR9 Zobraziť mnou vlastnené zariadenia
NFR10 Editovanie zariadenia z obrazovky *My borrowed devices*
NFR11 Zobraziť históriu pôžičiek
NFR12 Zobrazenie užívateľského *id* pre prípad duplicity mien
NFR13 Zvýraznenie sekcie Inventarizácie, Revízie a Kalibrácie
NFR14 Zmenšenie dialógu pre výber periódy
NFR15 Kontextové menu na rýchlejší výber akcii
NFR16 Pokročilé možnosti filtrovania nad zoznamom

Požiadavka	Užívateľ 1	Užívateľ 2	Užívateľ 3
NFR1	✓	✓	✓
NFR2	✓	✗	✗
NFR3	✓	✗	✓
NFR4	✓	✗	✗
NFR5	✗	✓	✗
NFR6	✗	✓	✓
NFR7	✗	✓	✗
NFR8	✗	✓	✗
NFR9	✓	✓	✗
NFR10	✗	✓	✗
NFR11	✗	✓	✗
NFR12	✗	✓	✗
NFR13	✗	✓	✗
NFR14	✗	✓	✗
NFR15	✗	✓	✗
NFR16	✗	✓	✗
(✓) požadoval (✗) nepožadoval			

Tabuľka G.1: Funkčné požiadavky žiadané užívateľmi