VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SERVEROVÝ FRAMEWORK PRO HROMADNÉ TESTOVÁNÍ ANDROID APLIKACÍ

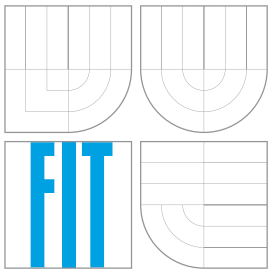
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. DUŠAN ČTVRTNÍČEK

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SERVEROVÝ FRAMEWORK PRO HROMADNÉ TESTOVÁNÍ ANDROID APLIKACÍ

SERVER FRAMEWORK FOR BATCH ANDROID APPLICATIONS TESTING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. DUŠAN ČTVRTNÍČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ALEŠ LÁNÍK

BRNO 2015

Abstrakt

Tato diplomová práce se věnuje problematice vzdáleného ovládání Android zařízení pomocí Android Debug Bridge nástroje. Další podstatná část této práce je automatizované hromadné testování Android aplikací. Zaměřuje se na jednotlivé nástroje (Android Debug Bridge, MonkeyRunner, logcat, Appium) potřebné k vyřešení této problematiky. Dále popisuje webový framework Nette a použité technologie, které byly využity pro výsledný návrh a implementaci informačního systému.

Abstract

This master's thesis is dedicated to the remote control of Android devices using the Android Debug bridge tool. Another essential part of the work is automated batch testing of Android applications. It focuses on individual tools (Android Debug Bridge, MonkeyRunner, logcat, Appium) needed to resolve this issue. It also describes a web framework Nette and other technologies that were used for the final design and implementation of information system.

Klíčová slova

Android, mobilní zařízení, Android Debug Bridge, MonkeyRunner, logcat, front-end, back-end, Nette Framework, Webové sokety, Appium, automatizované testování

Keywords

Android, mobile devices, Android Debug Bridge, MonkeyRunner, logcat, front-end, back-end, Nette Framework, Web sockets, Appium, automated testing

Citace

Dušan Čtvrtníček: Serverový framework pro hromadné testování Android aplikací, diplomová práce, Brno, FIT VUT v Brně, 2015

Serverový framework pro hromadné testování Android aplikací

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Aleše Láníka.

.....
Dušan Čtvrtníček
25. května 2015

Poděkování

Rád bych poděkoval svému vedoucímu diplomové práce Ing. Aleši Láníkovi za ochotu, čas a cenné rady, které mi poskytl během psaní této práce.

Obsah

Kapitola 1

Úvod

S rapidně narůstajícím počtem uživatelů, kteří používají Android zařízení, vzniká potřeba vyvíjet nové aplikace, které by splnily uživatelovi požadavky. Kvůli tomuto trendu trh s Android aplikacemi enormně narůstá. Bohužel ne každý Android vývojář testuje svoji aplikaci dostatečně předtím, než ji uvede na trh. Dnešní trh disponuje obrovským množstvím různých zařízení, což znamená, že rozhodně nestačí testovat Android aplikaci na jednom zařízení a předpokládat, že bude aplikace funkční na jiných zařízeních.

Cílem této práce je minimalizovat problém nedostatku dostupných zařízení k testování a ulehčit Android vývojáři práci. Práce se zabývá vzdáleným ovládáním Android zařízení, díky kterému je možné poskytnout Android vývojáři kontrolu nad testováním svojí aplikace na různých zařízeních. Jedná se o zařízení, které uživatel fyzicky nevlastní, ale jsou připojeny ke vzdálenému serveru. Tím by se měl minimalizovat problém, pokud by neměl vývojář fyzicky k dispozici žádné zařízení. Také to řeší problém s různorodostí zařízení. Pokud bude mít k dispozici více modelů mobilních zařízení na vzdáleném serveru, může svoji aplikaci lépe otestovat, popřípadě opravit.

Další možné vylepšení je automatizované hromadné testování vytvořené aplikace, které by mělo vývojáři ušetřit čas a aplikaci účiněji otestovat. Tato problematika bude další podstatnou částí práce. Vývojář si vytvoří vlastní testovací skript, podle kterého se bude testovat jeho aplikace. Po skončení testu vývojář obdrží záznam daného testu a informaci o tom jestli daný test proběhl úspěšně či nikoliv.

Samotný text této práce je rozdělen do několika kapitol. Kapitola 2 popisuje rozdělení webového vývoje na front-end a back-end. V rámci back-endové části je popsán Nette Framework, který byl použit při tvorbě výsledného systému. Dále jsou uvedeny prostředky k přenosu dat. Kapitola 3 obsahuje úvod do platformy Android a především popis potřebných nástrojů k vzdálenému ovládaní zařízení. Kapitola je zakončena automatizovaným testováním Android aplikací. Následující kapitola 4 analyzuje dostupná řešení a popisuje vlastní návrh vytvořeného systému. Návrh systému je doprovázen vývojovými diagramy. Další kapitola 5 se zabývá implementaci navrženého systému. Práci uzavírá kapitola 6, která se věnuje zhodnocením dosažených výsledků včetně vlastního přínosu a nastínění budoucího vývoje.

Tato práce plynule navazuje na semestrální projekt. V rámci semestrálního projektu jsem se seznámil s technologiemi, které byly potřebné k vyřešení vzdáleného ovládání Android zařízení a automatizovaného hromadného testování Android aplikací. Na základě těchto technologií byl proveden důkladný návrh výsledného informačního systému. Ze semestrálního projektu byly převzaty a upraveny kapitoly 2, 3 a 4.

Kapitola 2

Analýza použitých technologií

Cílem diplomové práce je vytvořit informační systém, který zprostředkovává vzdálené ovládání Android zařízení a hromadné automatizované testování aplikací skrze webový prohlížeč. Proto se tato kapitola zabývá technologiemi webového vývoje. První část se věnuje rozdělení webové aplikace na front-end a back-end vývoj. Poté je objasněná komunikace mezi jednotlivými částmi systému. Bližší specifikace požadavků na výsledný systém jsou popsány v sekci 4.2.

2.1 Front-end

V dnešní době je front-end vývoj chápaný jako část webové aplikace, se kterou uživatel interaguje. Mezi tři hlavní programovací jazyky front-end vývoje patří HTML[1], CSS[2] a JavaScript, které budou popsány níže v kapitole.

2.1.1 HTML

HTML je nejpoužívanějším značkovacím jazykem pro webové stránky. Jedná se o nejznámější aplikaci univerzálního jazyka SGML[3], kde je jednotlivým značkám jazyka přiřazena sémantika hypertextového dokumentu. První veřejná verze s označením HTML 0.9 byla vydána koncem roku 1991 a jejími autoři byli Tim Berners-Lee a Daniel Connolly. Tato verze ještě nepodporovala grafické rozhraní. Teprve ve verzi 2.0 přibyla podpora grafického rozhraní ve formě obrázků prostřednictvím značky <img>. V této práci byla použita verze HTML5.

HTML je charakteristické množinou značek a atributů, které dávají význam jednotlivým částem dokumentu. Značky v jazyce HTML mohou být párové či nepárové. Párové značky mění obsah dokumentu nacházejícího se mezi počáteční a koncovou značkou, a to podle významu dané značky. Nepárové značky nejsou uzavřeny a mají význam pouze ve specifickém místě dokumentů. Jako příklad nepárové značky je možné uvést zalomení textu pomocí značky
. Typickým zástupcem párové značky je .

Jak jsem již zmínil, v práci byla použita verze HTML5. Pro tuto verzi jsou typické některé změny, jako například deklarace dokumentu, nové značky pro vkládání multimediálního obsahu nebo značka <canvas>, která byla použita pro práci s obrázky a vektorovou grafikou.

[1] HyperText Markup Language.
[2] Cascading Style Sheets.
[3] Standard Generalized Markup Language.

Dokumenty v HTML mají předepsanou strukturu, která vypadá následovně:

- Nejprve je nutné deklarovat HTML verzi dokumentu. Jelikož je v práci použita verze HTML5, deklarace má zkráceny tvar <!DOCTYPE html>.
- Následuje párová značka <html>, která se nazývá kořenový element a reprezentuje celý dokument.
- Uvnitř elementu <html> se vyskytuje hlavička dokumentu v párové značce <head>, která obsahuje metadata.
- Po párové značce <head> následuje párová značka <body>, která obsahuje tělo dokumentu.

Samotné HTML slouží pro strukturování výsledného dokumentu. Pro výsledný celek front-end části je zapotřebí dalších jazyků, a sice CSS a JavaScript. Více o HTML v knize [20].

2.1.2 CSS

CSS je jazyk pro popis způsobu zobrazení jednotlivých elementů, které jsou napsány v značkovacím jazyce. V rámci této práce je CSS aplikováno na značkovací jazyk HTML. CSS bylo navrhnuto především z důvodu oddělení obsahu a vzhledu jednotlivých prvků dokumentu. Pro výběr jednotlivých elementů se používají tzv. selektory. Tyto selektory mohou být aplikovaný přímo na elementy v daném jazyce (v HTML např. element <h2> pro vytvoření hlavičky druhé úrovně) nebo na elementy specifikované atributem (id, třída). Také je možné vybrat element podle jeho umístění vůči DOM[4] modelu. Cascading Style Sheets, lze přeložit do českého jazyka jako kaskádové styly. Kaskádové, protože se na sebe mohou vrstvit definice různých stylů, ale platí jenom ten s největší prioritou. Na trhu je aktuální verze 3. Tato verze podporuje práci s animacemi, přidává nové textové efekty, podporuje více možností s ohraničením prvků a jiné. Jednotlivé priority v CSS jsou znázorněny v tabulce 2.1. Více o CSS v knize [20].

Priorita	**Název pravidla**	**Popis**
1	Důležitost	Přidáním deklarace **!important** k CSS stylu.
2	HTML styl	Přidáním **style** atributu v HTML elementu.
3	Média	Uvedení konkrétního CSS média.
4	Uživatelsky definované	Uživatelský definované CSS.
5	Specifičnost výběru	Výběr CSS podle specifičnosti.
6	Pořadí	Poslední pravidlo má vyšší prioritu.
7	Rodičovská dědičnost	Pokud není pravidlo definováno, nastává dědičnost.
8	CSS vlastnost	CSS pravidlo přepíše výchozí hodnotu prohlížeče.
9	Výchozí hodnoty	Výchozí hodnota je určena podle W3C specifikace.

Tabulka 2.1: Schéma CSS priorit.

[4] Document Object Model.

2.1.3 JavaScript

JavaScript je multiplatformní objektově orientovaný skriptovací jazyk, jehož autorem je Brendan Eich z tehdejší společnosti Netscape, který jej navrhl v roce 1995. Poté byl v červenci 1997 standardizován asociací ECMA[5] a v srpnu 1998 organizaci ISO[6]. Standardizovaná verze JavaScriptu je pojmenována jako ECMAScript a z ní byly odvozeny i další implementace, jako je například ActionScript. Přestože by se to dle názvu mohlo zdát, nemá nic společného s Javou. Java v názvu je pouze marketingový tah. S programovacím jazykem Java je podobná pouze syntaxe.

JavaScript běží na klientovi a je tak přímo závislý na použitém webovém prohlížeči. Kvůli tomuto faktu se dlouhou dobu nepoužíval, protože interpretace určitých konstrukcí byly rozdílné. S nástupem JavaScriptových frameworků se však začal prosazovat a využívat pro komplexnější aplikace. JavaScript je možné použít i na straně serveru. První implementací JavaScriptu na straně serveru byl LiveWire firmy Netscape vypuštěný roku 1996. Dnes existuje několik možností včetně open-source implementace Rhinola založené na Rhino, gcj, Node.js a Apache. Více o JavaScriptu v knize [20].

V rámci této práce bude použit framework jQuery z důvodu sjednocení chování webových prohlížečů, zjednodušení, zrychlení a zefektivnění práce. JQuery bylo ve své první verzi vydáno v roce 2006 Johnem Reesigem. Jedná se o open-source software pod duální licencí MIT[7] a GPL[8]. Usnadňuje přístup k elementům v DOM, změnu DOM elementů, manipulaci s CSS a nabízí různé efekty a animace. Další výhodou je, že má dobře propracovanou dokumentaci, díky které se lze snadno naučit využívat tento framework. Více o jQuery na webové stránce [14].

Další použitý framework jQuery UI[9] je javascriptový framework zaměřený na uživatelské rozhraní, který má za cíl ulehčit vývojářům implementaci pokročilých elementů a efektů ve webových stránkách. První verze byla vytvořena v roce 2007. JQuery UI je stejně jako ostatní součásti jQuery projektu vyvíjeno pod duální licencí MIT a GPL. JQuery UI se dělí na čtyři základní části:

- Interakce - Tato část obsahuje metody pro implementaci pokročilých interakcí mezi uživatelem a rozhraním webové aplikace.
- Widgety - V této části lze najít pokročilé elementy jako modální a nemodální dialogy, tlačítka, indikátor průběhu[10].
- Efekty - Zahrnuje efektové a barevné animace.
- Nástroje - Obsahuje nástroje pro úpravu pozic pokročilých elementů.

Více o jQuery UI je možné se dozvědět v knize [22]. V práci byly použity ještě další frameworky, jako například Highcharts k tvorbě interaktivních grafů a dále významný front-end framework Bootstrap. Bootstrap je open-source nástroj pod licencí MIT pro tvorbu webových aplikací. Obsahuje šablony pro typografii, formuláře, tlačítka, navigace a další komponenty. Bootstrap také obsahuje volitelné rozšíření JavaScriptu. Jedním z hlavních

[5] Europen Computer Manufacturers Association.
[6] International Organization for Standardization.
[7] Massachusetts Institute of Technology.
[8] General Public License.
[9] User interface.
[10] Progress bar.

cílů Bootstrapu je usnadnit a sjednotit vývoj webových aplikací, což jednoznačně splňuje. Více o Bootstrapu na webové stránce [4].

2.2 Back-end

Back-endová část se stará o chod aplikace. Tato část je uživatelovi v podstatě nepřístupná, protože komunikuje pouze s front-endovou částí, která zachycuje uživatelovi požadavky. Tyto požadavky jsou poté zpracovány aplikační logikou serveru. V nejjednodušších případech se jedná o databázové operace. Výsledná aplikace bude ovšem podstatně složitější z hlediska komunikace mezi jednotlivými částmi back-endu a front-endu. Komunikaci mezi jednotlivými částmi serveru se zabývá sekce 4.5. Co se týče programovacích jazyků pro back-endovou část jsme v podstatě limitování pouze vzdáleným serverem, kde výsledná aplikace běží. Mezi nejznámější patří PHP[11], Python, ASP.NET, Ruby, Java a Javascript (Node JS). Pro tuto práci byly vybrány back-endové technologie PHP a Python. Komunikace mezi serverovými částmi v Pythonu a PHP byla zajištěna pomocí IPC[12] (podrobněji v sekci 2.3.5).

2.2.1 PHP

PHP je skriptovací jazyk pro tvorbu dynamického webu a jeho počátky spadají do roku 1994. Tehdy se Rasmus Lerdorf rozhodl vytvořit jednoduchý systém pro počítání přístupu ke svým stránkám. Původní záměr byl napsán v jazyce PERL. Za nějakou dobu byl systém přepsán do jazyka C. Sada těchto skriptů byla ještě, později téhož roku, vydána pod názvem Personal Home Page Tools, zkráceně PHP. Rasmus Lerdorf veřejně vydal PHP v roce 1995 jako verzi 2, která již měla základní funkčnost jako má dnešní PHP. Koncem roku 1998 byla již k dispozici verze PHP 3.0. Vznik verze 4 přinesl do jazyka mnoho nových funkcí a rovněž přináší přepracované a tudíž podstatně rychlejší jádro Zend. V roce 2008 vzniká verze PHP 5, která přináší podporu objektově orientovaného programování.

Jedná se o dynamicky typovaný a interpretovaný jazyk, což znamená, že datový typ není určen při vytváření proměnné, ale až při jejím naplněním konkretní hodnotou. V sekci 2.3.3 bude podrobněji popsán protokol HTTP[13]. Tento protokol je bezstavový, což znamená, že neumí uchovávat stav komunikace. V PHP je možné uchovávat stav informace v tzv. session nebo cookies. Více o PHP v knize [24].

Velkou výhodou hovořící pro použití PHP v této práci je početné zastoupení knihoven a frameworků usnadňující tvorbu webové aplikace. Tyto frameworky usnadňují práci a zautomatizovávají tvorbu běžných rutin, jako je připojení k databázi, vytváření a validace formulářů, autorizace a autentizace uživatele. Implementují různé druhy filtrů a validátorů pro uživatelská data, jazykové komponenty, oddělují aplikační a prezentační logiku aplikace a mnoho dalšího. Existuje již celá řada frameworků pro PHP. Mezi nejznámější a nejvíce používané patří CakePHP, Zend Framework, Symfony a Nette Framework. V této práci byl použít Nette Framework, a proto zastává samostatnou sekci 2.2.2.

[11] PHP: Hypertext Preprocessor.
[12] Inter-Process Communication.
[13] Hypertext Transfer Protocol.

CakePHP

CakePHP je framework pro tvorbu webových aplikací napsaný v jazyce PHP. Tento framework je založen na softwarové architektuře MVC[14] a vznikl v roce 2005. CakePHP verze 1.0 byla vydána v květnu roku 2006 a v současně době je k dispozici verze 3.

CakePHP se vyznačuje srozumitelnou strukturou, díky čemuž je orientace v kódu snadná a vývoj aplikací rychlý. Framework lze propojit s nejpoužívanějšími databázemi (MySQL, PostgreSQL, SQLite). Má k dispozici velké množství nástrojů použitelných pro validaci vstupních dat, ověřování uživatelů apod. K dispozici je také rozsáhlá dokumentace, kde je možné nalézt i hotové ukázky aplikací. Více o frameworku na webové stránce [5].

Zend Framework

Zend Framework je open source, objektově orientovaný, webový aplikační framework implementovaný v PHP 5 a licencovaný pod New BSD[15] license. Tento framework je založen na softwarové architektuře MVC. Vývoj Zend Frameworku byl zahájen v roce 2005. Současná verze k dispozici je 2.4.0.

Zend Framework podporuje několik databázových systémů, jako jsou například MySQL, PostgreSQL, SQLite. Dále jsou k dispozici tzv. helpery, které programování webových stránek ještě více usnadňují. Helpery jsou dvojího typu, a to helpery akcí využívané v řadičích a helpery využívané v pohledech. Více o Zend Frameworku na webové stránce [29].

Symfony

Symfony je framework pro tvorbu webových aplikací v jazyce PHP postavený na softwarové architektuře MVC. Původně byl představen agenturou Sensio Labs, která jej využívala pro tvorbu webových stránek pro jejich klienty. Framework byl publikován v roce 2005 a dnes patří mezi nejpopulárnějších na světě. Používá ho například internetový portál Yahoo nebo systém Drupal. Aktuální verze k dispozici je 2.6

Tento framework používá konfigurační soubory, které se píšou v úsporném a čitelném syntaktickém zápisu YAML[16]. Je to velmi produktivní, deklarativní metoda programování. Dále nabízí automatické generování administrace pro aplikace. Více o frameworku Symfony na webové stránce [23].

2.2.2 Nette Framework

Nette Framework [12] je open source framework pro tvorbu webových aplikací v PHP 5. Jeho původním autorem je David Grudl, o jeho vývoj se nyní stará organizace Nette Foundation. Jedna z výhod tohoto frameworku je, že se zaměřuje na eliminaci bezpečnostních rizik (Cross Site Scripting[17], Cross-Site Request Forgery[18], atd.). Dále Podporuje AJAX[19] (podrobněji na konci této sekce), Dependency Injection (podrobněji níže v této sekci), také

[14] Model-view-controller.
[15] Berkeley Software Distribution.
[16] YAML Ain't Markup Language.
[17] Je metoda narušení WWW stránek využitím bezpečnostních chyb ve skriptech (především neošetřené vstupy).
[18] Je jedna z metod útoku do internetových aplikací (typicky implementovaných skriptovacími jazyky) pracující na bázi nezamýšleného požadavku pro vykonání určité akce v této aplikaci, který ovšem pochází z nelegitimního zdroje.
[19] Asynchronous JavaScript and XML.

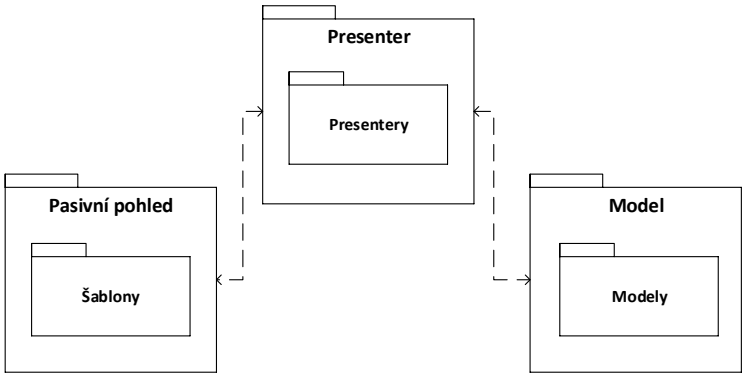
disponuje s výbornými ladícími nástroji, které ušetří mnoho času a dalšími užitečnými nástroji. Využívá událostmi řízené programování a z velké části je založen na použití komponent. Současná verze je 2.2.6.

Instalace Nette je díky komponentě composer velice snadná. Po úspěšném nainstalování composeru, stačí zavolat příkaz `composer create-project nette/sandbox nette-blog`, který stáhne potřebné soubory a vytvoří následující strukturu ve složce nette-blog [17]:

```
www/                kořenový adresář webu
 nette-blog/
    app/            adresář webové aplikace
      config/       konfigurační soubory
      model/        třídy modelové vrstvy
      presenters/   třídy presenteru (kontrolní vrstva)
      router/       konfigurace URL adres
      templates/    šablony
      bootstrap.php spouštěcí soubor
    vendor/         knihovny pro vaší aplikaci
      nette/        nette framework
      others/       ostatní knihovny
    log/            chybové logy
    temp/           místo pro dočasné soubory (cache, sessiony, atd.)
    test/           adresář pro unit testy
    www/            místní kořenový adresář webu - přístupný z internetu
```

Architektura

Nette pracuje na principu MVP[20]. MVP je softwarová architektura, která vznikla z potřeby oddělit u aplikací s grafickým rozhraním kód obsluhy (presenter) od kódu aplikační logiky (model) a od kódu zobrazujícího data (pohled). Tím jednak aplikaci zpřehledňuje, usnadňuje budoucí vývoj a umožňuje testování jednotlivých části zvlášť. MVP architektura v Nette je znázorněna na obrázku 2.1.



Obrázek 2.1: MVP architektura v Nette.

[20] Model-View-Presenter.

View vrstva

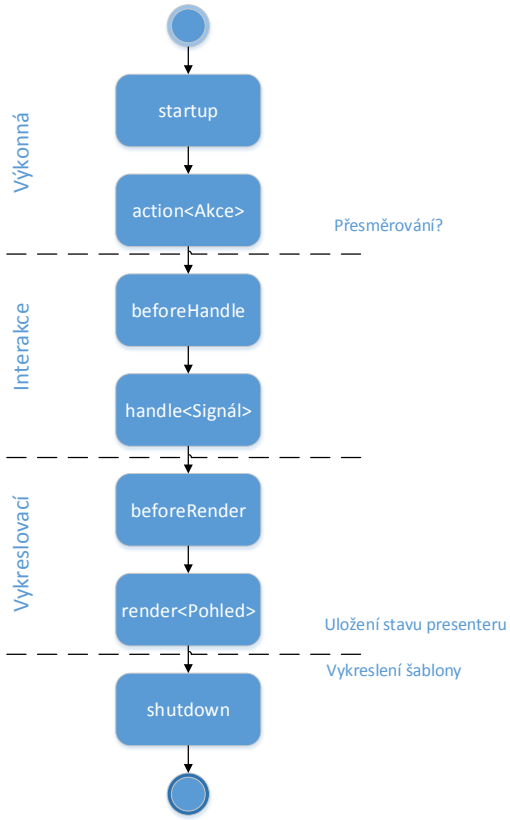
View vrstva aplikace má na starost zobrazení výsledku požadavku. Obvykle používá šablonovací systém a ví, jak se má zobrazit libovolná komponenta nebo výsledek získaný z modelu. Šablonovací systém, který je obsažen v Nette, nese název Latte. Z výše uvedené struktury je zřejmé, že šablony stránek jsou umístěny v adresáři app/templates a jeho podadresářích s koncovku **.latte**. Šablony se píši v jazyce HTML, popřípadě XHTML[21], což se dá nastavit při počáteční konfiguraci. Velice silným nástrojem jsou Latte makra a filtry. Makra se zapisují ve složených závorkách nebo jako n:makra viz. příklad:

```
{foreach $pole as $položka} <div>tělo</div> {/foreach}
<div n:foreach='$pole as $položka'>tělo</div>
```

V dokumentaci je dostupný seznam výchozích Latte maker a filtrů, které je možné uplatnit v Latte šablonách.

Presenter vrstva

Presenter vrstvu lze chápat jako ústřední výkonnou jednotku, která se stará o celkové provázání funkčnosti aplikace. V Nette se o to stará třída zvaná presenter. Následující obrázek 2.2 ilustruje, jak jsou postupně vykonávány metody presenteru v jeho životním cyklu a do jaké fáze tyto metody začleňujeme [19].



Obrázek 2.2: Životní cyklus presenteru.

[21] Extensible HyperText Markup Language.

- **Fáze výkonná:**

  - startup - Tato metoda je vyvolána na začátku životního cyklu presenteru. Může obsahovat například inicializaci proměnných. Během životního cyklu aplikace se může spustit více presenteru, metoda **startup** se může tedy volat vícekrát.

  - action<Akce> - Tato metoda by měla obsahovat vykonání operací, po kterých může následovat přesměrování. Zde probíhá například automatické přesměrování na jinou jazykovou verzi (např. podle detekce z prohlížeče). Také zde může být logika rozhodování pro členění na jednotlivé pohledy. Metoda **action** může například z validovat vstupní parametry nebo řešit exekutivu (např. zda se má záznam smazat). Onou validací můžeme chápat předání dat modelu k validaci, ověření práv a následné přesměrování do patřičných míst, pokud se vyskytne problém. Pokud bychom tedy chtěli vytvořit akci s názvem **Logout**, tak je nutné vytvořit metodu `public function actionLogout(){}`. Tuto metodu je možné poté zavolat pomocí **Presenter:logout**.

- **Fáze změn vnitřních stavů (interakce):**

  - handle<Signál> - Zpracování signálů neboli subrequestů je typicky určeno pro uživatelskou interakci a zpracování AJAXových požadavků.

- **Fáze vykreslovací:**

  - beforeRender - Typický příklad pro tuto metodu je společné nastavení filtrů a společných proměnných pro všechny šablony.

  - render<Pohled> - Má na starosti vykreslení a věci s tím spojené (přiřazení proměnných do konkrétní šablony, atd.).

- **Ukončení činnosti:**

  - shutdown - Metoda je vyvolána při ukončení životního cyklu presenteru.

Metody **startup, beforeHandle, beforeRender, shutdown** jsou společné pro všechny šablony daného presenteru. Oproti tomu metoda **render** je určena pro konkretní šablonu, **handle** pro konkretní signál a **action** pro konkretní akci.

Modelová vrstva

Model je datový a zejména funkční základ celé aplikace. Je v něm obsažena aplikační logika. Jakákoliv akce uživatele (přihlášení, změna hodnoty v databázi) představuje akci modelu. Model si spravuje svůj vnitřní stav a ven nabízí pevně dané rozhraní. Voláním funkcí tohoto rozhraní můžeme zjišťovat či měnit jeho stav. Model o existenci view nebo presenteru neví. Nette podporuje řadu databázových serverů, takže je pouze na programátorovi, který server mu vyhovuje.

Formuláře

Jak již bylo zmíněno na začátku této sekce, Nette klade velký důraz na bezpečnost aplikací, a proto úzkostlivě dbá i na dobré zabezpečení formulářů. Dělá to zcela transparentně a nevyžaduje manuálně nic nastavovat. Ochrání vaše aplikace před útokem Cross Site Scripting i Cross-Site Request Forgery, odfiltruje ze vstupů kontrolní znaky atd.

Vkládání závislostí

Vkládání závislostí (DI[22]) je v objektově orientovaném programování technika pro vkládání závislostí mezi jednotlivými komponentami programu tak, aby jedna komponenta mohla používat druhou, aniž by na ni měla v době sestavování programu referenci. Podstatou Dependency Injection (DI) je odebrat třídám zodpovědnost za získávání objektů, které potřebují ke své činnosti (tzv. služeb) a místo toho jim služby předávat při vytváření. Závislosti je možné do aplikačních tříd předávat čtyřmi hlavními způsoby [18]:

- Předávání **konstruktorem** - Závislosti, které se předávají v okamžiku vytváření objektu, kdy závislost je předána jako parametr konstruktoru:

```
class MyService
{
    /** @var AnotherService */
    private $anotherService

    public function __construct(AnotherService $service) {
        $this->anotherService = $service;
    }
}
```

- Předávání **set** metodou - Závislosti jsou předávány až po vytvoření objektu tzv. set metodou:

```
class MyService
{
    /** @var AnotherService */
    private $anotherService

    public function setAnotherService(AnotherService $service)
    {
        $this->anotherService = $service;
    }
}
```

- Předávání metodou **inject** - Tento způsob je specifický pro Nette DI kontejner. Jedná se o zvláštní případ setteru, metoda začíná prefixem inject. Základní rozdíl od předávání set metodou je ten, že Nette je takto pojmenovanou metodu schopné v presenterech najít a automaticky zavolat se správnými parametry:

```
class MyService
{
    /** @var AnotherService */
    public $anotherService;

    public function injectAnotherService(AnotherService $service)
```

[22] Dependency Injection.

```
    {
        $this->anotherService = $service;
    }
}
```

- Anotace **@inject** - Další a poslední způsob specifický pro Nette DI kontejner. Jedná se o zvláštní případ závislostí předávaných pomocí public proměnné. Tuto proměnnou je nutné označit pomocí anotace **@inject** v komentáři. Typ předávané závislosti je také uveden v komentáři:

```
class MyService
{
    /** @inject @var \App\AnotherService */
    public $anotherService;
}
```

První dva způsoby platí obecně ve všech objektově orientovaných jazycích, zbylé jsou specifické pro Nette. Jistě si říkáte, že přece nelze delegovat zodpovědnost do nekonečna. Že musí být nějaký počátek. Úplně na začátku je nutné vytvořit DI kontejner, který už nic nedeleguje a objekty tvoří. Říká se mu systémový kontejner. Prvotní systémový kontejner (*Nette\DI\Container*) v Nette vytváří třída *Nette\Con\Configurator*.

AJAX

AJAX je technologie, díky níž je možné měnit obsah části webových stránek bez nutnosti opětovného načtení celé stránky. Pojmem AJAX chápeme několik technologií, které se využívají současně. Jde především o HTML, CSS, DOM, JavaScript a objekt XMLHttpRequest, který slouží pro asynchronní komunikaci se serverem. V dnešní době AJAX slouží k tvorbě interaktivních webových stránek. Jeho název a popis se poprvé veřejně objevil v roce 2005 v článku *Ajax: A New Approach to Web Applications* od Jesse James Garreta. Ten je také dnes považován za autora tohoto pojmu. Mezi nevýhody technologie AJAX patří především větší objem přenesených dat z důvodu vyššího počtu HTTP požadavků.

Samotné Nette s touto technologií pracuje velice dobře. Podporuje výřezy šablon pomocí snippetů. Dále podporuje předávání proměnných mezi PHP a JavaScriptem. V rámci vývojářské komunity vyšlo i několik rozšíření k jednoduší práci s touto technologií.

2.3 Přenos dat

Pro přenos dat se používají různé protokoly. Protokol vymezuje strukturu a způsob předávání zpráv mezi dvěma komunikujícími prvky. V nejjednodušší podobě protokol definuje pravidla řídící syntaxi, sémantiku a synchronizaci vzájemné komunikace. Protokoly mohou být realizovány hardwarově, softwarově anebo kombinací obou.

2.3.1 TCP

Protokol TCP[23] je spojově orientovaný protokol pro přenos toku bajtů na transportní vrstvě TCP/IP modelu. Zasílá data z aplikační vrstvy jako proud dat, který je tvořen z číslovaných paketů zasílaných v určeném pořadí k cíli. Zajišťuje spolehlivý přenos dat. K tomu má implementováno řízení toku, přijetí paketů ve správném pořadí, jejich potvrzování a řízení zahlcení. Odesílatel i příjemce pracuje s tzv. posuvným okénkem (sliding window), jehož velikost se v průběhu přenosu mění. Jeho velikost udává maximální možný počet nepotvrzených paketů.

<table>
<tr><th>Bity</th><th colspan="32">0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</th></tr>
<tr><td>0</td><td colspan="16">Zdrojový port</td><td colspan="16">Cílový port</td></tr>
<tr><td>32</td><td colspan="32">Číslo sekvence</td></tr>
<tr><td>64</td><td colspan="32">Potvrzený paket</td></tr>
<tr><td>96</td><td colspan="4">Offset dat</td><td colspan="6">Rezervováno</td><td colspan="6">Příznaky</td><td colspan="16">Okénko</td></tr>
<tr><td>128</td><td colspan="16">Kontrolní součet</td><td colspan="16">Urgent data</td></tr>
<tr><td>160</td><td colspan="32">Volby (volitelné)</td></tr>
<tr><td>192</td><td colspan="24">Volby (pokračování)</td><td colspan="8">Zarovnání</td></tr>
<tr><td>224</td><td colspan="32">Data</td></tr>
</table>

Obrázek 2.3: Struktura TCP paketu.

TCP spojení je možné rozdělit do tří fází, a sice navázaní spojení, přenos dat a ukončení spojení. K navázání spojení slouží three-way handshake. V průběhu navazování spojení se obě strany dohodnou na čísle sekvence a potvrzovacím čísle. Pro navázání spojení se odesílají pakety s nastavenými příznaky SYN a ACK. Jak z názvu vyplývá, navázání spojení probíhá ve třech krocích [25]:

- Klient odešle paket na server s nastaveným příznakem SYN a náhodně vygenerovaným číslem sekvence X, potvrzovací číslo je nastaveno na 0.
- Server odešle klientovi paket s nastavenými příznaky SYN a ACK, potvrzovací číslo je nastaveno na hodnotu X+1, číslo sekvence je náhodně vygenerované Y.
- Klient odešle paket zpět na server s nastaveným příznakem ACK, číslo sekvence nastaví na hodnotu X+1, číslo odpovědi nastaví na hodnotu Y+1.

[23] Transmission Control Protocol.

Ukončení spojení probíhá podobně jako jeho navázání. Používá se k tomu příznaků FIN a ACK:

- Klient odešle paket s nastaveným příznakem FIN.
- Server odpoví paketem s nastaveným příznakem ACK.
- Server odešle paket s nastaveným příznakem FIN.
- Klient odpoví paketem s nastaveným příznakem ACK.

2.3.2 Webové sokety

Webový soket [10] je protokol, který poskytuje plně duplexní režim[24] v rámci jednoho spojení. Je navržen tak, aby bylo možné s tímto protokolem pracovat v internetových prohlížečích. Jedná se o nezávislý protokol postavený na TCP. Protokol Webový soket je ideální pro tvorbu aplikací v reálném čase, protože je vytvořeno pouze jedno plně duplexní spojení mezi serverem a klientem, přes které je možné komunikovat. Protokol Webový soket je v současné době podporován ve většině prohlížečů, včetně Google Chrome, Internet Explorer, Firefox, Safari a Opera.

Navázání spojení

K navázaní spojení mezi serverem a klientem je nutný tzv. handshake[25]. Toto navázání probíhá v rámci HTTP protokolu. Navázání je zahájeno na žádost klienta, který vyšle speciální klíč s žádosti. Server zpracuje tuto žádost a odešle zpět potvrzení. Tím je zajištěno, že spojení může být navázáno pouze se serverem, který protokol Webový soket podporuje. Nyní si ukážeme klientský požadavek k navázání komunikace [9]:

```
GET HTTP/1.1
   Upgrade: Websocket
   Connection: Upgrade
   Host: echo.websocket.org
   Origin: http://www.websocket.org
   Sec-WebSocket-Key: i9ri‘AfOgSsKwUlmLjIkGA==
   Sec-WebSocket-Version: 13
   Sec-WebSocket-Protocol: chat
```

Klient odešle požadavek GET pro aktualizaci protokolu. Sec-WebSocket-Key je jen soubor náhodných bytů. Server tyto byty vezme a připojí k nim speciální jedinečný identifikátor (GUID[26]) 258EAFA5-E914-47DA-95CA-C5AB0DC85B11. Poté z tohoto vytvořeného řetězce vytvoří hash pomocí algoritmu SHA-1[27], který je poté zakódován pomocí Base64[28]. Zakódována hodnota je zkopírována do části Sec-WebSocket-Accept v záhlaví odpovědi.

[24] Duplexní režim je taková komunikace (popř. přenos dat) mezi dvěma subjekty, při které mohou být data odesílána oběma směry současně.
[25] Handshake je název procesu, který je používán pro ustavení spojení.
[26] Globally unique identifier.
[27] Secure Hash Algorithm.
[28] Base64 je kódování, které převádí binární data na posloupnosti tisknutelných znaků.

Nyní si ukážeme odpověď serveru:

```
HTTP/1.1 101 Web Socket Protocol Handshake
  Upgrade: WebSocket
  Connection: Upgrade
  Sec-WebSocket-Accept: Qz9Mp4/YtIjPccdpbvFEm17G8bs=
  Sec-WebSocket-Protocol: chat
  Access-Control-Allow-Origin: http://www.websocket.org
```

Jakmile je spojení navázáno, může klient a server odesílat data v plně duplexním režimu pomocí upraveného TCP protokolu [9].

Klient

Na straně klienta jsou Webové sokety implementovány v JavaScriptu jako třída WebSocket. Programátor může vytvořit instanci této třídy, a tím navázat spojení se serverem (samozřejmě musí podporovat technologii Webových soketů).

```
var ws = new WebSocket("ws://www.websocket.org/echo/");

ws.onopen = function(e) {
  alert("Spojení otevřeno ...");
  ws.send("Zdravím");
};
ws.onmessage = function(e) {
  alert( "Přijata zpráva: " + e.data);
};
ws.onclose = function(e) {
  alert("Spojení uzavřeno.");
};
```

Pro URL[29] Webových soketů se používá prefix ws://. Pokud spojení používá SSL[30], lze použít wss://. Při vytvoření nového objektu třídy WebSocket je navázáno spojení pomocí protokolu ws/wss. Objekt WebSocket nabízí události, nazvané **onopen**, **onmessage** a **onclose**. Událost **onopen** je volaná při otevření spojení. Může sloužit jako příznak toho, že již lze posílat zprávy. Událost **onclose** zase oznamuje uzavření spojení. Pro vlastní výměnu dat slouží událost **onmessage**. Událost **onmessage** je zavolána ve chvíli, když ze serveru přijde jakákoliv zpráva. Posílání zpráv obstarává metoda **send**. Jejím parametrem je řetězec, který má být poslán na vzdálený server. Spojení je možné uzavřít pomocí metody **disconnect**.

Server

Na straně serveru je zapotřebí víc než běžný HTTP server (nutná podpora technologie Webových soketů).

[29] Uniform Resource Locator je řetězec znaků s definovanou strukturou, který slouží k přesné specifikaci umístění zdrojů informací na Internetu.

[30] Secure Sockets Layer je protokol, který poskytuje zabezpečení komunikace šifrováním a autentizaci komunikujících stran.

Serverů, které podporují Webové sokety, je několik:

- **Java**
  - Kaazing Gateway
  - Atmosphere
  - GlassFish 3.0
- **C#**
  - SuperWebSocket
  - Nugget
  - Alchemy-WebSockets
- **PHP**
  - Ratchet
  - PHPWebSocket
  - PHPDaemon
- **Python**
  - Tornado
  - PyWebSockets
  - Autobahn

Uvedl jsem pouze několik příkladů. V rámci této práce byl vybrán framework Tornado především kvůli implementaci v programovacím jazyce Python.

2.3.3 HTTP

HTTP je v současnosti nejpoužívanější protokol pro přenos dat mezi klientem a serverem. Přestože je určen pro výměnu hypertextových dokumentů ve formátů HTML, díky standardu MIME[31] můžeme přenášet i jiné libovolné soubory. HTTP používá TCP spojení na portu 80.

První verze HTTP/0.9 byla publikována v roce 1991. Jednalo se o nejjednodušší možnou implementaci. Server na obdržený požadavek klienta rovnou zasílal požadovaný dokument. Verze HTTP/1.0 vyšla v roce 1996, která začala využívat internetový standard MIME pro identifikaci typu dokumentu. Aktuální verze tohoto protokolu vyšla až o rok později a je označena jako HTTP/1.1. Tato verze je stabilní a v roce 1999 byla její specifikace vydána pod normou RFC 2616 [11].

Činnost tohoto protokolu je založena na principu dotaz-odpověď. Klient požadující nějaký dokument zašle požadavek na server, který na takovou žádost odpoví. Požadavek mimo jiné obsahuje identifikátor požadovaného dokumentu. Protokol HTTP nazýváme bezstavový, protože při dalším požadavku klienta server nepozná, zda nový požadavek souvisí s předchozím či nikoliv.

[31] Multipurpose Internet Mail Extensions.

K protokolu HTTP existuje také jeho bezpečnější verze HTTPS[32]. HTTPS používá protokol HTTP, přičemž přenášená data jsou šifrována pomocí SSL nebo TLS[33] a standardní port na straně serveru je 443.

Podporovaných dotazovacích metod v HTTP/1.1 je několik a mezi nejznámější patří GET, HEAD, POST.

- GET - Vyjadřuje požadavek na zaslání určitého objektu. Umožňuje i předání omezeného množství parametrů. Jedná se o často používanou metodu, jelikož se používá např. pro stažení HTML souboru při jeho zobrazovaní v internetovém prohlížeči.
- HEAD - Stejně jako GET vyjadřuje zájem o určitý objekt. Server však neodpovídá objektem, ale pouze hlavičkou, ve které jsou vyplněny metadata objektu.
- POST - Metoda slouží k odeslání dat od klienta na server. Umožňuje předat teoreticky neomezený objem dat. Proto se často využívá při odesílání formulářů z webových stránek nebo souboru.

2.3.4 XML-RPC

XML-RPC[34] je protokol, s jehož pomocí lze jednoduše provádět vzdálené volání procedur[35]. Tento protokol zapouzdřuje data pomoci značkovacího jazyka XML, která jsou přenášena pomocí protokolu HTTP.

Požadavek se skládá ze dvou částí. Jedná se o hlavičku a tělo požadavku.

Hlavička požadavku

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181
```

První údaj v prvním řádku hlavičky je druh dotazu. Druhá položka nese informaci o umístění XML-RPC serveru. Třetí údaj definuje verzi a druh protokolu.

Další čtyři řádky se skládají vždy z názvu položky a hodnoty. První ze čtveřice `User-agent` obvykle informuje o druhu a verzi implementace. Řádek `Host` určuje adresu počítače, na kterém běží XML-RPC server. `Content-Type` značí druh odesílaných dat, musí mít vždy hodnotu `text/xml`. `Content-length` udává délku těla požadavku.

[32] Hypertext Transfer Protocol Secure.
[33] Transport Layer Security je nástupce protokolu SSL poskytující možnost zabezpečené komunikace.
[34] Extensible Markup Language - Remote procedure call.
[35] Tato technologie umožňuje programu vykonat proceduru, která může být uložena na jiném místě, než je umístěn volající program.

Tělo požadavku

```
<?xml version="1.0"?>
<methodCall>
   <methodName>examples.getStateName</methodName>
   <params>
      <param>
         <value>
            <int>41</int>
         </value>
      </param>
   </params>
</methodCall>
```

Tělo požadavku musí uvozovat kořenový element `<methodCall>`. Podelement `<methodName>`, udává název volané procedury. Následuje seznam parametrů předávaných vzdálené proceduře (pokud nějaké vyžaduje). Parametry se uvozují elementem `<params>`, ve kterém je možné specifikovat libovolný počet parametrů. Každý parametr je uzavřen v elementu `<param>` a v tomto elementu se nachází jeho hodnota v elementu `<value>` obalená elementem datového typu [13].

Úspěšná odpověď

```
HTTP/1.1 200 OK
Date: Sun, 14 Apr 2013 01:43:38 GMT
Server: Apache
Vary: Accept-Encoding,User-Agent
Content-Length: 1298
Connection: close
Content-Type: text/html

<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <value><string>Odpověď</string></value>
       </param>
   </params>
</methodResponse>
```

Tělo odpovědi tvoří element `<methodResponse>`, mezi kterými se, na rozdíl od požadavku, musí vyskytnout element `<params>`. Uvnitř musí existovat alespoň jeden element `<param>` obsahující hodnotu a datový typ.

V případě neúspěchu je element `<methodResponse>` následován elementem `<fault>`. Uvnitř se nachází element `<value>` obsahující datový typ `<struct>`. Struct obsahuje dva členy – první s názvem `faultCode`, který udává chybovou hodnotu, a druhý člen je `faultString`, který obsahuje textovou část chyby.

2.3.5 IPC soket

IPC soket, také jinak Unixový doménový soket (UDS), je označení pro jeden z možných prostředků meziprocesové komunikace, který slouží pro předávání dat mezi různými procesy v rámci jednoho počítače. UDS podporují, jak spolehlivý, tak i nespolehlivý přenos dat. UDS jsou standardní součástí POSIXu[36] unixového operačního systému.

API pro UDS je podobné internetovým soketum, ale místo základních síťových protokolů používá pro komunikaci jádro operačního systému. UDS používají souborový systém jako jejich adresový jmenný prostor. Odkazují se pomocí procesu jako inode[37] v souborovém systému, což umožňuje dvěma procesům otevřít stejný soket a v daném pořadí komunikovat. Po úspěšném navázání procesy komunikují pomocí send a recv zpráv [28].

[36] Portable Operating System Interface je informatice označení standardu používaného hlavně unixovými operačními systémy, jehož úkolem bylo vytvořit jednotné rozhraní.

[37] Inode je v informatice datová struktura uchovávající metadata o souborech a adresářích používaná v unixových souborových systémech.

Kapitola 3

Použité nástroje a testování aplikací na platformě Android

Android je operační systém založený na Linuxovém jádru, který je primárně určen pro dotykové mobilní zařízení (tablety, mobilní telefony, PDA). Na rozdíl od svého největšího konkurenta, kterým je operační systém iOS od firmy Apple, je Android šířen volně pod licencí open-source[1].

V dnešní době Android patří k nejvíce expandujícímu operačnímu systémů, což je hlavním důvodem, proč byl tento systém zvolen. V budoucnu se ovšem nevylučuje i zahrnutí testování operačního systému iOS, popřípadě Windows Phone.

3.1 Nástroje vývojářského balíčku

Pro vývoj aplikací je určený balík Android SDK tzv. Software Development Kit, který je dostupný pro všechny operační systémy. Balík obsahuje nástroj na ladění chyb, sadu knihoven, dokumentaci a další důležité nástroje. V rámci této práce budou pro nás nejdůležitější nástroje `adb`[2], `monkeyrunner` a `aapt`[3]. Nyní si blíže popíšeme jednotlivé nástroje.

3.1.1 Android Debug Bridge

Android Debug Bridge (ADB) [1] je univerzální nástroj, díky kterému jsme schopni komunikovat s připojeným Android zařízením, popřípadě emulátorem pomocí příkazové řádky. ADB je založeno na komunikačním modelu klient-server a skládá se z těchto komponent:

- **Klient** - Klienta, který běží na uživatelově počítači, je možné vyvolat zadáním jakéhokoliv ADB příkazu do příkazové řádky.
- **Server** - Server, který taktéž běží na pozadí uživatelova počítače, slouží k řízení komunikace mezi klientem a démonem[4] na Android zařízení.
- **Démon** - Démon, který běží na pozadí každého Android zařízení či emulátoru.

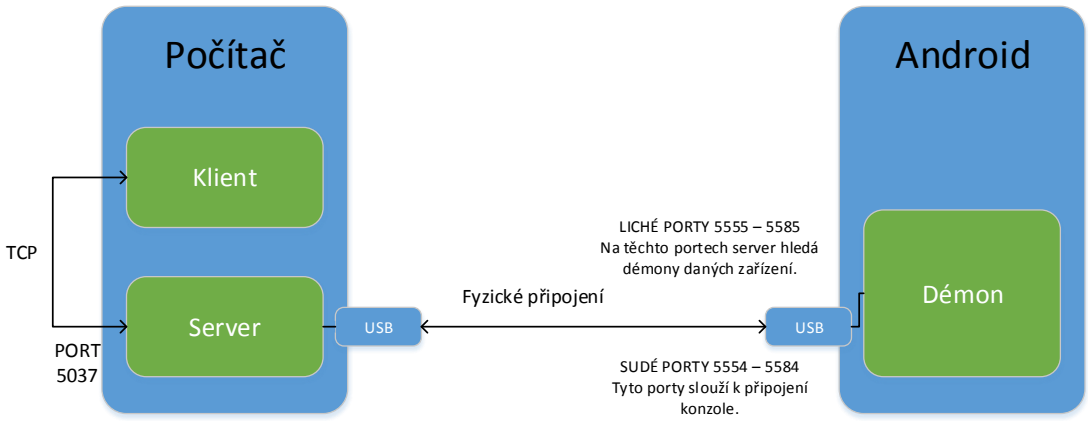
[1] Otevřený software (anglicky open-source) je počítačový software, jehož zdrojové kódy jsou poskytovány za účelem volného užívání a úprav za dodržení určitých licenčních podmínek.

[2] Android Debug Bridge.

[3] Android Asset Packaging Tool.

[4] Démon je v informatice označení programu, který je spuštěn dlouhodobě a není v přímém kontaktu s uživatelem.

Pokud zadáme jakýkoliv ADB příkaz, klient nejprve zkontroluje zdali běží proces serveru, pokud ne, vytvoří se server na pozadí, který naslouchá na TCP portu 5037. Na obrázku 3.1 je znázorněna architektura ADB.



Obrázek 3.1: Architektura Android Debug Bridge.

Pokud chceme využívat ADB pomocí USB připojení, je nutné povolit USB ladění ve vývojářských možnostech. Nyní si ukážeme pár základních ADB příkazů.

- Pro zobrazení připojených zařízení k počítači uživatele slouží příkaz:

  ```
  adb devices
  ```

- Pokud máme zapojených více zařízení k jednomu počítači je nutné rozlišit, se kterým zařízením chceme právě komunikovat. K tomuto účelu slouží parametr `-s`, díky kterému jsme schopni rozlišit jednotlivé zařízení:

  ```
  adb -s <serialNumber> <command>
  ```

- Další užitečné příkazy jsou k zastavení a nastartování ADB serveru:

  ```
  adb kill-server
  ```

  ```
  adb start-server
  ```

Spousty dalších užitečných příkazů je možné nalézt na webových stránkách Android vývojářů [1]. V závěru kapitoly 4, která se zabývá návrhem výsledného systému, bude tento protokol důkladně analyzován.

3.1.2 MonkeyRunner

Dalším užitečným nástrojem je MonkeyRunner [16]. Tento nástroj zprostředkovává API[5], díky kterému jsme schopni ovládat Android zařízení, popřípadě emulátor. Nástroj MonkeyRunner používá Jython[6], proto je možné zařízení ovládat pomocí programovacího jazyka Python. Jedna z předností tohoto nástroje je možnost ovládání více zařízení najednou. MonkeyRunner obsahuje tři moduly, které jsou obsaženy v balíčku `com.android.monkeyrunner`.

[5] Application Programming Interface.
[6] Jython představuje implementaci programovacího jazyka Python v jazyce Java.

Nyní si blíže popíšeme jednotlivé moduly:

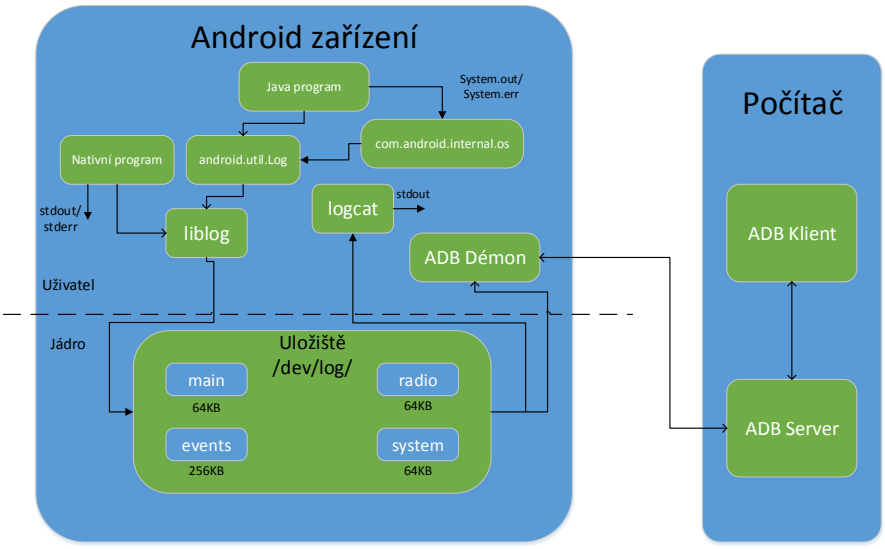
- **MonkeyRunner** - Obsahuje jednu z nejpodstatnějších metod, a to připojení k danému zařízení, se kterým budeme komunikovat.
- **MonkeyDevice** - Modul představuje samotné zařízení či emulátor. Poskytuje nám metody pro instalaci aplikačních balíčků a možnost informovat zařízení o uživatelových akcích (dotek obrazovky, stisk klávesy).
- **MonkeyImage** - Poslední modul zachycuje obrazovku daného zařízení. Je možné vytvořit aktuální obraz displeje a poté s tímto obrazem libovolně pracovat.

Pokud tedy chceme pracovat s těmito moduly v programovacím jazyku Python, je nutné tyto moduly importovat, a to následovně:
`from com.android.monkeyrunner import <modul>` , kde modul je jeden z výše popsaných modulů.

3.1.3 Android Asset Packaging Tool

Android Asset Packaging tool je užitečný nástroj pro práci s APK[7] balíčky. S tímto nástrojem jsme schopni jednoduše zjistit oprávnění k danému balíčku, aktuální verzi vývoje, cílovou verzi SDK atd. Například zjištění aktuální verze APK balíčku je možné provést pomocí `aapt dump badging <cesta k apk souboru> | grep "version"` (v operačním systému Linux).

3.1.4 Záznamový systém



Obrázek 3.2: Android záznamový systém [15].

[7] Je formát souboru balíčku, který se používá k distribuci a instalování aplikace pro operační systém Android.

Záznamový systém v Androidu, který je zobrazen na obrázku 3.2, obsahuje následující části:

- Úložiště v systémovém jádru pro uložení záznamu a řadič.
  - main - Hlavní log aplikace.
  - events - Informace systémových událostí.
  - radio - Zprávy, které se vztahují k telefonu nebo rádiu.
  - system - Záznam pro nízkoúrovňové systémové zprávy.
- C, C++ a Java třídy, které vytvářejí aktuální záznamy.
- Pro nás nejpodstatnější, samostatný program pro prohlížení záznamů (logcat).

Je důležité poznamenat, že ve výchozím nastavení pracuje **logcat** pouze s `/dev/log/main` a `/dev/log/system` úložišti. Pokud chceme pracovat i s jinými úložišti je možné použít parametr `-b`, a sice `adb logcat -b radio`.

3.2 Nativní přístup k zařízení

Android NDK[8] poskytuje všechny nástroje (kompilátory, knihovny a hlavičkové soubory), které umožňují nativní přístup k danému zařízení. Nativní kód, který je psaný v C či C++, slouží především k urychlení kritických pasáží ve zdrojovém kódu. Poté je tento kód možný přeložit pro cílovou architekturu ARM[9], MIPS[10], popřípadě x86. Následně je spustitelný kód možný nahrát a spustit buď pomocí ADB konzole, nebo volat nativní kód přímo z Javy pomocí rozhraní JNI[11]. Je nutné podotknout, že použití Android NDK však neznamená vždy zvýšení výkonu aplikace, takže je nutné s tímto nástrojem pracovat obezřetně. V této práci bude práce s Android NDK převážně pro experimentální účely s možnosti zrychlení určitých pasáží (např. získávání obrazu zařízení).

Pro překlad zdrojových souborů jsou pro uživatele důležité soubory **Android.mk** a **Application.mk**. V souboru **Android.mk** uživatel definuje vytvořené C, popřípadě C++ soubory pro NDK. Jedná se o malý fragment GNU[12] Makefile používaný při překladu a kompilaci. Dále je nutné definovat, zda se jedná o vytvoření statické či sdílené knihovny, nebo o samostatně spustitelný soubor. V souboru **Application.mk** uživatel definuje cílovou platformu např. ARM, cílové Android API apod.

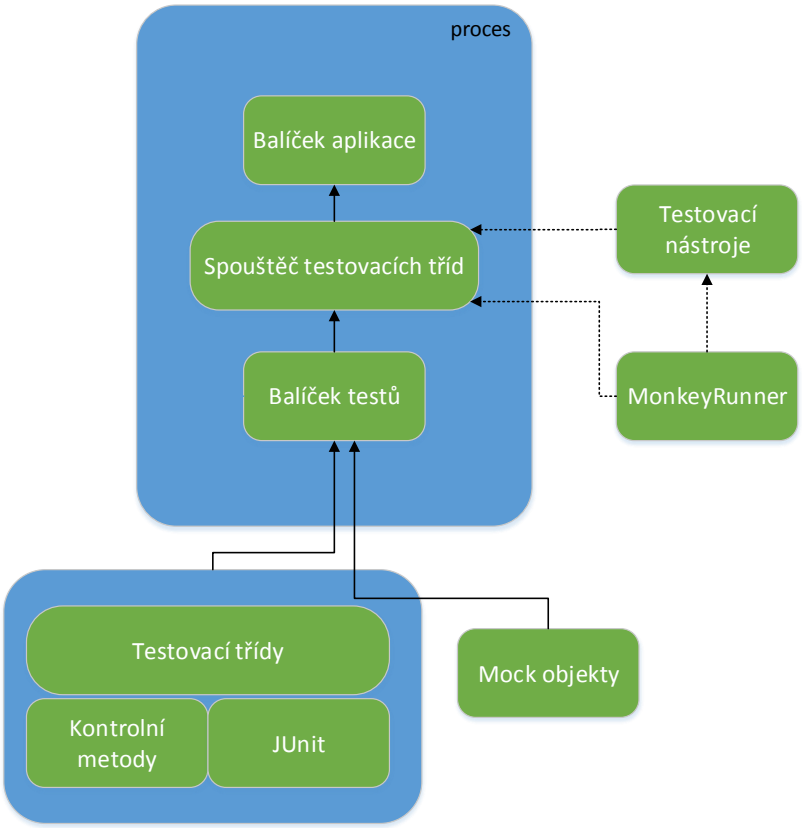
3.3 Automatizované testování

Jak již bylo zmíněno v úvodu, další části této práce bude automatizované hromadné testování Android aplikací. Testování Android aplikací je důležité vzhledem k obrovskému množství dostupných zařízení. S automatizovaným testem souvisí pojem **unit test**. Pojem unit test v kontextu informačních technologií označuje automatické testování a ověřování funkčnosti určité jednotky. Za jednotku je považována samostatně testovatelná část aplikačního

[8] Native Development Kit.
[9] Advanced RISC Machines.
[10] Microprocessor without Interlocked Pipeline Stages.
[11] Java Native Interface je rozhraní umožňující propojit kód běžící na virtuálním stroji Javy s nativními programy v jiných jazycích.
[12] GNU's Not Unix.

programu. Z pohledu procedurálního programování může být jednotkou program, funkce, procedura atd. Z pohledu objektově orientovaného programování je jednotkou obvykle třída, či konkrétní metoda. Automatizované testování aplikací se dnes nejčastěji provádí pomocí dostupných frameworků.

Základem téměř všech dostupných frameworků pro automatizované testování Android aplikací je **Android Instrumentation Framework**. Tento framework poskytuje architekturu a výkonné nástroje, které pomohou testovat výslednou aplikaci. Architektura je znázorněna na obrázku 3.3.

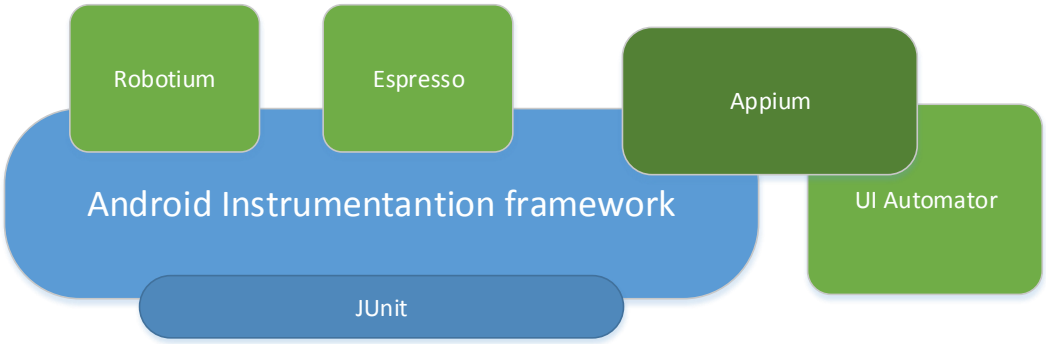


Obrázek 3.3: Architektura Android Instrumentation Frameworku [26].

Spouštěč testovacích tříd je základna pro Android testy. Jedná se o hlavní třídu, která nahrává a spouští testovací metody definované uživatelem. Prostřednictvím API komunikuje se systémem Android. Pokud se spustí test pro Android aplikaci a nějaký test již běží, tento test je Android systémem vypnut a poté načte novou instanci. Kontrolní metody řídí životní cyklus složek aplikace. Mock objekty jsou metody pro tvorbu systémových objektů, jako jsou například intenty[13].

Mezi nejznámější frameworky automatizovaného testování, které využívají Android Instrumentation architekturu, jsou Appium, Robotium, Espresso. Následující obrázek 3.4 znázorňuje rodinu testovacích frameworků.

[13]Intent je abstraktní popis operace, které má být provedena.



Obrázek 3.4: Ukázka frameworků automatizovaného testování.

3.3.1 UI Automator

Android SDK obsahuje nástroje UI Automator a spouštěcí engine. UI Automator je Java knihovna, která slouží k tvorbě UI testů. Druhým nástrojem je engine, který slouží ke spuštění těchto testů. Oba nástroje fungují pouze pro Android API 4.2 a vyšší.

UI Automator test je samostatný Java projekt. V tomto projektu je nutné pouze přilinkovat knihovny uiautomator.jar a android.jar z nástrojů Android SDK.

Samotný UI Automator poskytuje třídu **UiDevice** ke komunikaci se samotným zařízením. Dále **UiSelector** k vyhledávání elementů na obrazovce a **UiObject**, který reprezentuje UI elementy. Poskytuje samozřejmě ještě další třídy, jako například **UiCollection**, **UiScrollable** a další. Více o UI Automator na webových stránkách [27].

3.3.2 Espresso

V říjnu 2013 Google uvolnil další framework s názvem Espresso. Espresso je tenká vrstva na vrcholu Android Instrumentation Frameworku. Cílem tohoto frameworku byla tvorba rychlých, snadných a spolehlivých UI testů. Espresso je založen na JUnit 3 a využívá Hamcrest knihovnu pro své kontroly. Více o frameworku Espresso na webových stránkách [7].

3.3.3 Robotium

Robotium je automatizační testovací framework, který má plnou podporu pro nativní vývoj. Robotium usnadňuje psát výkonné a robustní automatizované black-box[14] UI testy. S podporou Robotia může vývojář napsat systémové a uživatelské akceptační testy. Dále zvládá obsluhovat více Android aktivit najednou. Více o frameworku Robotium na webových stránkách [21].

[14]Black-box je metoda testování softwaru, která zkoumá funkčnost aplikace, aniž by znala informace o vnitřních strukturách nebo fungování.

3.3.4 Appium

V této práci bude použit framework Appium. Appium je open-source nástroj pro automatizované testování nativních, webových, a hybridních aplikací pro operační systémy iOS a Android. Rozdíl mezi webovou, nativní a hybridní aplikací je popsán v mojí bakalářské práci. Více na webových stránkách [8].

Appium používá pro testování aplikací dva rozdílné frameworky pro platformu Android. Pro Android API 2.3 a vyšší využívá Android Instrumentation Framework. Druhým frameworkem je UI Automator, který je možný využít pro Android API 4.2 a vyšší. Podpora pro Instrumentation je poskytnuta skrze samostatný projekt Selendroid. V této práci bude prozatím využita první možnost, a sice podpora skrze Selendroid z důvodu podpory nižšího Android API.

Appium je založeno na klient-server architektuře. Jádro Appium je webový server, který naslouchá na určitém portu. Po úspěšném připojení klienta k serveru, klient posílá příkazy, které server vykoná na mobilním zařízení a poté výsledek posílá zpět formou HTTP (podrobněji v sekci 2.3.3). Jelikož je Appium postaveno na klient-server architektuře, je možné psát klientský testovací kód v jakémkoli jazyce, který podporuje HTTP klient API. Je ovšem jednoduší využít již existujících klientských Appium knihoven. Appium server je napsán ve frameworku Node.js. Appium klienty je možné naprogramovat v Pythonu, Javě, Ruby, PHP, C# a dalších jazycích. V rámci této práce budou prozatím podporovány klientské skripty napsané v jazyce Python s možností dalšího rozšíření. Více o Appium na webových stránkách na [3].

Výsledné shrnutí popsaných frameworků je znázorněno v tabulce 3.1.

	Robotium	**UI Automator**	**Espresso**	**Appium**
Android	ANO	ANO	ANO	ANO
iOS	NE	NE	NE	ANO
Mobilní vývoj	ANO	LIMITOVÁN	NE	ANO
Skriptovací jazyk	Java	Java	Java	skoro neomezený
Podporovaná API	neomezená	17 a výš	8, 10, 15-19	neomezená

Tabulka 3.1: Shrnutí testovacích frameworků.

Kapitola 4

Návrh systému pro vzdálené ovládání a hromadné testování aplikací

Následující kapitola se zabývá analýzou dostupných řešení a následným návrhem vlastního řešení systému pro automatizované hromadné testování Android aplikací. Postupně jsou uváděny klíčové návrhové diagramy vytvořené v programu Microsoft Visio. Diagramy zachycují nejdůležitější části výsledného informačního systému. V závěru této kapitoly je provedena analýza ADB protokolu, kterou jsem zmiňoval na konci sekce 3.1.1.

4.1 Analýza dostupných řešení

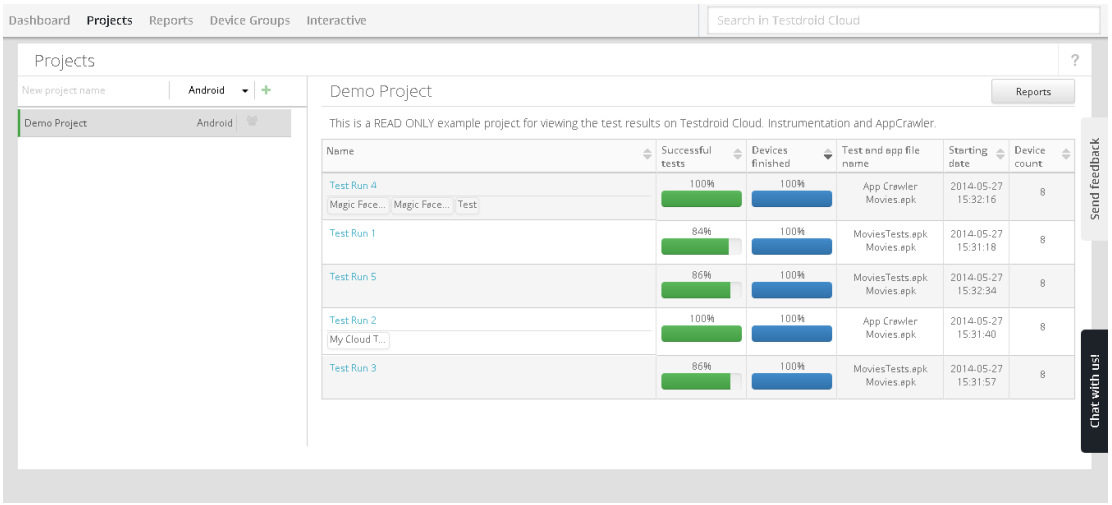
Tato sekce popisuje již existující webové řešení dané problematiky. Mezi tyto řešení patří Testdroid[1] a pCloudy[2].

Testdroid

Testdroid poskytuje vývojářům webové rozhraní, skrze které je možné testovat vyvíjenou aplikaci na mnoha (řádově stovky) různých zařízení. Poskytuje vývojáři různé statistiky o provedených testech. Výsledek testu je doprovázen obrázky, které informují o průběhu daného testu. Na obrázku 4.1 je ukázka vytvořeného Demo projektu, nad kterým bylo spuštěno 5 testů. U každého testu jsou poskytnuty informace o jeho průběhu, a sice jestli byl dokončen úspěšně, na kolika zařízeních se testovalo, datum spuštění skriptu, název testového souboru a aplikace, nad kterou byl test spuštěn. Hromadné testování je zajištěno pomocí frameworků, které byly popsány v sekci 3.3.

Testdroid neposkytuje osobitý přístup k jednotlivým zařízením. Jedná se pouze o hromadné testování aplikace na více zařízeních. Tato služba je ovšem placená (měsíční platby).

[1] testdroid.com
[2] www.pcloudy.com



Obrázek 4.1: Ukázka webového rozhraní Testdroid.

pCloudy

PCloudy poskytuje oproti Testdroidu osobitý přístup k jednotlivým zařízením. Podporuje také hromadné testování, ale to není zdaleka tak propracované jako v případě Testdroid. Ukázka webového rozhraní pCloudy je zobrazena na obrázku 4.2.

Obrázek 4.2: Ukázka webového rozhraní pCloudy.

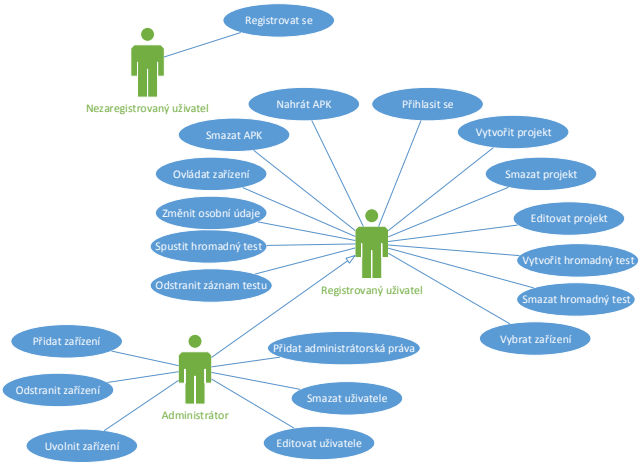
Uživatel má aktuální informace o obrazu zařízení, vytíženosti procesoru a paměti mobilu. Zařízení je možné libovolně ovládat. Dále může uživatel nahrát libovolnou aplikaci na dané zařízení. Má k dispozici nástroje pro ovládání zvuku, vstup klávesnice a další. Tato služba je také placená (měsíční platby).

Obě zmíněné existující řešení používají mnoho (řádově stovky) dostupných zařízení, na kterých je možné testovat danou aplikaci. Což je podle mě důvodem, proč se za tyto služby platí nemalé peníze. Proto jsem navrhl řešení, které by bylo možné nainstalovat na svůj vlastní server a testovat na vlastních zařízeních.

4.2 Specifikace požadavků

Cílem je navrhnout informační systém (bude umožněna instalace na vlastním serveru), který bude zprostředkovávat vzdálený přístup k mobilním zařízením připojených k serveru skrze ADB. Kromě samotného přístupu by měl systém podporovat hromadné (na více zařízeních současně) testování uživatelových aplikací.

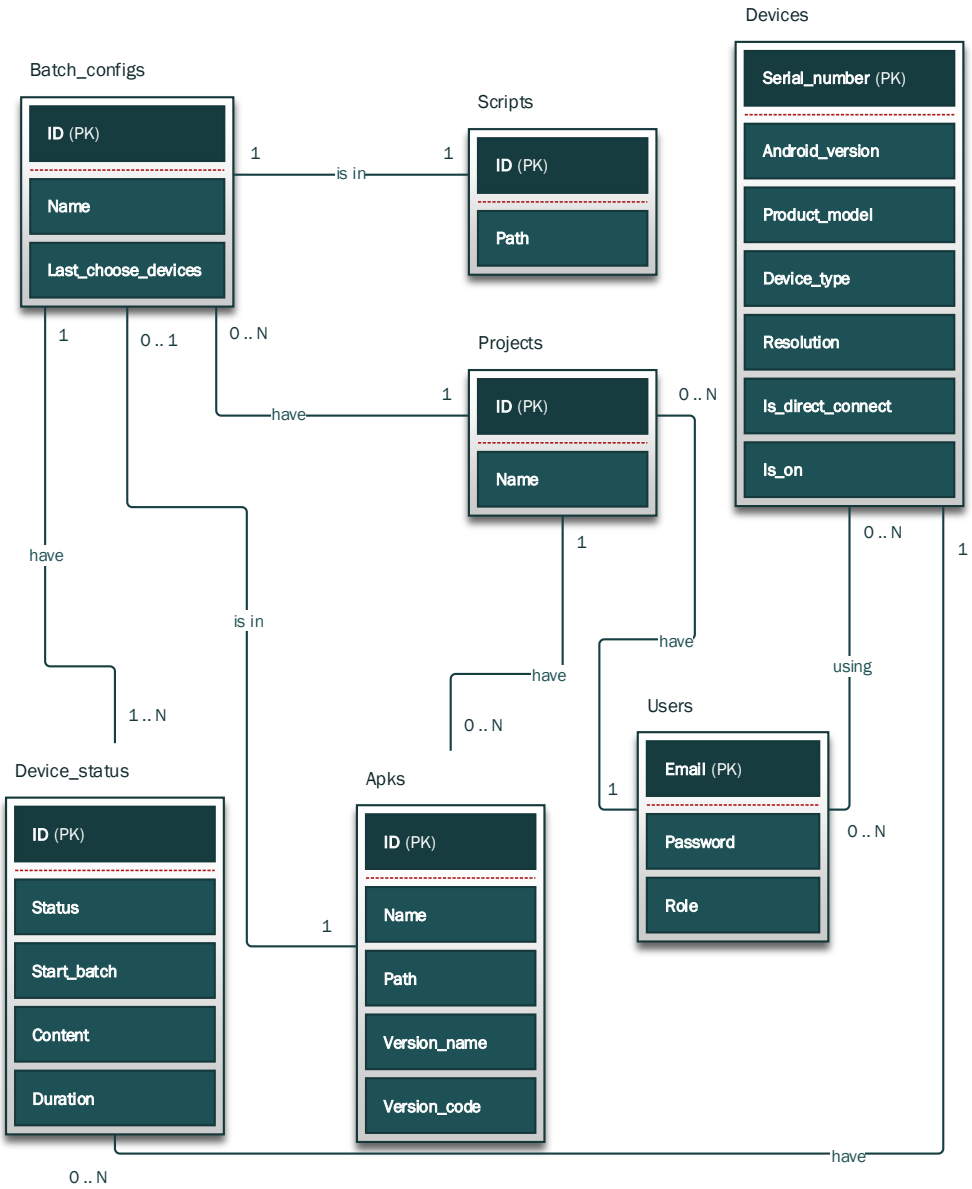
Diagram případu užití navrženého systému je znázorněn na obrázku 4.3. V systému se vyskytuje administrátor, nezaregistrovaný a registrovaný uživatel. Nezaregistrovaný uživatel se může pouze registrovat a prohlížet úvodní obrazovku, která bude obsahovat informativní video k výslednému dílu. Jiné operace mu nebudou povoleny. Zaregistrovaný uživatel si může po úspěšném přihlášení vytvořit vlastní projekt. V rámci tohoto projektu může vlastník přidat libovolný počet svých Android aplikací. Dále si vlastník může vytvořit libovolný počet hromadných testů v rámci jednoho projektu. Jeden hromadný test může obsahovat právě jednu Android aplikaci, kterou chce vlastník otestovat na více zařízeních. Pokud uživatel vytvoří alespoň jeden hromadný test v rámci jednoho projektu a bude-li chtít pokračovat z hlavního okna projektu k výběru zařízení, pak se bude muset rozhodnout, zdali chce testovat právě jedno zařízení nebo využít hromadný test. Po úspěšném výběru uživatelovi akce (jednoduché otestování aplikace nebo hromadný test) bude přesměrován na výběr zařízení (zobrazí se pouze ty zařízení, které nejsou obsazeny). Po výběru zařízení bude již moci testovat svoji aplikaci na reálných zařízeních. V rámci jednoduchého otestování aplikace bude uživatel moci vzdáleně ovládat Android zařízení. Ovládáním zařízení jsou myšleny doteky (krátké i dlouhé) obrazovky, tahy prstu po obrazovce a klávesové vstupy. Bude mu umožněna filtrace záznamu podle různých kriterií (priorit, tagu). Dále bude informován o vytíženosti paměti a procesoru spuštěnou aplikací. U hromadných testů je nutné podotknout, že budou prováděny pomocí testovacího frameworku Appium. To znamená, že si vlastník hromadného testu bude muset vytvořit vlastní testovací skript, který bude testovat jeho aplikaci. Tento test bude současně proveden na vybraných zařízeních (pokud je zařízení používané, jestli ne bude test zařazen do fronty). Poté uživatel obdrží informace o provedení daných testů ve formě záznamu a informaci o úspěšném či neúspěšném provedení. Administrátor bude disponovat veškerými funkcemi ke správnému chodu systému a sice smazaní uživatelů, editace uživatelů apod.



Obrázek 4.3: Diagram případů užití.

4.3 Entity-relationship diagram

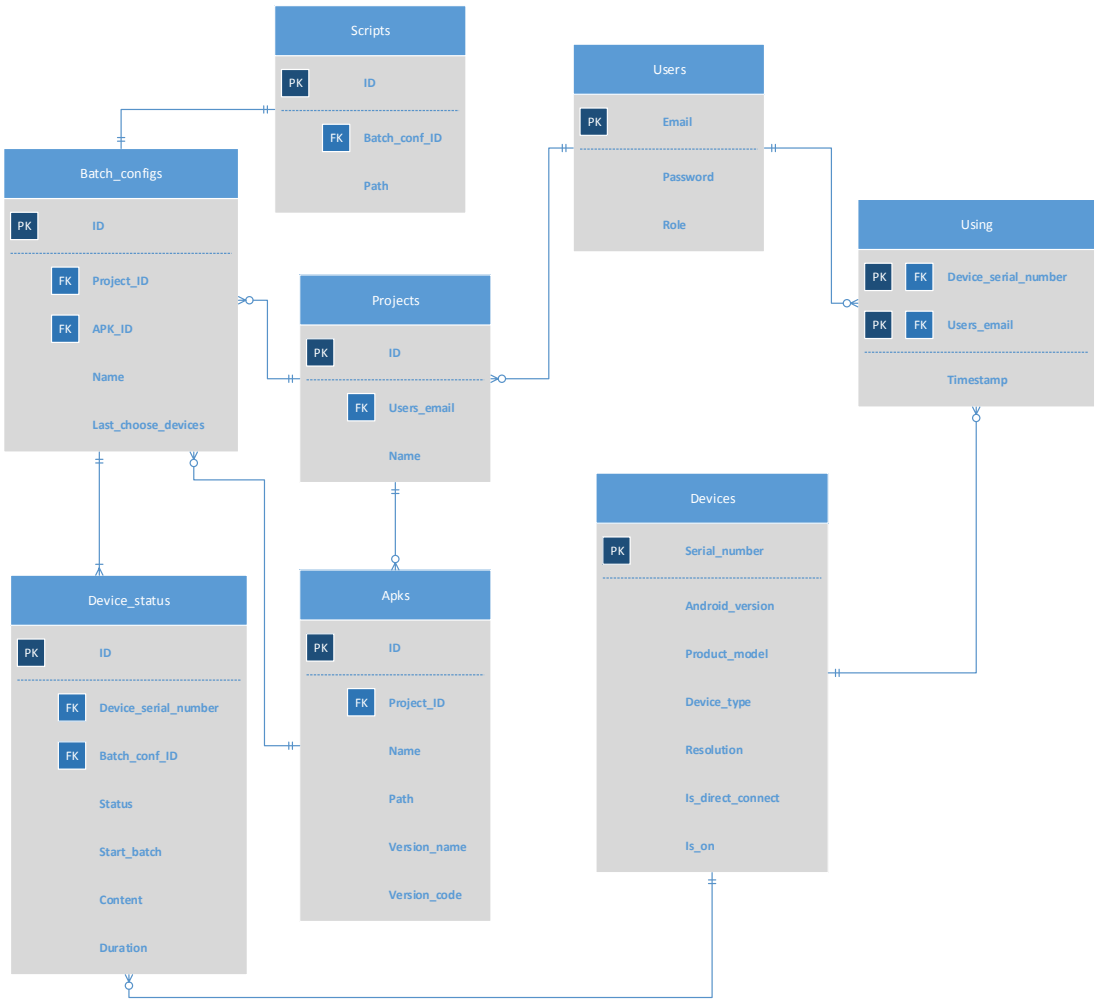
Ze sekce 4.2 lze určit klíčové entity výsledného systému. Tyto entity a vztahy jsou zobrazeny na obrázku 4.4. Mezi tři hlavní entity patří **Users, Projects, Devices**. Dále jsou na vstupních a výstupních koncích hran uvedeny násobnosti vyjadřující počet instancí daných entit. Z tohoto diagramu vychází databázové schéma v následující sekci 4.4.



Obrázek 4.4: ER diagram výsledného systému.

4.4 Databázové schéma

Na základě Entity-relationship diagramu z předchozí sekce byl vytvořen návrh struktury výsledné databáze, která je zobrazena na obrázku 4.5. Popsané vztahy mezi jednotlivými entitami využívají Crow's foot[3] notaci. Oproti předchozímu diagramu bylo nutné odstranit M:N vztah mezi entitami **Users** a **Devices**, který se odstranil přidáním další tabulky **Using**. U všech entit jsou již zobrazeny potřebné atributy, není však vyloučeno, že se tyto atributy budou v průběhu implementace měnit.



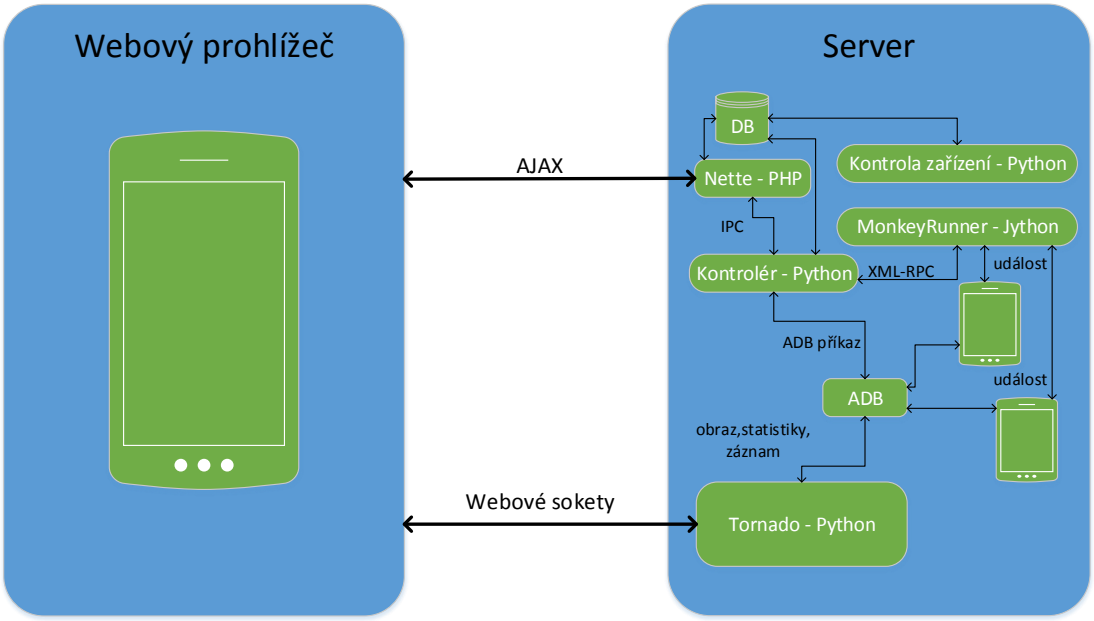
Obrázek 4.5: Databázové schéma výsledného systému.

[3]Notace Crow's Foot znázorňuje vztahy propojováním entit čarami, přičemž symboly na obou koncích čáry symbolizují kardinalitu vztahu.

4.5 Návrh architektury

Tato sekce popisuje základní propojení jednotlivých komponent ke komunikaci se vzdáleným zařízením na serveru. Níže uvedený obrázek 4.6 vystihuje komunikaci se vzdáleným zařízením.

Pro zachycení obrazovky, záznamu a statistik zařízení v reálném čase bude použita technologie webových soketů. Na serveru bude tedy nutné vytvořit server, který tuto technologii podporuje. Jak již bylo zmíněné v sekci 2.3.2, bude se jednat o Tornado server naprogramovaný v jazyce Python. Dále bude nutné zachytit uživatelovu interakci se vzdáleným zařízením. K tomuto účelu slouží hlavní kontrolér (provádí také jiné operace), který rozpozná uživatelovu akci a podle toho vykoná potřebnou operaci. Pokud se bude jednat o události, které souvisí s dotykem obrazovky, pak je nutné komunikovat skrze MonkeyRunner, protože samotné ADB podporuje tyto operace až od určité verze. Hlavní kontrolér s MonkeyRunnerem komunikuje pomocí XML-RPC. Samotný kontrolér bude také řídit hromadné testování aplikací. Dále bude na serveru běžet skript, který bude kontrolovat připojené zařízení. Pokud se přípojí nějaké nové zařízení skrze USB[4] ke vzdálenému serveru, tento skript musí rozpoznat dané zařízení a přidat ho do databáze. Naopak, pokud se odpojí nějaké zařízení od vzdáleného serveru, skript musí oznámit tuto událost databázi.

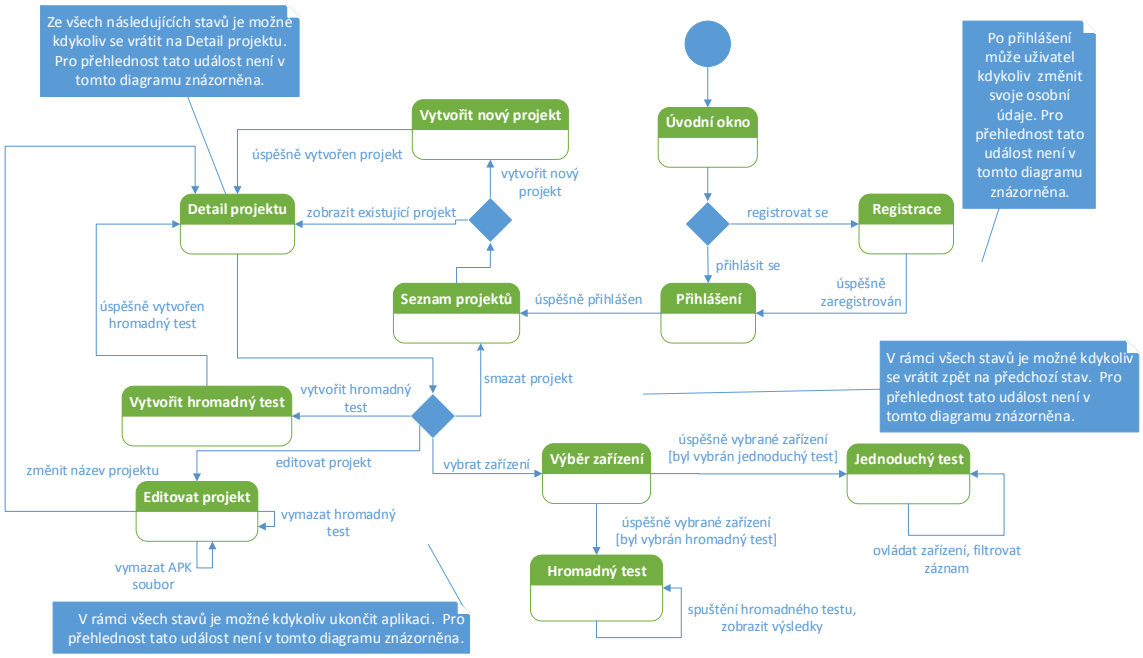


Obrázek 4.6: Komunikace se vzdáleným zařízením.

[4]Universal Serial Bus.

4.6 Stavový diagram obrazovek

Na obrázku 4.7 je znázorněna navigace skrze jednotlivá okna v rámci celého informačního systému. Diagram slouží pro základní představu výsledného systému. V rámci implementace se předpokládá se změnami. Diagram zachycuje hlavní okna, na které se registrovaný uživatel může dostat. Okno v diagramu odpovídá jednomu stavu. Na hranách jsou znázorněny uživatelovi akce, které lze z daného okna provést. Dále jsou v diagramu zobrazeny rozhodovací uzly, které jsou znázorněny modrými kosočtverci. Tyto uzly slouží pro vyjádření větvení na základě volby z více možností, které může uživatel provést.



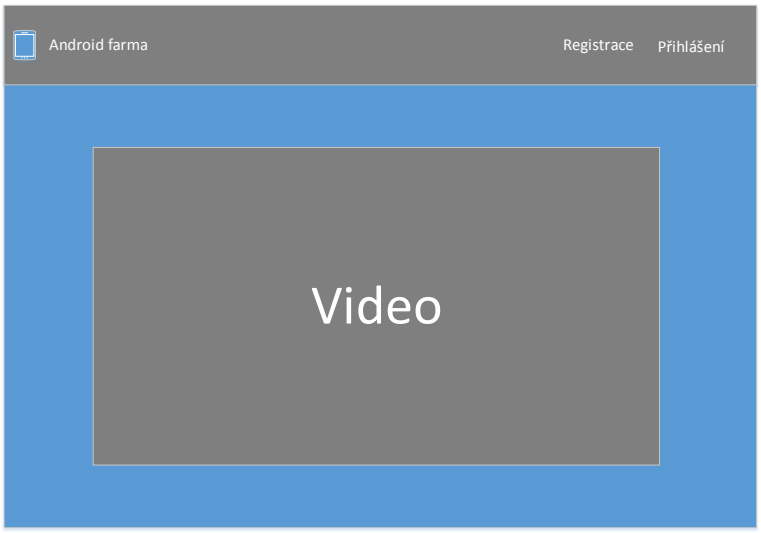
Obrázek 4.7: Stavový diagram návazností jednotlivých obrazovek.

Následující sekce 4.7 bude popisovat grafický návrh hlavních oken, která byla zmíněna v diagramu na obrázku 4.7.

4.7 Grafický návrh

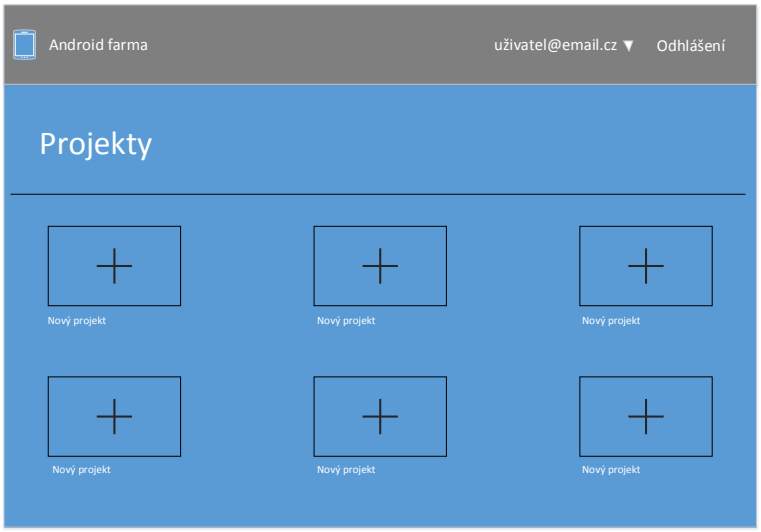
Veškeré grafické návrhy v této sekci slouží pouze k základní orientaci, jak by měl výsledný informační systém vypadat. V rámci implementace jsou předpokládány určité změny.

Ze sekce 4.6 jsou již známy možnosti interakce uživatele v rámci jednotlivých oken. V rámci úvodního okna, které je znázorněno na obrázku 4.8, se bude moci uživatel pouze přihlásit, popřípadě zaregistrovat. Bude zde také uvedeno motivační video, které by mělo zaujmout nového uživatele.



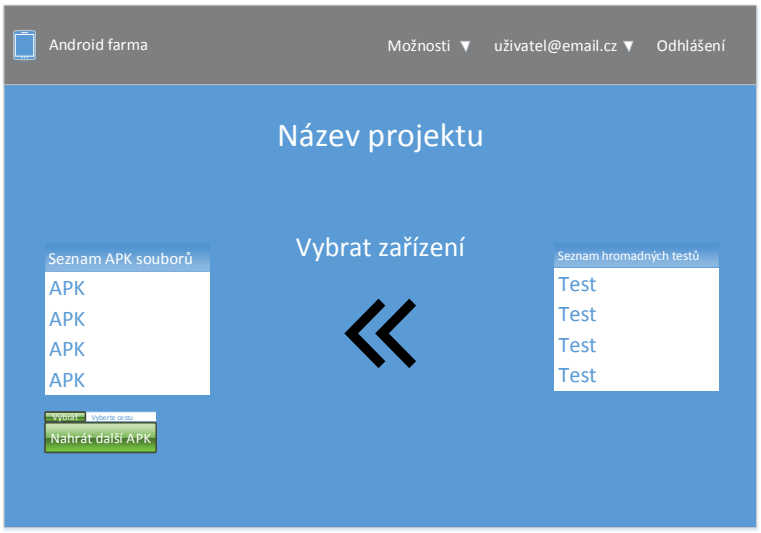
Obrázek 4.8: Úvodní okno.

Po úspěšném přihlášení bude uživatel moci procházet svoje jednotlivé projekty, popřípadě vytvořit nový projekt. Ukázku seznamu projektů zachycuje obrázek 4.9.



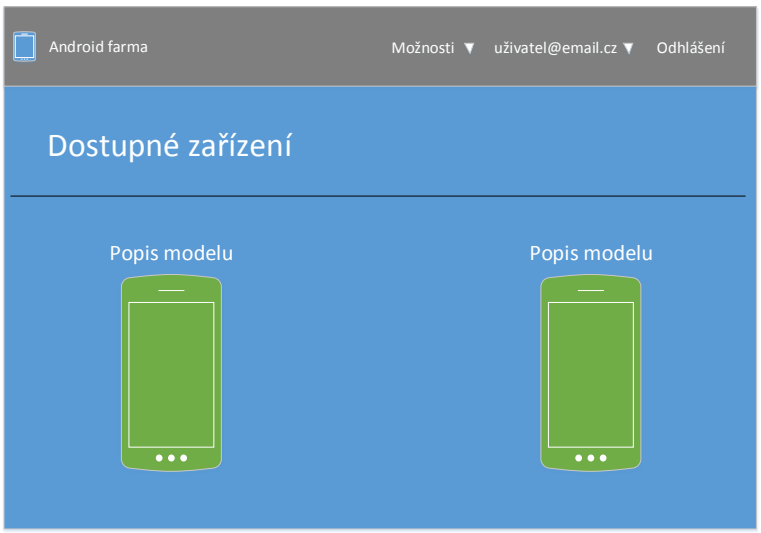
Obrázek 4.9: Seznam uživatelových projektů.

Detail jednoho projektu, který je znázorněn na obrázku 4.10, by měl obsahovat jednotlivé APK soubory nahrané uživatelem a již vytvořené hromadné testy. Uvedená šipka by měla ukazovat aktuální výběr uživatele. Pokud uživatel vybere jeden z APK souborů ze svého listu a bude chtít s tímto souborem pokračovat, bere se tato událost jako testování na jednom zařízení.



Obrázek 4.10: Detail projektu.

Po úspěšném výběru uživatelovi akce (jednoduché otestování aplikace nebo hromadný test) následuje výběr dostupného zařízení k provedení dané akce. V rámci jednoduchého otestování aplikace může uživatel ovládat pouze jedno zařízení. Grafický návrh výběru dostupných zařízení je znázorněn na obrázku 4.11.



Obrázek 4.11: Dostupné zařízení.

Po úspěšném výběru dostupného zařízení následuje buď jednoduché otestování aplikace nebo hromadný test (záleží na tom co uživatel požaduje). V rámci jednoduchého otestování aplikace bude moci uživatel ovládat zařízení v reálném čase. Také by měl vidět aktuální záznam zařízení a jeho vytížení (pravděpodobně procesoru a paměti). Ukázka grafického rozložení jednotlivých komponent je zobrazena na obrázku 4.12.

Obrázek 4.12: Ovládání jednoho zařízení.

V rámci hromadného testu by měl uživatel specifikovat skript, podle kterého se bude testovat jeho aplikace. Po úspěšném vykonání testu bude zobrazen status (úspěch, neúspěch, apod.) k jednotlivým zařízením. Grafické rozložení hromadného testování je zobrazeno na obrázku 4.13.

Obrázek 4.13: Hromadný test.

4.8 Analýza Android Debug Bridge protokolu

Tato sekce popisuje malou, ale klíčovou součást protokolu ADB, a sice USB komunikaci mezi hostitelem a zařízením. Související zdrojové kódy s ADB jsou umístěny ve složce **system/core/adb** [2]. V této složce se také nachází dokumentační soubory, a sice overview.txt, services.txt, protocol.txt a sync.txt.

- overview.txt - Obecné informace o ADB architektuře, které byly popsány na začátku této práce.
- protocol.txt - Zde je popsán výsledný protokol, struktura paketu apod.
- services.txt - V tomto souboru jsou zdokumentovány všechny požadavky ADB klienta, které může učinit vůči ADB serveru.
- sync.txt - Zejména popisuje protokol pro přenos souborů (push, pull příkazy pomocí ADB).

Bez ohledu na typ připojení (USB nebo TCP) je komunikační protokol mezi hostitelským serverem a démonem na zařízení pořád stejný. Jedná se o paketovou komunikaci, což znamená, že data, jsou rozdělena do jednotlivých paketů. Paket se skládá ze 2 části, a to z 24 bajtové hlavičky a dat. Nyní bude popsána struktura ADB zprávy a ukázka několika typů standardních paketů.

Ukázka třídy ADB zprávy v jazyce C++:

```
#define MAX_PAYLOAD 4096
class ADBMessage {
public:
    unsigned command;       // identifikace příkazu, např. typ A_OPEN
    unsigned arg0;          // první argument
    unsigned arg1;          // druhý argument
    unsigned data_length;   // velikost dat
    unsigned data_check;    // kontrolní součet
    unsigned char data[MAX_PAYLOAD]; //samotné data

public:
    ADBMessage();
    ADBMessage(unsigned command); //např. pro zprávu reboot(bez dat)
    ADBMessage(unsigned command, unsigned arg0,
               unsigned arg1, const char *data);
    ADBMessage(unsigned command, unsigned arg0,
               unsigned arg1, const char *data, unsigned data_length);
    ~ADBMessage();

    void SetCommand(unsigned command);
    void SetRemoteID(unsigned remoteID);
    void SetData(const char *data, unsigned data_length);
};
```

ADB definuje 6 základních typů paketů, a sice A_SYNC, A_CNXN, A_OPEN, A_OKAY, A_CLSE, A_WRTE, A_AUTH. Každý tento paket má svoji identifikační hodnotu, podle které je zpráva rozeznána.

- OPEN - Používá se k otevření komunikačního kanálu.
- CLOSE - Slouží k uzavření komunikačního kanálu.
- WRITE - Zpráva, která se používá k přenosu dat.
- OKAY - Slouží pro potvrzování paketů.
- SYNC - Zpráva, která se používá k synchronizaci.
- AUTH - Tato zpráva se používá k autentizaci.

Ukázka posloupnosti zpráv pro odeslání shell příkazu **ls** (zobrazit soubory v dané složce) na daném zařízení.

```
ADBMessage(A_OPEN, local_id, 0, "shell:ls"); //server
ADBMessage(A_OKAY, remote_id, local_id, NULL); //démon
ADBMessage(A_WRTE, remote_id, local_id, data); //démon
ADBMessage(A_OKAY, local_id, remote_id, NULL); //server
ADBMessage(A_CLSE, local_id, remote_id, NULL); //server
ADBMessage(A_CLSE, remote_id, local_id, NULL); //démon
```

Kapitola 5

Implementace navrženého systému

Tato kapitola popisuje implementaci výsledného systému. Nejprve je popsána komunikační část v Pythonu. Poté je vysvětlena vícevrstvá architektura v Nette Frameworku, která byla napojena na tyto skripty. V závěru kapitoly se zabývám testováním výsledného systému.

5.1 Komunikační část

Jak již bylo zmíněno, back-endová část výsledného systému byla vytvořena pomocí Nette Frameworku a Python skriptů. Tato sekce se zabývá skripty, které byly vytvořeny v jazyce Python.

5.1.1 Kontrolér

Základem veškeré komunikace je skript **controller.py**. Tento skript jako jediný komunikuje s Nette Frameworkem. Komunikace probíhá pomocí IPC soketu. Což znamená, že místo toho, aby identifikace serveru byla podle IP adresy a portu, tak se server identifikuje cestou k souboru. Je zřejmé, že klient (Nette) a server (kontrolér) musí mít stejnou cestu k danému souboru, pokud chtějí komunikovat. Rychlost komunikace mezi Nette Frameworkem a kontrolérem by tedy měla být rychlejší (podle dostupných testů až dvakrát) než oproti klasickému loopback soketu pomocí TCP.

Jedná se o konkurentní server, což znamená, že dokáže obsloužit více klientů zároveň. Konkurentnost tohoto serveru je zajištěna pomocí nekonečné smyčky tím, že pokud se k tomuto serveru připojí další uživatel, je vytvořen nový proces, který bude obsluhovat právě připojeného uživatele.

```
while True:
    print "Waiting for another connection."
    conn, address = self.socket.accept()
    self.logger.debug("New connection")
    process = multiprocessing.Process(target=handle, args=(conn, address))
    process.daemon = True
    process.start()
    self.logger.debug("Started process %r", process)
```

Vytvoření nového procesu je zajištěno pomocí modulu **multiprocessing**. Tento proces je zpracován obslužnou funkci **handle**. Tato funkce již zpracovává uživatelův požadavek. Požadavky jsou založeny na různých typech zpráv. Podle dané zprávy kontrolér vykoná

příslušné operace. Kontrolér rozlišuje několik druhů zpráv. Nyní budou popsány zprávy k samotnému spuštění uživatelovi aplikace a následné ovládání daného zařízení.

- **init_device** - V rámci této zprávy kontrolér dostane informaci o sériovém číslu zařízení, se kterým bude komunikovat a cestě k aplikaci, kterou chce uživatel testovat. Po přijetí této zprávy kontrolér vykoná několik operací. Jako první vykoná vzdálenou inicializační metodu v skriptu **monkey-runner-server.py** pomocí XML-RPC. Tato operace bude podrobněji popsána při popisu samotného skriptu, který obsluhuje MonkeyRunner. Dále je nutné probudit zařízení pomocí aplikace, která je skrze ADB do tohoto mobilu nainstalována. Probuzení zařízení bylo nutné řešit pomocí vlastní aplikace, protože pomocí ADB nebylo možné tuto operaci provést spolehlivě. Po úspěšném probuzení kontrolér nainstaluje aplikaci, kterou si uživatel zvolil k testování do daného zařízení. Dále jsou do zařízení nainstalovány některé podpůrné aplikace (změna landscape a portrait pohledu), protože samotné ADB tyto operace neumožňuje.

- **install_apk** - V této zprávě dostane kontrolér informaci o sériovém číslu zařízení a cestě k danému APK souboru. Tyto informace použije k tomu, aby správně nainstaloval aplikaci do daného zařízení.

- **unlock_device** - Slouží k odemčení zařízení s daným sériovým číslem.

- **lock_device** - Tato zpráva slouží k uzamčení zařízení s daným sériovým číslem.

- **valid_apk** - V rámci této zprávy kontrolér obdrží cesty dvou různých aplikací. Poté kontrolér vyhodnotí, zdali tyto dvě aplikace mají stejný název balíčků[1]. Vyhodnocení probíhá pomocí nástroje aapt, který byl popsán na začátku této práce.

- **sleep_device** - Zpráva **sleep_device** slouží k navrácení zařízení do původního stavu. Kontrolér obdrží informaci o sériovém číslu zařízení, dále o všech aplikacích, které byly uživatelem do mobilu nainstalovány. Díky těmto informacím kontrolér odinstaluje všechny aplikace, které uživatel nainstaloval. Dále ukončí komunikaci se skriptem **monkey-runner-server.py**.

- **touch_screen** - Slouží k obsluze dotyku obrazovky. S touto zprávou kontrolér obdrží hodnoty souřadnic dotyku **x** a **y**. Kontrolér poté vykoná vzdálenou metodu **touch** s parametry **x** a **y** ve skriptu **monkey-runner-server.py**.

- **long_touch_screen** - Tato zpráva slouží k obsluze dlouhého dotyku obrazovky. S touto zprávou kontrolér obdrží hodnoty souřadnic dotyku **x** a **y**. Kontrolér poté vykoná vzdálenou metodu **long_touch** s parametry **x** a **y** ve skriptu **monkey-runner-server.py**.

- **drag_screen** - Další zpráva **drag_screen** slouží k simulaci tahu prstu po obrazovce. S touto zprávou kontrolér obdrží hodnoty souřadnic **x** a **y** dvou bodů (počáteční a koncový). Kontrolér poté vykoná vzdálenou metodu **drag** s parametry souřadnic daných bodů ve skriptu **monkey-runner-server.py**.

- **set_landscape, set_portrait** - Tyto zprávy slouží k přepnutí landscape a portrait pohledů na daném zařízením. Tyto změny se provádí pomocí aplikací, které byly nainstalovány do zařízení v rámci zprávy **init_device**.

[1] Název balíčku se nastavuje v Android manifestu dané aplikace.

- **check_apk** - S touto zprávou kontrolér obdrží pouze cestu k aplikaci. Poté určí zda se skutečně jedná o Android aplikaci či nikoliv. Toto ověření probíhá opět pomocí nástroje aapt.

- **send_keyevent** - Poslední zpráva **send_keyevent** slouží k simulaci stisku klávesy na klávesnici. S touto zprávou kontrolér obdrží číselnou hodnotu (převedena z ASCII do Android konstant) dané klávesy. Kontrolér poté vykoná vzdálenou metodu **press**, kde jako parametr použije získanou číselnou hodnotu, ve skriptu **monkey-runner-server.py**.

Kontrolér zpracovává ještě dvě zprávy, a sice **open_ports** a **run_batch**. Tyto zprávy slouží k hromadnému testování aplikací.

- **run_batch** - Jak již bylo zmíněno v sekci 3.3.4, architektura frameworku Appium se skládá z klienta a serveru. Kontrolér proto musí řídit, jak serverovou tak i klientskou část. Pro každé testované zařízení je vytvořena nová instance serveru Appium z důvodu paralelního zpracování. Každá nová instance potřebuje definovat tři různé porty pro komunikaci. Jedná se o klasický port, na kterém naslouchá Appium, bootstrap port, který slouží ke komunikaci s Android zařízením a selendroid port ke komunikaci se Selendroidem. Proč je použita komunikace skrze Selendroid bylo vysvětleno v sekci 3.3.4. Zmíněné porty se získávají pomocí zprávy **open_ports**. Po úspěšném vytvoření instance serveru je spuštěn klientský testovací skript. Výsledek daného skriptu je po provedení uložen do databáze. Poté je ukončena instance Appium serveru. Samotnému spuštění instance Appium serveru a daného klientského skriptu předchází testování volného zařízení a času spuštěného skriptu. To znamená, že skripty jsou vykonávány, pouze pokud je volné dané testovací zařízení. Pokud dané testovací zařízení je volné, přichází na řadu druhý parametr, a sice v jakém časovém pořadí byly skripty spuštěny.

5.1.2 MonkeyRunner

Samotné ovládání zařízení je řešeno pomocí skriptu **monkey-runner-server.py**. Jedná se o konkurentní XML-RPC server, což znamená, že dokáže obsloužit více klientů zároveň. Server naslouchá na adrese **localhost:8000**. Konkurentnost tohoto serveru je zajištěna pomocí třídy **RPCThreading**, a to tak, že s každým novým požadavkem je vytvořeno nové vlákno, které obsluhuje daný požadavek. V rámci tohoto skriptu je vytvořena jedna třída **Devices**, která obsahuje jednotlivé metody k ovládání zařízení skrze MonkeyRunner, který byl popsán v sekci 3.1.2. Jedná se o metody **init**, **touch**, **long_touch**, **press**, **drag**, **destroy**.

- **init** - Slouží k vytvoření nového objektu **MonkeyDevice**. Tento objekt je vytvořen pomocí metody *MonkeyRunner.waitForConnection()* s parametrem sériového čísla zařízení, se kterým bude navázána komunikace.

- **touch** - Přijímá souřadnice bodu dotyku **x** a **y**. Tyto body dotyku jsou dále použity jako parametry metody *MonkeyDevice.touch()*. Metoda *MonkeyDevice.touch()* přijímá ještě další parametr, a sice událost, která má být vykonána. Jako událost je použita *MonkeyDevice.DOWN_AND_UP*.

- **long_touch** - Metoda **long_touch** přijímá souřadnice body dotyku **x** a **y** stejně jako předchozí metoda. Po úspěšném přijetí těchto bodů je vykonána opět metoda *MonkeyDevice.touch()*, ale s jinou událostí, a sice *MonkeyDevice.DOWN*.

- **press** - Přijímá hodnotu klávesy (jak již bylo zmíněno, jedná se o hodnotu převedou z ASCII do Android konstant), která byla stisknuta. Následuje vykonání metody *MonkeyDevice.press()*. Jeden z parametru této metody je hodnota stisknuté klávesy. Dalším parametrem je událost *MonkeyDevice.DOWN_AND_UP*.
- **drag** - Tah prstu po obrazovce je obsloužen metodou **drag**. Metoda přijímá souřadnice dvou bodů (počáteční a koncový). Tyto body jsou použity jako parametry metody *MonkeyDevice.drag()*.
- **destroy** - Metoda **destroy** slouží k odstranění vytvořeného objektu **MonkeyDevice**. Dále je nutné ukončit spojení se samotným zařízením.

5.1.3 Servery založené na webových soketech

Pro zpracování informací v reálném čase (obrazu, statistik vytíženosti, záznamu daného zařízení) byla použita již zmíněna technologie webových soketů. Pro implementaci této technologie byl využit webový server **Tornado**[2].

Byly vytvořeny tři různé skripty **tornado-server-img.py**, **tornado-server-logcat.py**, **tornado-server-stats.py**. Z názvu skriptu je zřejmé, že každý skript zastupuje jinou funkcionalitu. Každý z těchto skriptů je implementován pomocí webového serveru Tornado. Všechny tři skripty tedy mají základní strukturu stejnou.

```
class WebSocketHandler(tornado.websocket.WebSocketHandler):
    def open(self):
        print "Připojen nový klient."

    def on_message(self, message):
        print "Přišla nová zpráva od klienta."

    def on_close(self):
        print "Ukončení spojení s klientem."
```

Poslat zprávu zpět ke klientovi je možné pomocí metody *self.write_message()*, která patří třídě **WebSocketHandler**.

K těmto serverům se připojuje klient pomocí JavaScriptu, pokud přistoupí k testování jedné aplikace na daném zařízením. Díky tomuto připojení bude klient dostávat periodicky informace o statistikách vytíženosti, obrazu a záznamu daného zařízení. Periodické odesílání je zajištěno pomoci třídy *PeriodicCallback(callback, callback_time)*, která je součástí Tornado serveru.

Zpracování statistik

Aktuální statistiky ze zařízení jsou zpracovávány skriptem **tornado-server-stats.py**. Zisk samotných statistik je prováděn pomocí nástroje **dumpsys**, který je spouštěn pomocí shellu skrze ADB. Jedná se o nástroj, který běží na zařízení a poskytuje zajímavé informace (vytíženost procesoru, paměti, baterie apod.) o stavu systémových služeb. Tento skript vyfiltruje informace o vytíženosti procesoru a paměti zařízení a přepošle je zpět klientovi. Je zde otevřen prostor pro další rozšíření zpracování statistik.

[2] http://www.tornadoweb.org/en/branch2.1/websocket.html

Zpracování záznamu

Zpracování aktuálního záznamu je řešeno pomocí skriptu **tornado-server-logcat.py**. Zisk samotného záznamu je prováděn pomocí příkazu **adb logcat** skrze ADB. Tento záznam je skriptem filtrován podle dané priority (implicitně je nastavena nejnižší možná priorita[3]), kterou si klient zvolí. Jako další filtrovací pravidlo je použito PID[4] procesu aplikace, kterou si klient spustí k testování. Filtrování podle aplikace je možné zrušit, což ovšem znepřehlední výsledný záznam. Vyfiltrovaný záznam je odeslán zpět ke klientovi.

Zpracování obrazu

Zisk obrazu ze zařízení má na starosti skript **tornado-server-img.py**. Aktuální obraz ze zařízení je momentálně získáván pomocí ADB příkazu **screencap**. Rychlost tohoto přístupu není až tak plynulá. Ovšem po dohodě s vedoucím práce jsme dospěli k názoru, že tato rychlost je pro tuto práci momentálně dostačující s možnosti optimalizačního vývoje. Optimalizační vývoj spočívá v nativním přístupu k danému zařízení. Nativní přístup k danému zařízeni je možný díky Android NDK, které bylo zmíněno v sekci 3.2.

5.1.4 Kontrola zařízení

Tento skript má za úkol sledovat stav zařízení připojených k serveru. Skript musí jednak rozpoznat nově připojené zařízení, ale také musí sledovat již připojená zařízení, zdali nebyly odpojeny. Tyto kontroly jsou prováděny ve skriptu **check-devices.py**.

Kontrola je zajištěna pomocí **GUDevMonitorObserver** objektu. Tento objekt monitoruje události (odpojení či připojení zařízení z USB), na základě které skript vykoná odpovídající operace (jedná se především o databázové operace).

5.2 Vícevrstvá architektura

V implementaci systému byla dodržena již zmíněna architektura MVP. Celý systém v Nette Frameworku je rozdělen do dvou části, a to do uživatelského a administrátorského modulu. Oba dva moduly používají stejné databázové modely. Pokud tedy dojde k nějaké změně v databázi, stačí změnit pouze tyto modely. Popis uživatelského modulu bez Latte šablon nejlépe znázorňuje diagram tříd na obrázku B.1. Administrátorský modul je rozšířen o další funkcionalitu, a sice o správu zařízení a uživatelů. Pro přehlednost v presenter a view vrstvě budu popisovat pouze uživatelský modul.

5.2.1 Modelová vrstva

Tato vrstva zajišťuje přístup k datům a manipulaci s nimi skrze metody obsahující dotazy, které jsou prováděny nad databází MySQL. Pro vytvoření nového připojení k databázi stačí vytvořit novou instanci třídy *Nette\Database\Connection*. Nejjednodušším způsobem, jak vytvořit toto připojení, je nechat si ho vytvořit samotným Nette Frameworkem. A to tak, že v aplikační konfiguraci (soubor config.neon[5]) toto připojení definujeme.

[3] Android záznam se dá filtrovat podle priorit, kvůli jeho čitelnosti. Prioritou je zde myšlena váha (warning, debug, error atd.) dané zprávy.

[4] Identifikace procesu v operačním systému.

[5] Pomocí config.neon se konfiguruje systémový kontejner, což je kontejner, ve kterém se nacházejí všechny služby a parametry potřebné pro běh aplikace.

```
database:
        default:
            dsn: 'mysql:host=127.0.0.1;dbname=login'
            user: username
            password: password
```

Všechny modelové třídy kromě **UserAuthenticator.php** a **BaseDatabase.php**, která slouží jako základ pro všechny ostatní třídy, obsluhují jednotlivé tabulky v databázi. Jedná se o tabulky, které byly navrhnuty v sekci 4.4.

- **BaseDatabase.php** - Slouží jako základ pro všechny ostatní modelové třídy. Obsahuje základní společné metody pro všechny ostatní třídy. Tyto metody jsou ukázány na obrázku B.1 diagramu tříd. Samotný model **BaseDatabase.php** dědí třídu *\Nette\Object*, která je společným předkem všech tříd Nette frameworku. Všechny níže uvedené modely dědí od tohoto modelu.
- **Apks.php** - Model obsluhuje jednotlivé balíčky APK, se kterými uživatelé pracují.
- **BatchConfigs.php** - Pracuje s jednotlivými hromadnými testy, které uživatel používá k automatizovanému hromadnému testování.
- **DeviceStatus.php** - Slouží pro práci s jednotlivými výsledky hromadného testu.
- **Devices.php** - Implementuje metody, které slouží k práci s jednotlivými zařízeními.
- **Projects.php** - Tento model byl vytvořen pro obsluhu jednotlivých uživatelských projektů.
- **Scripts.php** - Pro jednotlivé ukládání a nahrávání vytvořených uživatelských testovacích skriptů slouží model **Scripts.php**.
- **Users.php** - Implementuje jednotlivé metody k práci se samotnými uživateli.
- **Using.php** - Předposlední model **Using.php** slouží k získání informací ohledně aktuálně dostupných (nejsou momentálně používány) zařízeních.
- **UserAuthenticator.php** - Poslední model slouží k autentizaci uživatele. Model je implementován rozhraním *Nette\Security\IAuthenticator*, které je určeno právě k autentizaci uživatele.

5.2.2 Presenter vrsta

Presenter vrstva aplikace je tvořena jednotlivými presentery, které jsou znázorněny na obrázku B.1 diagramu tříd. Presenter je základní třída Nette a jedná se v podstatě o řadič, který řídí nějakou část webové aplikace. Jeho úlohou je reagovat na události pocházející od uživatele a zajišťovat změny v modelu nebo v zobrazení uživatelského rozhraní, které zajišťuje view vrstva.

V Nette se aplikace řídí následujícím způsobem. Nejprve uživatel zašle aplikaci požadavek skrze webový prohlížeč, který směruje na konkrétní URL. Tento požadavek Nette Framework pomocí routování zanalyzuje a najde příslušný presenter, který spustí a předá mu řízení. Presenter komunikuje s modelem, kterému sdělí, jaká data požaduje. Model tyto

data načte a předá zpět presenteru. Tato data presenter předá view vrstvě, která se stará o jejich zobrazení. Celý tento proces se opakuje s dalším uživatelovým požadavkem.

Samotný životní cyklus presenteru byl vysvětlen v sekci 2.2.2. Z obrázku B.1 diagramu tříd je patrné, že nejčastější používanou metodou je metoda **handle<Signál>**, která zpracovává AJAXové požadavky.

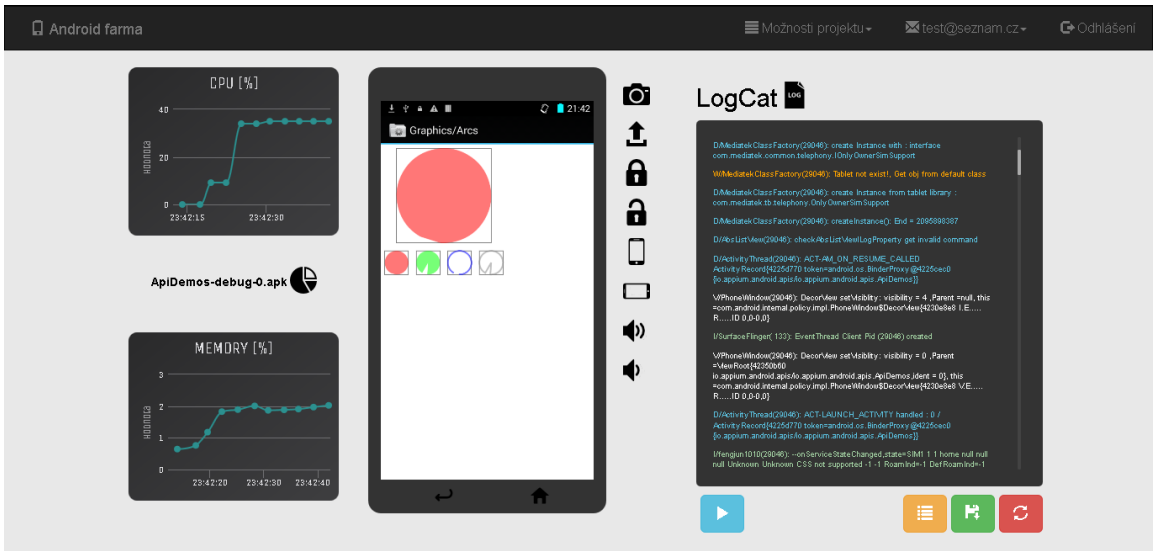
- **BasePresenter.php** - Tato abstraktní třída je základem celého uživatelského modulu presenter vrstvy. Obsahuje všechny instance databázových modelů, které byly získány pomocí anotace *@inject*, která byla popsána v sekci 2.2.2. Dále obsahuje metody, které slouží ke komunikaci s již zmíněným kontrolérem pomocí IPC soketu. Tato třída dědí od abstraktní třídy *Nette\Application\UI\Presenter*. Všechny níže uvedené presentery dědí od abstraktní třídy **BasePresenter.php**.
- **BatchConfigPresenter.php** - Řídí samotné hromadné testování aplikací. Dále obsahuje metody k uložení a nahraní uživatelského skriptu. Samotné vykonávání hromadného testu se provádí v kontroléru, proto může uživatel po spuštění hromadného testu tuto webovou stránku opustit.
- **ChooseDevicePresenter.php** - Tento presenter řídí výběr zařízení, které si uživatel vybírá pro svoje testování.
- **DetailProjectPresenter.php** - Především pracuje se samotným projektem. Obstarává také jednotlivé úpravy, a sice mazaní Android aplikací, hromadných testů apod. Také zjišťuje jestli je momentálně k dispozici nějaké zařízení, se kterým by mohl uživatel pracovat.
- **EditUserPresenter.php** - Jedná se o presenter, který slouží k editaci uživatele.
- **ErrorPresenter.php** - Slouží pro řízení chybových kódů 403, 404 atd.
- **HomepagePresenter.php** - Pracuje s dostupnými projekty, které si uživatel vytvořil.
- **NewProjectPresenter.php** - Presenter slouží k vytváření nových projektů.
- **NewTestPresenter.php** - Řídí vytváření nových hromadných testů.
- **RunSingleTestPresenter.php** - Tento presenter řídí vzdálené ovládání jednoho zařízení. Obsahuje několik metod (dotyk obrazovky, stisk klávesy apod.), které slouží k samotnému ovládání. Všechny tyto metody jsou řešeny pomoci AJAXových požadavků a jsou přeposílány do kontroléru.
- **SignInPresenter.php** - Jedná se o presenter, který řídí přihlašování a autentizaci uživatele.
- **SignUpPresenter.php** - Řídí registraci nového uživatele.

Některé řízení presenteru je myšleno tak, že daný presenter vytváří novou instanci formuláře, který je poté vykreslen view vrstvou. Každý formulář zastupuje samostatnou jednu třídu. Které presentery pracují s formuláři je možné vidět na obrázku B.1 diagramu tříd. Všechny metody, které obsahují metodu **handle<Signál>** a jako parametr přijímají typ zprávy jsou odesílány kontroléru.

5.2.3 View vrstva

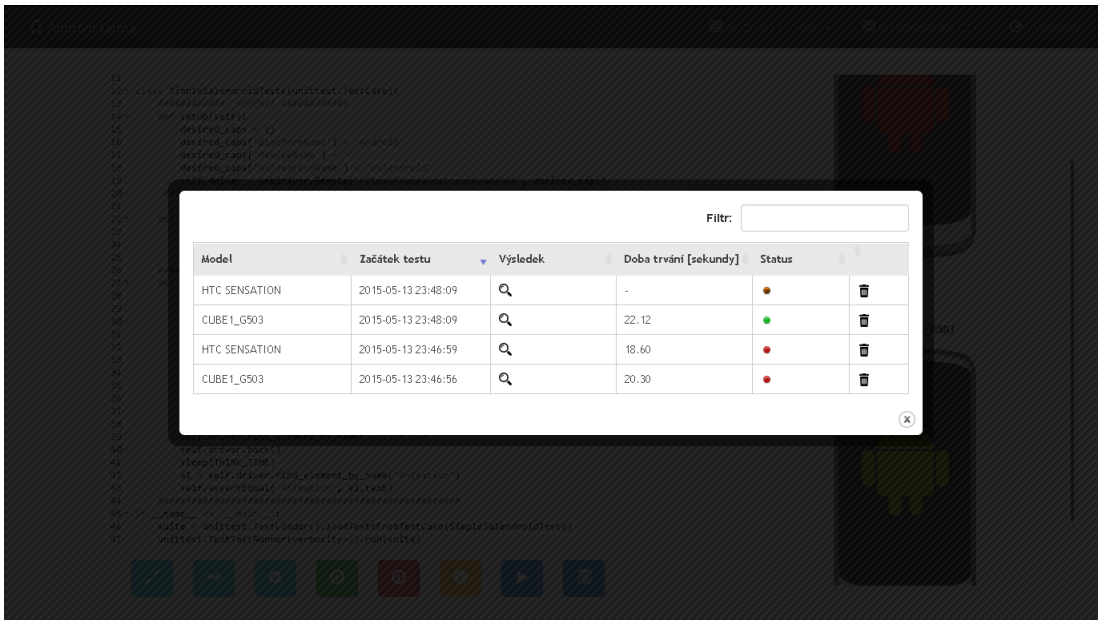
Tato vrstva má na starost zobrazení výsledku požadavku. Jak již bylo zmíněno v sekci 2.2.2, šablonovací systém, který je obsažen v Nette, nese název Latte. Ve výsledném systému je použita jedna společná šablona pro všechny ostatní šablony s názvem **@layout.latte**. Tato šablona definuje hlavičku dokumentu, CSS styly, použité JavaScript frameworky (jQuery, Highcharts, atd.) a navigační menu. Ostatní šablony pouze vkládají výsledný obsah do této společné šablony. Počet výsledných šablon je stejný jako počet stavů stavového diagramu v sekci 4.6. V rámci jednoho presenteru je vytvořeno i více šablon. Presenter poté vynutí vykreslení šablony podle názvu akce. Například pro vytvoření nového projektu bude zavolán presenter **NewProject** s akcí **newproject**. Cestu k souboru se šablonou odvodí presenter NewProject podle jednoduché logiky, a sice **/templates/NewProject/newproject.latte**. Pokud není akce definována, presenter hledá název šablony **default.latte**. Jak již bylo zmíněno v sekci 2.1, všechny šablony byly naprogramovány v jazycích HTML5, CSS a JavaScript. V rámci CSS byl použit framework Bootstrap, který podstatně usnadnil práci. I když mají šablony nejobsáhlejší kód, nejlépe jak popsat view vrstvu je formou obrázku výsledného systému. Výsledný návrh rozložení jednotlivých oken jsem se snažil dodržet podle grafického návrhu v sekci 4.7.

Na obrázku 5.1 je ukázáno vzdálené ovládání zařízení. Na levé straně obrázku je zobrazeno vytížení zařízení spuštěnou aplikací. Uprostřed obrázku je samotné zařízení, které je možné ovládat. Na pravé straně od zařízení jsou zobrazeny možnosti ovládání. Mezi tyto možnosti patří instalace aplikace, změna landscape a portrait pohledu atd. Pravá strana obrázku zachycuje aktuální záznam zařízení. Tento záznam je možné ovládat pomocí zobrazených tlačítek a filtrovat podle určitých pravidel.



Obrázek 5.1: Vzdálené ovládání zařízení.

Na obrázku 5.2 je zobrazen průběh automatizovaného hromadného testování aplikací. Tabulka, která je zobrazena na obrázku, popisuje výsledky hromadného testu. Jeden řádek tabulky popisuje testovací model zařízení, začátek testu, výsledek testu, dobu trvání testu a status. Status může nabývat 4 stavů, a sice úspěšný, neúspěšný, běžící a ve frontě.



Obrázek 5.2: Hromadné testování aplikací.

Vzhled celého výsledného systému je v příloze A.

5.3 Testování

Tato sekce se zabývá testováním výsledného systému. Testování systémů probíhalo ve dvou částech, při implementaci a pak znovu celkové testování systému. K testování byl využit desktop s operačním systémem Ubuntu 14.04, který mi byl poskytnut od vedoucího této práce. K tomuto desktopu bylo připojeno několik Android zařízení. Parametry těchto zařízení jsou znázorněny v tabulce 5.1.

Model	Android verze	Rozlišení	Typ zařízení
CUBE1 G503	4.4.2	540x960	mobil
GO CLEVER	4.0.4	1024x600	tablet
HTC SENSATION	4.0.3	540x960	mobil
NEXUS 7	4.1.2	720x1280	tablet

Tabulka 5.1: Seznam testovaných Android zařízení.

Testování probíhalo na adrese http://pclanik.fit.vutbr.cz/xctvrt05/nette-blog/www/, což je adresa již zmíněného desktopu.

5.3.1 Vývojové a testovací prostředí

Samotný vývoj a testování systému bylo prováděno v prostředí NetBeans 8.01. Jako výchozí webový prohlížeč při vývoji jsem používal Firefox. V průběhu implementace byl systém také testován v prohlížečích Google Chrome a Internet Explorer.

V prohlížeči Firefox lze využít nástroj Firebug, který v dnešní době patří mezi velice populární vývojový nástroj tohoto prohlížeče. Tento nástroj umožňuje sledování HTML, CSS a JavaScript kódu. Podporou PHP do tohoto rozšíření dodává doplněk FireLogger, díky kterému je možné vypisovat do konzole obsahy proměnných apod. Nástroj FireLogger byl využit v několika případech pro rychlé ladění. Samotný Nette Framework disponuje třídou, která ladění usnadňuje. Tato třída se nazývá *\Tracy\Debugger*, která usnadňuje odhalování chyb, výpis proměnných, měření času, zajišťuje záznam chyb a další. Zajišťuje vizualizaci chyb a výjimek, což znamená, že v případě chyby v PHP kódu se uživateli zobrazí webová stránka, která napoví, kde se chyba nachází a jak ji opravit. Mimo tyto možnosti ještě přidává plovoucí panel Debugger Bar, který obsahuje informace o jednotlivých požadavcích, které byly provedeny na danou stránku. Více o třídě *\Tracy\Debugger* na webových stránkách [6]. Validita stránek je v souladu s technickými pravidly, která jsou definována pro psaní zvoleného značkovacího jazyka. Správná kontrola komunikace byla testována pomocí nástroje wireshark.

Kapitola 6

Závěr

Cílem této diplomové práce bylo navrhnout a implementovat webovou aplikaci, která zprostředkuje vzdálený přístup k mobilním zařízením připojeným ke vzdálenému serveru. Kromě samotného přístupu musí webová aplikace podporovat hromadné testování aplikací.

V úvodu práce bylo nutné se seznámit s dostupnými prostředky, díky kterým bylo možné systém navrhnout a implementovat. Na základě těchto informací a analýzy dostupných řešení byl proveden důkladný návrh výsledného systému. Celý návrh byl doprovázen vhodnými vývojovými diagramy. Po dokončení návrhu bylo přistoupeno k samotné implementaci. Veškerá komunikační část se vzdáleným zařízením byla vytvořena pomocí skriptů v jazyce Python. Na tyto skripty byla napojena vícevrstvá architektura, která byla implementována pomocí Nette Frameworku. V rámci implementace došlo k některým změnám vůči návrhu. Jedná se především o grafické změny jednotlivých oken. Implementovaný systém splnil stanovené cíle, které byly vytyčeny na začátku této práce.

Výsledný systém je momentálně kompatibilní s operačním systémem Ubuntu. Po úspěšné instalaci výsledného systému na vzdálený server je možné začít s jeho ovládáním pomocí nejrozšířenějších webových prohlížečů Mozilla Firefox, Google Chrome a Windows Internet Explorer. Výsledný systém by měl zjednodušit a zkvalitnit otestování uživatelovi aplikace před jejím uvedením na trh. Jednak z důvodu hromadného testování, ale také i díky možnosti otestování aplikace na různých mobilních zařízeních. Tím mám na mysli např. menší firmu, která bude mít na serveru k dispozici několik různých mobilních zařízení. K tomuto serveru se bude moci připojit jakýkoliv zaměstnanec, který bude mít k dispozici více modelů k testování vytvářené aplikace.

V rámci dalšího vývoje očekávám rozšíření výsledného systému i na jiné operační systémy. Další věc, kterou bych chtěl zajistit, je lepší plynulost obrazu ovládaného zařízení. V rámci automatizovaného hromadného testování bych chtěl poskytnout uživatelům možnost tvorby nového testu pomocí uživatelovi interakce s vlastní aplikací. Tím je myšleno, že by si uživatel mohl naklikat vlastní test pro svoji aplikaci aniž by musel napsat řádek zdrojového kódu. Také bych chtěl poskytnout uživatelům možnost sdílení jednotlivých projektů mezi sebou.

Literatura

[1] ADB [online]. http://developer.android.com/tools/help/adb.html, 2014 [cit. 2014-12-29].

[2] ADB protokol [online]. https://android.googlesource.com/platform/system/core/+/master/adb, [cit. 2015-05-09].

[3] Appium [online]. http://appium.io/, 2015 [cit. 2015-05-06].

[4] Bootstrap [online]. http://getbootstrap.com/, [cit. 2015-05-10].

[5] CakePHP [online]. http://cakephp.org/, [cit. 2015-05-07].

[6] Debugování a zpracování chyb [online]. http://doc.nette.org/cs/2.3/debugging#toc-debugger-bar, [cit. 2015-05-10].

[7] Espresso [online]. https://code.google.com/p/android-test-kit/, [cit. 2015-05-06].

[8] Čtvrtníček, D.: Nástroje a technologie pro výtvaření mobilních aplikací [online]. https://www.fit.vutbr.cz/study/DP/BP.php.cs?id=15155&file=t, 2013 [cit. 2015-05-06].

[9] Fain, Y.; Rasputnis, V.; Gamov, V.; aj.: *Enterprise Web Development*. O'Reilly Media, 2014, ISBN 978-1-4493-5681-1.

[10] Fette, I.; Melnikov, A.: WebSocket protokol [online]. https://tools.ietf.org/html/rfc6455, 2011 [cit. 2015-05-08].

[11] Fielding, R.; Gettys, J.; Mogul, J.; aj.: HTTP/1.1 [online]. http://tools.ietf.org/html/rfc2616, 1999 [cit. 2015-05-08].

[12] Grudl, D.: Nette Framework [online]. http://nette.org/cs/, 2014 [cit. 2014-12-29].

[13] Harold, W.: XML-RPC specification [online]. https://tools.ietf.org/html/rfc3529, 2003 [cit. 2015-05-10].

[14] jQuery [online]. https://jquery.com/, rev. 2015-04-05 [cit. 2015-05-05].

[15] Kobayashi, T.: Android Logging System [online]. http://elinux.org/Android_Logging_System#.27log.27_command_line_tool, rev. 2012-02-07 [cit. 2014-12-29].

[16] MonkeyRunner [online].
http://developer.android.com/tools/help/monkeyrunner_concepts.html, 2014 [cit. 2014-12-29].

[17] Nette Framework composer [online]. http://doc.nette.org/cs/2.2/composer, 2014 [cit. 2014-12-29].

[18] Nette Framework dependency injection [online].
http://doc.nette.org/cs/2.2/di-usage, 2014 [cit. 2014-12-29].

[19] Nette Framework presenter [online].
http://doc.nette.org/cs/0.9/nette-application-presenter, 2014 [cit. 2014-12-29].

[20] Robbins, J. N.: *Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics*. O'Reilly Media, 2012, ISBN 978-1449319274.

[21] Robotium [online]. http://robotium.com/, [cit. 2015-05-06].

[22] Sarrion, E.: *jQuery UI*. O'Reilly Media, 2012, ISBN 978-1-4493-1698-3.

[23] Symfony [online]. http://symfony.com/, [cit. 2015-05-07].

[24] Tatroe, K.: *Programming PHP*. O'Reilly Media, 2013, ISBN 978-1449392772.

[25] TCP [online]. https://www.ietf.org/rfc/rfc793.txt, 1981 [cit. 2015-05-08].

[26] Testing Fundamentals [online].
http://developer.android.com/tools/testing/testing_android.html, [cit. 2015-05-06].

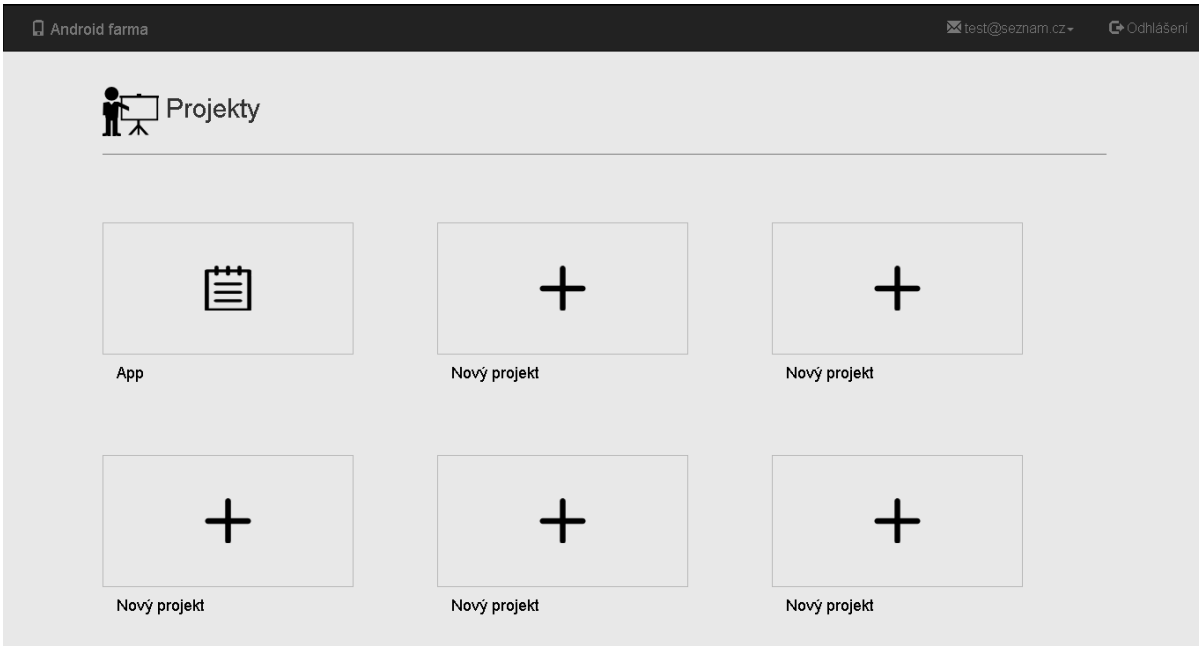
[27] UI Automator [online]. https://developer.android.com/tools/testing-support-library/index.html#UIAutomator, [cit. 2015-05-06].

[28] UNIX domain sockets [online].
http://man7.org/linux/man-pages/man7/unix.7.html, [cit. 2015-05-08].

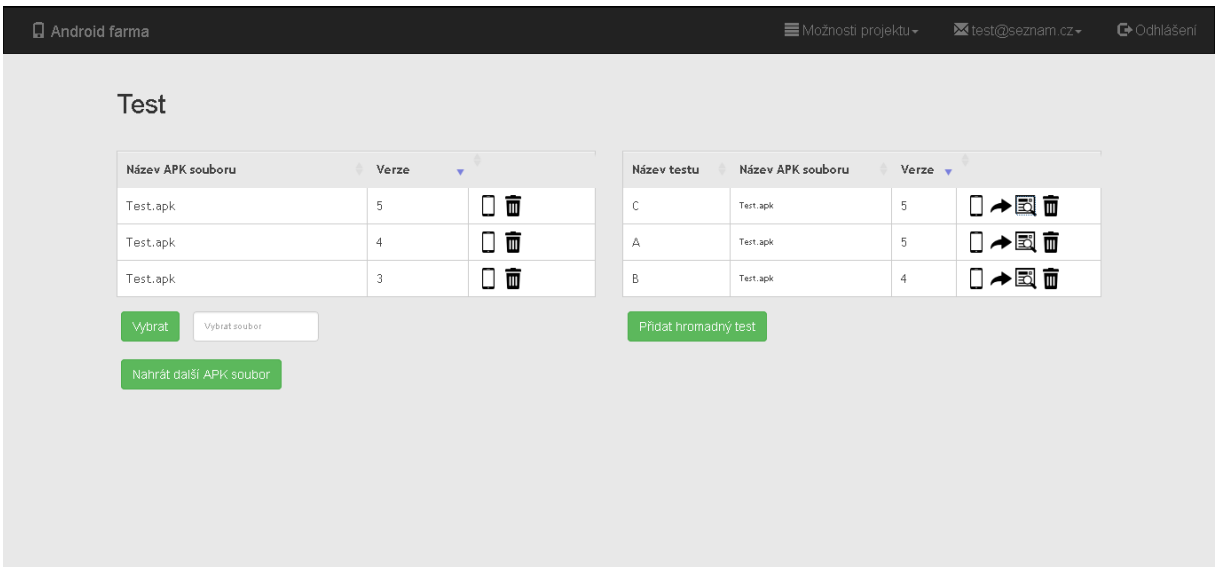
[29] Zend Framework [online]. http://framework.zend.com/, [cit. 2015-05-07].

Příloha A

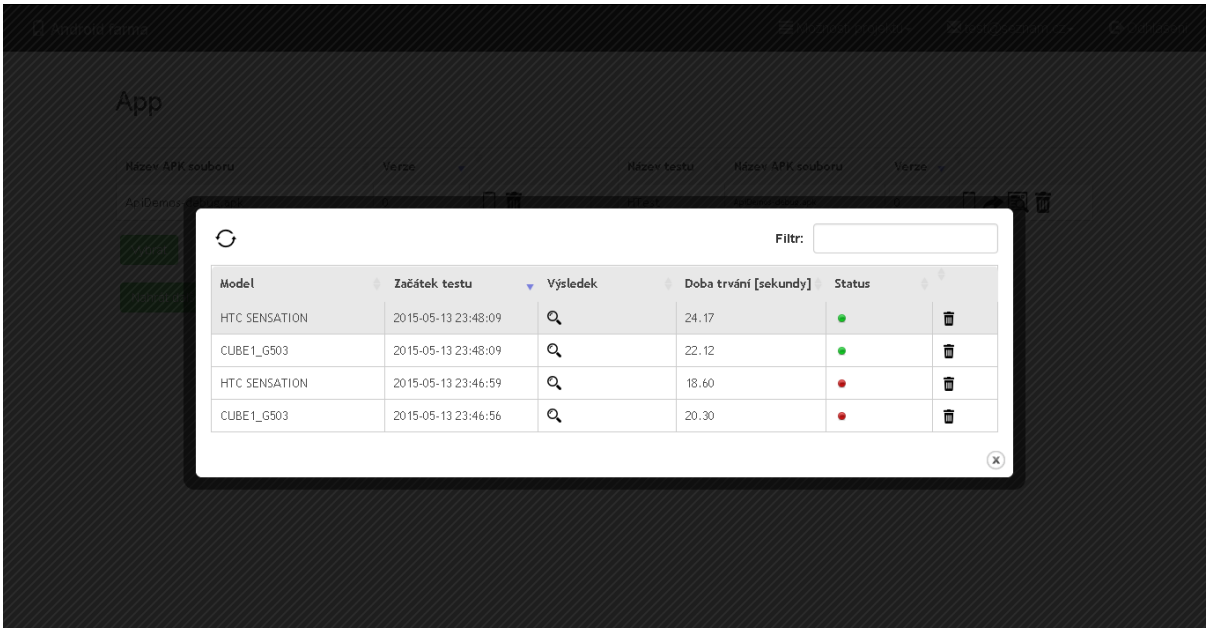
Obrázky výsledné aplikace



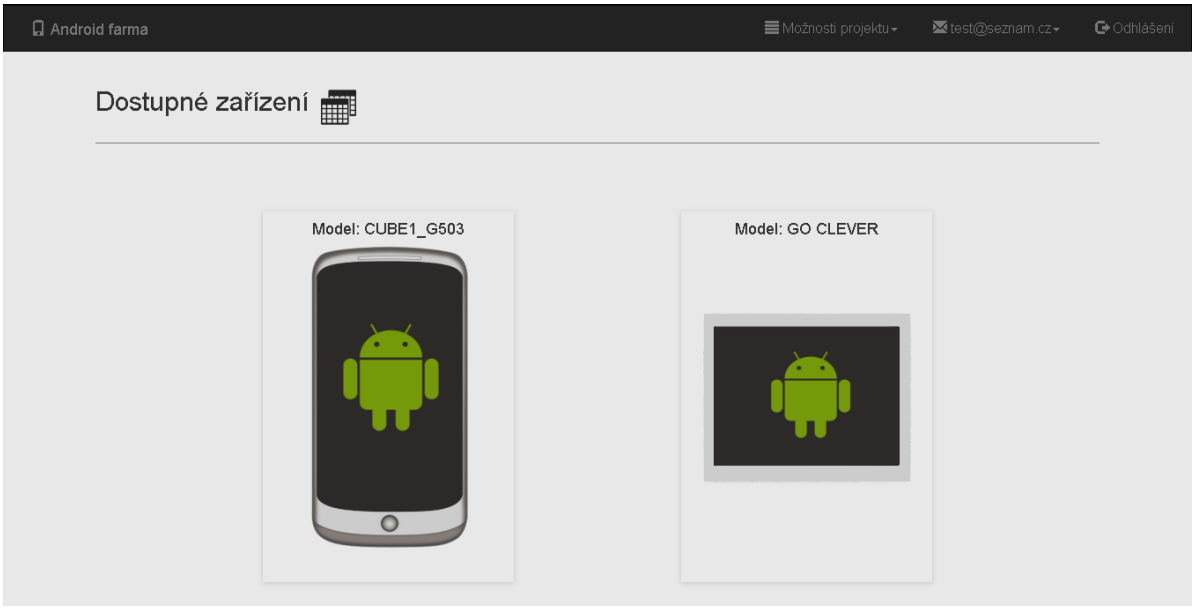
Obrázek A.1: Seznam dostupných projektů.



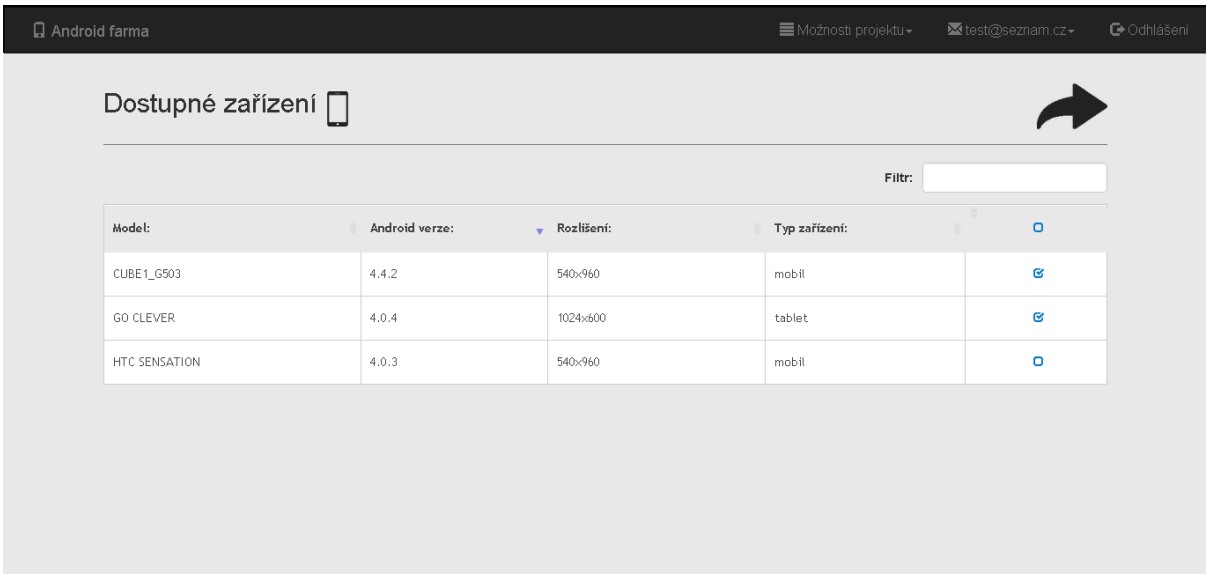
Obrázek A.2: Detail projektu.



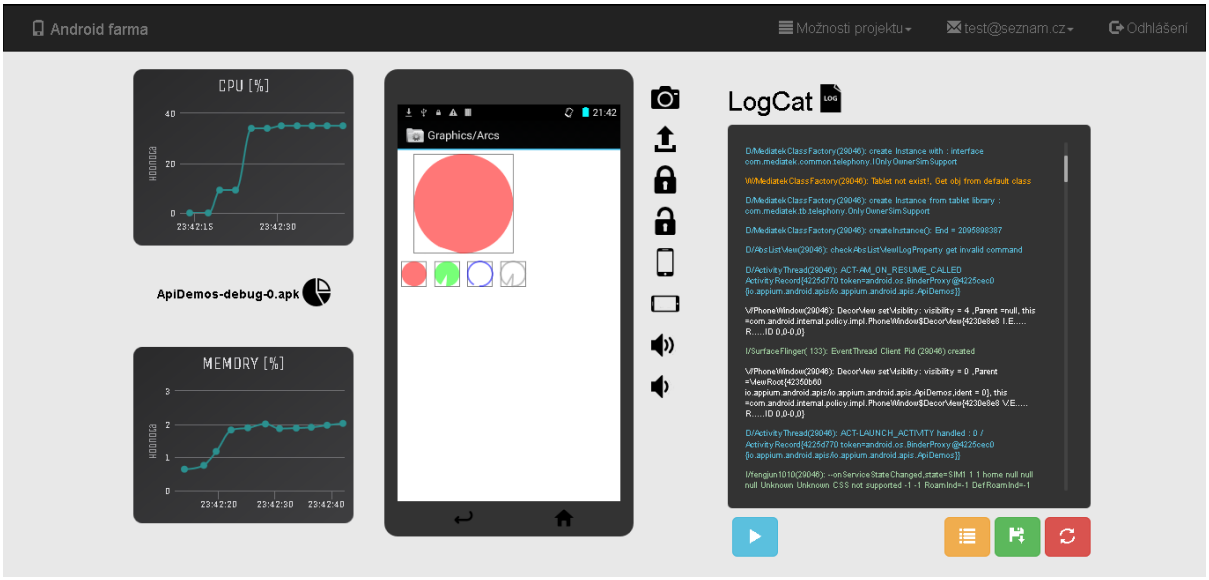
Obrázek A.3: Zobrazení výsledků hromadných testů v detailu projektu.



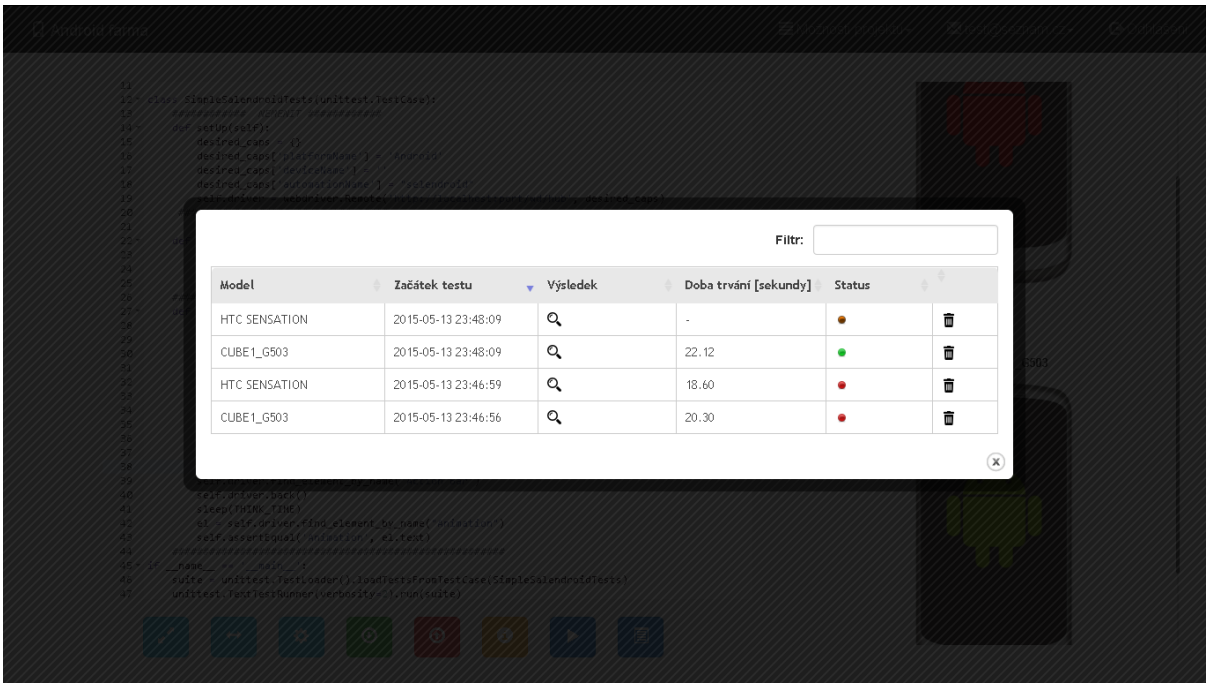
Obrázek A.4: Výběr zařízení v obrázkovém režimu.



Obrázek A.5: Výběr zařízení v tabulkovém režimu.



Obrázek A.6: Vzdálené ovládání zařízení.

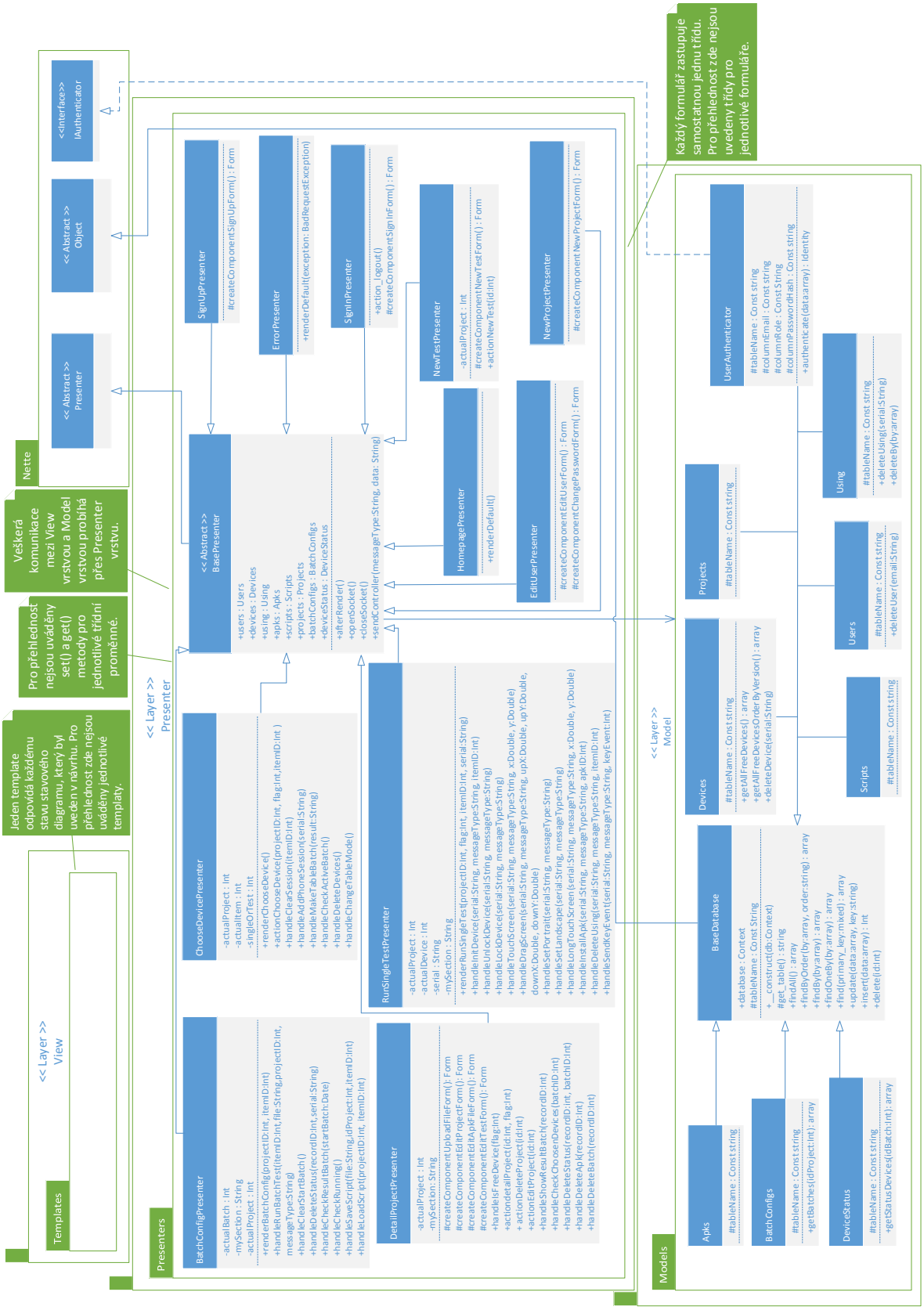


Obrázek A.7: Hromadné testování aplikací.

Příloha B

Diagram tříd vícevrstvé architektury

Na obrázku B.1 je zobrazen diagram tříd vícevrstvé architektury uživatelského modulu v Nette Frameworku.



Obrázek B.1: Diagram tříd uživatelského modulu.