FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Educhild – Backend rodičovského módu
Student:	Miloš Popovič
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem této práce je návrh a implementace serverové části rodičovského módu android aplikace Educhild, která slouží jako pomůcka pro rodiče k motivaci plnění edukačních úkolů a limitaci jiných aktivit dítěte na mobilním zařízení s OS android.

Postupujte v těchto krocích:
• Analyzujte potřeby budoucí android aplikace, spolupracujte s ostatními členy týmu zabývající se rodičovskou částí android aplikace Educhild
• Na základě analýzy proveďte řádný návrh backendu (datový model, business logika a API apod.)
• Realizujte implementaci serverového backendu společně se synchronizací rodičovského módu v android aplikaci.
• Navrhněte a realizujte vhodné sady testů
• Připravte vhodné prostředí pro budoucí rozvoj
• Zhodnoťte výsledný stav

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 17. ledna 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Educhild – Backend rodičovského módu

Miloš Popovič

Katedra softwarového inženýrství
Vedúci práce: Ing. Jiří Hunka

12. mája 2021

Poďakovanie

Chcel by som sa poďakovať vedúcemu tejto práce Ing. Jiřímu Hunkovi za to, že si vždy vedel nájsť čas na konzultácie a na rýchle odozvy na dotazy napriek tomu, že je v tomto semestri maximálne vyťažený. Taktiež by som sa mu chcel poďakovať za to, že viedol predmety súvisiace so softvérovými tímovými projektami a za to, že som mohol byť súčasťou tohto tímu. Chcel by som sa poďakovať tiež všetkým členom tímu, konktétne Petrovi Šímovi, Danielovi Matouškovi, Janovi Stejskalovi, Marekovi Trzaskalikovi, Tereze Langovej, Tomášovi Hankovi, Barbore Kyseľovej za to, že boli vždy ochotní poradiť a riešiť vzniknuté problémy na projekte. V neposlednom rade by som sa chcel poďakovať mojej rodine za ich sústavnú podporu.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 12. mája 2021 . . . . . .

Odkaz na túto prácu

Popovič, Miloš. *Educhild – Backend rodičovského módu.* Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Táto bakalárska práca sa zaoberá mobilnou aplikáciu zameranou na rodičovskú kontrolu a vzdelávanie detí. Cieľom práce je navrhnúť a realizovať server pre Android aplikáciu a webového klienta so zameraním na administratívnu a rodičovskú časť aplikácie, a zrealizovať komunikáciu Android aplikácie so serverom.

Teoretická časť práce je zameraná na výber vhodného aplikačného rámca pre realizáciu práce, analýze spôsobov asynchrónnej práce v operačnom systéme Android a následne je zanalyzovaný súčasný stav Android aplikácie, na základe ktorého sú vytvorené požiadavky na server.

V praktickej časti práce je navrhnutý server vrátane databázového návrhu, logickej a fyzickej architektúry systému a návrhu rozhrania pre komunikáciu s klientskými aplikáciami. Systém je následne realizovaný pomocou aplikačného rámca Spring Boot a nástroja Docker. V praktickej časti je taktiež popísaná forma vývoja servera pomocou DevOps a je ukázaná implementácia komunikácie Android aplikácie so serverom na základe rôznych typov požiadaviek. Okrem výsledného stavu boli v praktickej časti popísané taktiež nedostatky počas vývoja. Výsledkom práce je funkčný server realizujúci všetky funkčné požiadavky a je pripravený na používanie na testovacom prostredí.

Kľúčové slová rodičovská kontrola, webový server, REST, Spring, Android

Abstract

This bachelor thesis deals with a mobile application focused on parental control and education of children. The target of this thesis is to design and implement a server for Android application and web client with focus on the administrative and parent part of the application, and to implement communication of the Android application with the server.

The theoretical part of the work is focused on the selection of a suitable framework for the implementation, analysis of ways of asynchronous work in the Android operating system and then the current state of the Android application is analyzed, based on which server requirements are created.

Server is designed in the practical part of the thesis, including database design, logical and physical system architecture and interface design for communication with client applications. The system is then implemented using Spring Boot framework and Docker tool. The practical part also describes the form of server development using DevOps and shows the implementation of Android application communication with the server based on different types of requsets. Along with the final state of the program, deficiencies during development were described in the thesis as well. The result is a functional server that implements all functional requirements and is ready for use in a test environment.

Keywords parental control, web server, REST, Spring, Android

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam zdrojových kódov

Úvod

V súčasnosti trávia deti priveľa času na mobilných telefónoch bez väčšej kontroly rodičov. Deti strávia na mobilných telefónoch väčšinu času dňa hraním hier a sledovaním videí. Momentálna celosvetová pandémia tento problém ešte zväčšila a navyše mnoho detí nemá prístup k vzdelaniu.

Jednou z alternatív riešenia tohto problému je vytvorenie aplikácie na vzdelávanie detí s rodičovskou kontrolou. Mobilná aplikácia bude rozdelená na detskú časť, slúžiacu na vzdelávanie a monitorovanie dieťata, a rodičovskú časť, ktorá bude slúžiť na správu dieťaťa.

V rodičovskej časti aplikácie bude možné nastaviť čas strávený dieťaťom na hranie, výber aplikácií, ktoré sa zablokujú po uplynutí času určeného na hranie, nastavenie rozvrhu hrania pre dieťa a prehľad kvízov a štatistík. Detská časť bude slúžiť na kontrolu dieťaťa, zamedzenie prístupu do aplikácií určených rodičom a bude slúžiť na hranie kvízov určených rodičom alebo systémom.

Výsledok práce bude prospešný pre rodičov, ktorí získajú väčšiu kontrolu o digitálnej aktivite svojich detí a pre deti, ktoré získajú nové vedomosti zábavnou formou.

Téma práce bola zvolená s úmyslom pomôcť rodičom a deťom zvládnuť súčasnú situáciu vo svete, ale aj pomôcť vyriešiť pretrvávajúce problémy s nadmerným časom tráveným deťmi hraním hier na mobilných telefónoch.

Väčšina podobných aplikácií je zameraná výlučne na vzdelávaníe detí alebo je zameraná na rodičovskú kotrolu detí. Aplikácia si kladie za cieľ zlúčiť dané oblasti, čím poskytne účelnejšie používanie mobilného zariadenia dieťaťom.

Projekt vznikol v rámci predmetov BI-SP1 a BI-SP2 vyučovaných na FIT ČVUT v Prahe. Táto práca sa zameriava na časť aplikácie určenej pre rodiča. Kvízová časť aplikácie je riešená v rámci bakalárskej práce Petra Šímu. Implementáciou rodičovského módu na Android zariadeniach sa zaoberá bakalárska práca Daniela Matouška a implementáciou webového klienta sa zaoberá bakalárska práca Jana Stejskala.

Havným cieľom tejto práce je návrh a implementácia servera pre mobilnú aplikáciu na vzdelávanie detí s rodičovskou kontrolou. Server bude podporovať komunikáciu s mobilnou aplikáciou pre Android zariadenia a bude obsluhovať webového klienta. V Android aplikácií je cieľom navrhnúť a implementovať komunikáciu so serverom, zameranou na rodičovskú časť.

Prvá kapitola je venovaná výberu vhodnej technológie na implementáciu servera, ktorá je podstatným krokom pri tvorbe nového servera. Pre úspešnú implementáciu komunikácie na Android zariadeniach je potrebné pochopiť princípy asynchrónnej práce v systéme Android, ktorým je venovaná druhá kapitola. Súšasný stav Android aplikácie spolu s požiadavkami na server plynúcimi z funkcionalít mobilnej aplikácie sa rozoberá v kapitole Analýza. Cieľom praktickej časti je navrhnúť a implementovať server a časť Android aplikácie súvisiacu s komunikáciou so serverom. Kapitola Návrh a realizácia sa bude zaoberať rodičovskou časťou aplikácie. Cieľom praktickej časti je aj navrhnúť a realizovať vhodnú sadu testov v kapitole 5, a zhodnotiť výsledný stav a pripraviť prostredie pre budúci rozvoj v kapitole 6.

Kapitola 1

Výber vhodného aplikačného rámca pre server

Ako bolo zmienené v úvode, cieľom práce je vytvorenie servera, ktorý bude obsluhovať mobilnú aplikáciu a webového klienta. Jednou z najdôležitejších aktivít pri tvorbe nového systému je výber vhodného aplikačného rámca[1] pre danú doménu. Nasledujúca časť je venovaná analýze najdôležitejších faktorov ovplyvňujúce výber vhodnej technológie a následne sú zhrnuté najrelevantnejšie technológie vhodné na použitie pri tvorbe webového servera. Na záver je učinená voľba najvhodnejšieho aplikačného rámca.

1.1 Kritéria pri výbere vhodného aplikačného rámca

Jedným z kritérií pri výbere vhodnej technológie je popularita a veľkosť komunity. Podľa [1] je jedným z meradiel zdravia projektu počet knižných publikácií a počet príspevkov na webovej stránke stackoverflow.com, ktorá obsahuje tipy pre vývojárov. Ďalším meradlom zdravia projektu je počet ľudí na internete zaoberajúcich sa technológiou a taktiež oficiálna dokumentácia technológie. Známy a dobre udržiavaný aplikačný rámec má vyššiu pravdepodobnosť byť ďalej vyvíjaný a životaschopný.

Ďalším z kritérií pri voľbe technológie je zabezpečenie aplikácie. Ak by mal každý vývojár reimplementovať základné bezpečnostné funkcie, predstavilo by to riziko zavedenia bezpečnostnej zraniteľnosti do systému. Medzi najčastejšie zraniteľnosti patrí SQL injektovanie[2], nefunkčné overovanie[3], únik citlivých

[1] angl. framework

[2] angl. SQL injection, forma útoku, pri ktorom sa útočník snaží vložiť kód do nesprávne ošetreného databázového vstupu

[3] angl. broken authentication, forma útoku, pri ktorom sa útočník snaží získať privilégiá napadnutých užívateľov

dát a XSS útok[4]. [2] Prehľad najväčších bezpečnostných hrozieb pre webové aplikácie pravidelne aktualizuje komunita OWASP[5].

Zásadným faktorom pri voľbe aplikačného rámca je licencovanie. Mnoho licencií umožňuje používanie produktu na komerčné účely. Pred použitím programu je nutné skontrolovať licenčné podmienky a podmienky použitia. Okrem licencie samotného aplikačného rámca je nutné taktiež skontrolovať licencie všetkých použitých zásuvných modulov, nakoľko môžu mať vlastné licenčné podmienky.

Každý aplikačný rámec ponúka predimplementovanú architektúru adaptovateľnú na každú aplikáciu svojho druhu. Objektovo-orientovaný aplikačný rámec je znovupoužiteľný v rôznych aplikáciách vďaka rozširovaniu špecifických tried. Testovateľnosť rámca zhoršujú premenné a fakt, že exekúcia istej časti je podmienená samotným rámcom. [3] Preto je nutné, aby aplikačný rámec umožňoval dôkladné pretestovanie funkčností, ktoré mohol programátor rozšíriť alebo upraviť.

Ďalším faktorom je podľa [4] počet vývojárov, ktorí ovládajú daný aplikačný rámec.

1.2 Analýza aplikačných rámcov

Nasledujúca časť analyzuje najpopulárnejšie aplikačné rámce pre serverovú časť aplikácie. Hodnotenie bolo vykonané na základe počtu udelených hviezd na projekt za rok 2020 vo webovej službe GitHub, ktorá podporuje vývoj softvéru pomocou verzovacieho nástroja Git. Päť najpopulárnejších aplikačných rámcov za rok 2020 sú Laravel, Django, Flask, Spring Boot, Express [5].

1.2.1 Laravel

Laravel je webový aplikačný rámec napísaný v jazyku PHP. Je založený na aplikačnom rámci Symfony. Jeho prvá verzia bola zverejnená v roku 2011 a momentálne je zverejnená jeho ôsma verzia. Laravel je licencovaný pod licenciou MIT[6] [6], povoľujúcou opätovné využitie kódu na ľubovoľný účel, preto je vhodná na komerčné využitie. Laravel implementuje architektonický návrhový vzor MVC. Laravel poskytuje modulárny systém balíkov, umožňujúci rozdeliť časti kódu a účelné spravovanie závislostí. [7]

V rámci aplikačného rámca sú poskytnuté bezpečnostné funkcionality ako autentizácia, autorizácia, ochrana proti podvrhnutiu požiadavky medzi rôz-

[4]angl. Cross-site scripting, forma útoku, kde sa útočník snaží vložiť kód do webovej stránky overeného užívateľa

[5]The Open Web Application Security Project

[6]slobodná softvérová licencia vytvorená Massachusettským technologickým inštitútom.

nymi stránkami[7], XSS útoku a SQL injektovaniu, zavedenie relácií[8], manipulácia s chybami. Ďalšou funkcionalitou je objektovo-relačné mapovanie[9], umožňujúce namapovať objektové triedy jazyka PHP na relačný databázový model. Laravel natívne poskytuje aj interakciu s databázou, umožňujúcu prepojenie databázy s aplikačným rámcom a dotazovanie v jazyku SQL, podporujúce ochranu proti injekcii kódu. V rámci databáz sú podporované aj funkcionality ako stránkovanie, migrácia medzi rôznymi verziami databázy alebo generovanie dát pre účely testovania. [8]

Laravel má prednastavené testovacie prostredie. V rámci testovania rozdeľuje testy do dvoch adresárov, a to `Unit` pre jednotkové testovanie, určené na testovanie jednotlivých častí kódu a `Feature`, v ktorom sú testy určené pre testovanie rozsiahlejšieho kódu a celkovej funkčnosti systému, ako napríklad HTTP testovanie, slúžiace na kontrolu správnosti HTTP požiadaviek, Laravel Dusk pre simuláciu webového prehliadača a testovanie API. Laravel poskytuje aj nástroje na testovanie databáz, ako resetovanie databázy po teste, generovanie dát alebo vytváranie modelu pomocou továrných metód a testovanie väzieb medzi modelmi. Aplikačný rámec poskytuje simulovanie rôznych komponentov[10], vďaka ktorým sa nemusia exekuovať skutočné komponenty. Pre testovanie je pripravené prostredie, ktoré sa konfiguruje v rámci súboru `.env.testing`.

Viac informácii o aplikačnom rámci je možné získať v oficiálnej dokumentácii [9].

1.2.2 Django

Django je aplikačný rámec vyvíjaný v programovacom jazyku Python, známy svojou jednoduchosťou a silnou podporou integrácie s inými jazykmi a nástrojmi. Django zdieľa filozofiu jazyka Python, a to filozofiu „batérií súčasťou“ – mať bohatú a všestrannú štandardnú knižnicu, ktorá je okamžite k dispozícii bez nutnosti sťahovania samostatných balíkov [10]. Django bol prvýkrát vydaný v roku 2005 a v súčasnosti je vydaná jeho tretia verzia. Je vyvíjaný pod licenciou BSD[11], a teda je vhodný na komerčné použitie.

Django kladie dôraz na rýchly vývoj a pragmatický dizajn a prezýva sa ako „webový rámec pre perfekcionistov s termínmi“. Sústreďuje sa na automatizáciu a dodržiavanie princípu neopakovania sa[12]. Django nazýva architektúru svojho systému MVT, ktorá je ale veľmi podobná architektúre MVC. Aplikačný rámec je škálovateľný a používa takzvanú „shared-nothing“ architek-

[7] angl. Cross-Site Request Forgery

[8] angl. sessions, dočasná a interaktívna výmena informácií medzi komunikujúcimi zariadeniami

[9] ORM, angl. object-relational mapping

[10] angl. mocking

[11] Berkeley Software Distribution

[12] angl. Don't repeat yourself, DRY

túru, čo znamená, že je možné pridať hardvér na ľubovoľnej úrovni (databázové, kešovacie alebo aplikačné servery). Django má v sebe taktiež zabudované bezpečnostné funkcionality podobne ako aplikačný rámec Laravel, podporuje ORM, migrovanie databázoveho modelu alebo stránkovanie.

Django používa na testovanie knižnicu `unittest`, štandardnú knižnicu jazyka Python. Pre databázové testovanie je možné použiť databázu SQLite, vytvorenú v pamäti programu alebo je možné prepojiť reálnu PostgreSQL databázu. Pre testovanie koncových bodov je možné využiť testovacieho klienta, ktorý ako trieda v jazyku Python simuluje webový prehliadač. Je možné taktiež využiť továrenské metódy pre testovanie API. Viac informácií o aplikačnom rámci Django je možné získať z oficiálnej dokumentácie [11].

1.2.3 Flask

Ďalším aplikačným rámcom, vyvíjaným v jazyku Python, je Flask, ktorý vo svojom základe ponúka minimalistické funkcionality a nazýva sa ako tzv. mikroframework. Je vhodný na malé projekty a na rozdiel od rámca Django vývojári používajú rozšírenia a knižnice pre ďalšie funkcionality[13], čím zabezpečí flexibilitu výberu najvhodnejších komponentov pre danú aplikáciu. [12]

1.2.4 Spring Boot

Spring je aplikačný rámec vyvíjaný na platforme Java, umožňujúci vyvinúť aplikáciu z obyčajného Java objektu[14] a aplikuje podnikové služby na tieto objekty, využívajúc Java EE[15] platformu. Jeho prvá verzia vyšla v roku 2002 a momentálne je zverejnená jeho piata verzia. [13] Spring je licencovaný pod open-source licenciou Apache 2.0 [14]. Spring Boot umožňuje vytvoriť samostatnú aplikáciu, založenú na aplikačnom rámci Spring, pripravenú na okamžité použitie s nutnosťou minimálnej konfigurácie, umožňujúci rýchlejší a prístupnejší počiatočný vývoj. Momentálne je prístupná druhá verzia aplikačného rámca Spring Boot. [15]

Spring je zložený z funkcionalít organizovaných do približne 20 modulov, medzi ktoré patrí napríklad modul JDBC[16], ponúkajúci abstraktnú vrstvu nad rozhraním pre prístup k relačným databázam, modul ORM, slúžiaci na objektovo-relačné mapovanie alebo modul Transaction, slúžiaci na manažment databázových transakcií. Modul Web-Servlet obsahuje implementáciu architektonichého vzoru MVC pre webové aplikácie. Pre autorizáciu, autentizáciu, manažment sedení, ochranu proti útokom a ostatné bezpečnostné funkcionality je určený modul Spring Security. Modul Test ponúka testovanie Spring

[13] Jedná sa o prístup, v ktorom si vývojár sám zvolí, ktoré knižnice chce používať, angl. bring-your-own-library appproach

[14] angl. Plain old Java object, POJO

[15] Java Enterprise Edition

[16] Java Database Connectivity

komponent spolu s JUnit aplikačným rámcom, určený pre jednotkové testy v jazyku Java.[13]

Aplikačný rámec poskytuje modul AOP, určený pre aspektovo-orientované programovanie, zvyšujúce modularitu projektu pomocou prierezových záujmov[17], kde je možné aspekt ako jednotku modularity rozdeliť do viacerých typov a objektov. Spring má taktiež vlastný výrazový jazyk Spring Expression Language (SpEL), ktorý umožňuje dotazovanie a manipuláciu objektového grafu za behu programu. [13]

1.3 Zhrnutie

V tejto časti boli zanalyzované aplikačné rámce Laravel, Django a Spring Boot. Všetky zvolené rámce implementujú architektonický vzor MVC alebo jemu podobný vzor, obsahujú funkcionality pre prácu s relačnou databázou a zahŕňajú potrebné bezpečnostné funkcionality a umožňujú dôkladné pretestovanie systému. Všetky aplikačné rámce sú licencované licenciami umožňujúce ľubovoľné využitie a modifikáciu zdrojového kódu. Tieto aplikačné rámce sú teda vhodné pre použitie na serverovú časť aplikácie. Aplikačný rámec Flask je vhodný pre menšie projekty a nemá zabudované funkcionality na vývoj väčšieho systému, a teda v porovnaní s ostatnými analyzovanými aplikačnými rámcami nie je vhodný pre vývoj serverovej časti aplikácie.

	Laravel	Django	Flask	Spring Boot
Programovací jazyk	PHP	Python	Python	Java
Popularita (GitHub 2020)	63 100	54 500	53 300	52 500
Licencia	MIT	BSD	BSD	Apache 2.0
Architektúra	MVC	MVT	–	MVC
Testovanie	PHPUnit	unittest	pytest	JUnit
Zabudované bezpečnostné funkcionality	áno	áno	nie	áno

Tabuľka 1.1: Prehľad základných kritérií zanalyzovaných aplikačných rámcov

Na základe splnenia všetkých zmienených kritérií aplikačnými rámcami Laravel, Django a Spring Boot, ďalším kritériom pri výbere aplikačného rámca je počet vývojárov ovládajúcich danú technológiu, čomu najviac vyhovuje aplikačný rámec Spring Boot, a teda ďalej bude v tejto práci používaný aplikačný rámec Spring Boot a programovací jazyk Kotlin, ktorý je plne interoperabilný s programovacím jazykom Java a samotným aplikačným rámcom.

[17] angl. cross-cutting concerns

Kapitola 2

Asynchrónna práca v operačnom systéme Android

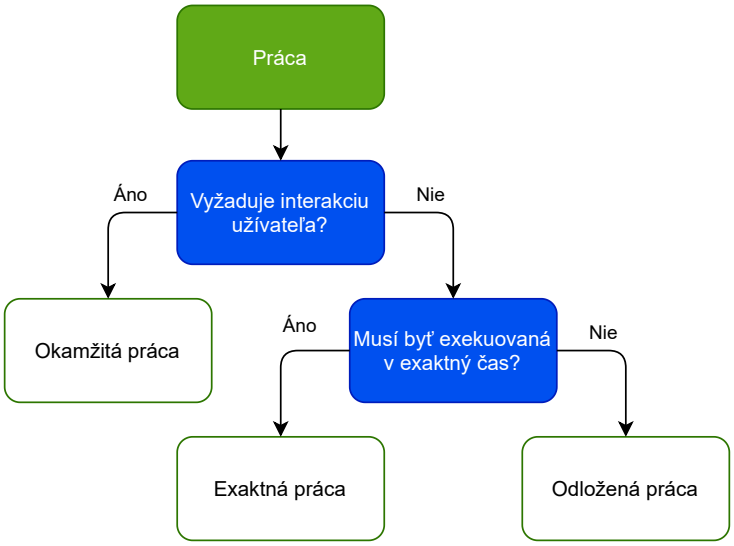
Dôležitou súčasťou väčšiny Android aplikácií vrátane aplikácie Educhild je komunikácia so serverom, na základe ktorej získavajú a odosielajú dáta potrebné pre činnosť aplikácie. Pre komunikáciu je nutná práca mimo hlavného vlákna, slúžiaceho primárne na prácu na popredí. Táto podkapitola je venovaná analýze spôsobov práce na pozadí v súvislosti s komunikáciou so serverom, ktorá je dôležitá pre pochopenie návrhu a implementácie komunikácie Android aplikácie so serverom v podkapitole 4.6.

V Android aplikácii je frekvencia snímkov typicky 60 Hz [16], čo znamená, že hlavné vlákno má čas na vykreslenie jedného snímku približne 16 milisekúnd. Z toho dôvodu je nutné vykonávať zložitejšie procesy na pozadí aplikácie. V operačnom systéme Android existujú tri kategórie prác na pozadí, a to okamžitá, odložená a exaktná (Immediate, Deferred, Exact).

Okamžitá práca vyžaduje interakciu užívateľa s aplikáciou. Pre okamžitú prácu sa využívajú v programovacom jazyku Java vlákna. V programovacom jazyku Kotlin sa v súčasnosti taktiež využívajú korutiny. V špecifických prípadoch je taktiež možné využiť komponent `Service`. Pre prácu bez priamej užívateľskej interakcie a prácu, ktorá môže byť kedykoľvek v budúcnosti odložená na neskôr, je možné využiť odloženú prácu. Pre odloženú prácu je odporúčané využiť knižnicu `WorkManager`. Pre prácu, ktorú je nutné exekuovať v presne stanovený čas, je možné využiť triedu `AlarmManager`. [17] Keďže je exaktná práca vhodná pre práce typu budík, nebude v nasledujúcom texte uvažovaná ako relevantná možnosť komunikácie so serverom.

2.1 Vlákna

Pred zavedením Kotlin korutín boli pre exekúciu okamžitej práce spúšťané vlákna na pozadí. Pre použitie viacerých vlákien v Android aplikácii je nutné



Obrázok 2.1: Rozhodujúci strom pre kategorizáciu prác na pozadí v OS Android podľa [17]

vytvoriť fond vlákien[18] pri inicializácii aplikácie s fixným alebo dynamickým počtom vlákien.

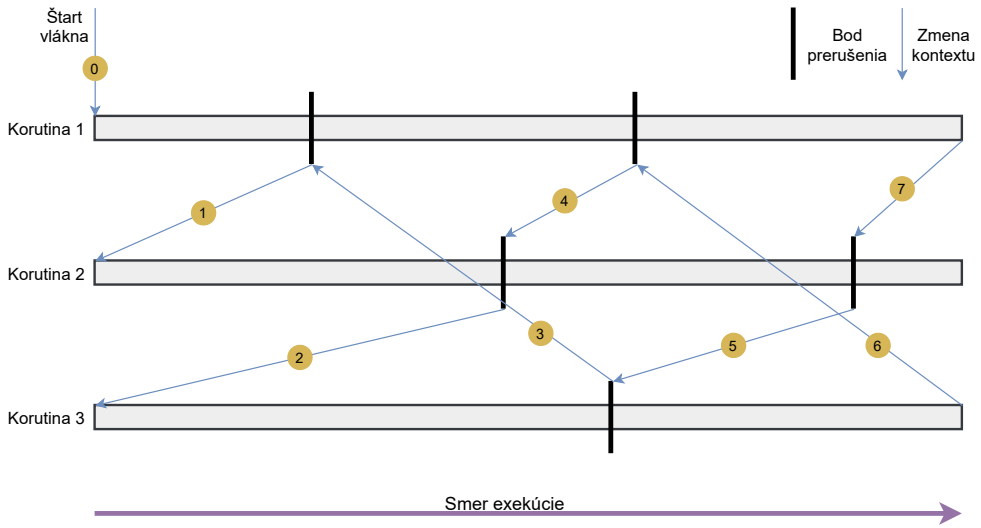
Pre exekúciu práce je nutné vytvoriť inštanciu rozhrania `Runnable` s implementáciou logiky práce, ktorá sa exekuuje v inštancii vlákna `Executor`. Pre komunikáciu s hlavným vláknom je možné využiť tzv. spätné volanie[19], ale volanie sa vykoná vo vlákne na pozadí, a teda nie je možné komunikovať priamo s užívateľským rozhraním. Pre zaradenie akcie pre exekúciu v inom vlákne je možné použiť triedu `Handler`. [18]

2.2 Kotlin korutiny

Korutiny sú v súčasnosti podľa [19] odporúčaný spôsob pre okamžitú asynchrónnu prácu v systéme Android. Boli pridané do programovacieho jazyka Kotlin vo verzii 1.3. Kotlin korutiny podporujú oproti bežným vláknam prerušenie, vďaka čomu sa môžu v jednom reálnom vlákne exekuovať viacero korutín bez nutnosti blokovania vlákna. Vďaka prerušeniu sú korutiny nenáročné, šetria pamäť a nároky na procesor.

Android aplikácie v súčasnosti používajú architektúru MVVM[20], ktorá spočíva v tom, že pohľad (View) získava dáta z komponentu ViewModel, ktorý ďalej spravuje dáta s úložiskom. Komponent ViewModel má svoj životný cyklus, ktorý pozostáva z vytvorenia pri inicializácii prvého pohľadu až

[18] angl. thread pool
[19] angl. callback
[20] Model-View-ViewModel



Obrázok 2.2: Ukážka použitia viacero korutín v rámci jedného vlákna podľa [20]. Korutina napriek pozastaveniu neblokuje použité vlákno.

po deštrukciu, ktorá nasleduje po tom, čo danú inštanciu nevyužíva žiaden pohľad.

Každá korutina má svoj rozsah platnosti[21] a kontext[22] reprezentujúci stav korutiny a umožňujúci riadenie a rušenie všetkých inštancií v rámci rozsahu platnosti. [21, 22] Komponent ViewModel pri použití korutín vykonáva procesy na pozadí pomocou Android knižnice `lifecycle-viewmodel-ktx`, poskytujúcu preddefinovaný rozsah platnosti korutín `viewModelScope`, ktorý automaticky zruší všetky aktívne korutiny pri deštrukcii inštancie `ViewModel`. Tento mechanizmus umožňuje menšie úniky pamäte[23] oproti pôvodným spôsobom riešenia asynchrónnej práce (vlákna s použitím komponentu `Handler` alebo už zastaraný komponent `AsyncTask` [23]).

Pri spúšťaní korutiny je odporúčané explicitne definovať tzv. dispečera, ktorý určí, v ktorom fonde vlákien sa má korutina primárne spúšťať v závislosti od typu operácie. V OS Android slúži dispečer `Dispatchers.Main` pre prácu súvisiacu s vykresľovaním prvkov na obrazovke, dispečer `Dispatchers.IO` pre vstupno-výstupné operácie pri práci s úložiskom alebo so sieťou a dispečer `Dispatchers.Default` – zvolený implicitne –, určený pre výpočetne náročné operácie. [19]

[21] rozsah platnosti sa vytvára pomocou rozhrania `CoroutineScope`, typicky je rozsah vytvorený použitím zabudovaných funkcií ako `launch`, `async`, atď.

[22] každý kontext implementuje rozhranie `CoroutineContext`, ktorý obsahuje inštanciu rozhrania `Job` pre riadenie korutiny, dispečera pre určenie vlákna, v ktorom pobeží korutina a názov korutiny

[23] angl. memory leaks

2.3 Komponent Service

Komponnent Service je určený pre dlhotrvajúce operácie na pozadí. Komponent môže byť exekuovaný aj bez interakcie užívateľa alebo prítomnosti užívateľa v aplikácii. Komponent implicitne využíva hlavné vlákno procesu, a teda pre zabránenie neodpovedaniu aplikácie je nutné vytvoriť separátne vlákno. Každá inštancia má svoj životný cyklus začínajúci metódou `onStartCommand`, ktorej návratová metóda určuje, akým spôsobom sa bude inštancia chovať pri prerušení procesu (napr. konštanta `START_NOT_STICKY` opätovne spustí inštanciu len v prípade, že v systéme sú čakajúce požiadavky na spustenie danej inštancie).

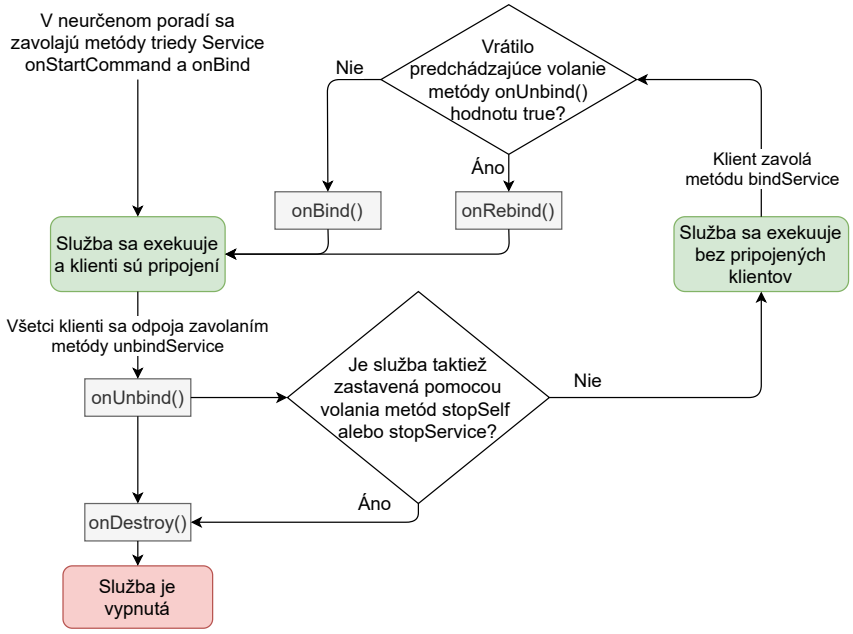
Každá implementácia komponentu musí byť deklarovaná v tzv. manifeste aplikácie, ktorý popisuje esenciálne informácie o danej aplikácii. V manifeste je taktiež možné definovať dostupnosť implementácii ostatným aplikáciám. Existujú tri typy tohto komponentu, a to služba na popredí, služba na pozadí a viazaná služba (Foreground Service, Background Service, Bound Service).

Služba na popredí je spomedzi ostatných typov iba výnimočne zastavovaná systémom v prípade, že má systém nedostatok prostriedkov. Táto služba pokračuje v exekúcii aj po opustení aplikácie užívateľom a z toho dôvodu je vhodná najmä pre operácie, ktoré vyžadujú dlhotrvajúci neprerušovaný proces, ako sú napríklad prehrávanie hudby alebo monitorovanie činností užívateľa. Služba na popredí musí byť viditeľná užívateľovi, a preto je nutné bezodkladne pri vytvorení služby vytvoriť notifikáciu viditeľnú v stavovom riadku[24], ktorú nie je možné odstrániť po dobu exekúcie služby na popredí. Od API verzie 28 systému Android je nutné vyžiadať od užívateľa špeciálne povolenie umožňujúce použitie tohto typu služby. Od rovnakej verzie je nutné taktiež deklarovať v manifeste aplikácie používanie lokalizácie zariadenia a od verzie 29 je nutné deklarovať taktiež použitie fotoaparátu alebo mikrofónu.

Služba na pozadí je určená na vykonávanie operácií, ktoré nie sú priamo určené pre užívateľa. Tento typ služby je náchylnejší na zastavenie systémom v prípade nedostatku prostriedkov. Je určený napríklad pre vstupno-výstupné operácie. Pre zlepšenie užívateľského zážitku systém od API verzie 26 ukladá restrikcie, čo môže proces na pozadí vykonávať, a preto je vhodné použiť pre dlhotrvajúce procesy vyžadujúce neprerušenie využiť službu na popredí.

Viazaná služba je typ služby, kde je umožnené ostatným komponentám spojiť sa so službou, odosielať službe správy, prijímať odpovede a vykonávať medziprocesnú komunikáciu. Viazaná služba sa typicky exekuuje len počas toho, čo obsluhuje inú komponentu aplikácie a nebeží na pozadí neobmedzene. Viazanú službu je ale možné počas exekúcie zmeniť na službu na pozadí a naopak. Podrobnejšie informácie o komponente Service je možné zistiť v [25].

[24] angl. status bar



Obrázok 2.3: Diagram životného cyklu komponentu Service typu služba na pozadí a viazaná služba podľa [24]. Komponent môže byť spustený ako služba na pozadí pomocou metódy `onStartCommand` alebo ako viazaná služba pomocou metódy `onBind` alebo kombináciou týchto metód. Služba je ukončená až potom, čo sa všetci klienti (ostatné komponenty aplikácie) odpoja pomocou metódy `onUnbind` a zároveň je zastavená služba na pozadí pomocou metódy `stopSelf` alebo `stopService`.

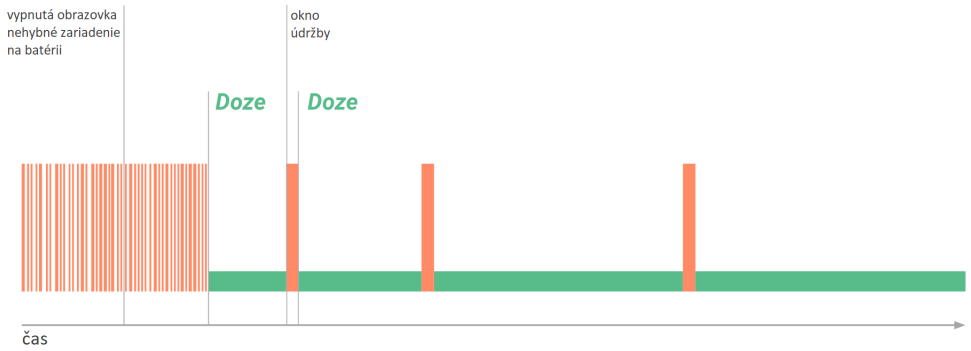
2.4 Knižnica WorkManager

Pre odloženú prácu je v súčasnosti podľa [26] odporúčané použiť knižnicu `WorkManager`. Knižnica nahrádza všetky existujúce riešenia pre odloženú prácu a poskytuje jednotné aplikačné rozhranie. Na základe levelu Android API zariadenia si knižnica vyberie najvhodnejšie riešenie spomedzi ponúkaných spôsobov, napríklad pre zariadenia s levelom API 23 alebo vyšším je použitý komponent JobScheduler.

Práca je v knižnici reprezentovaná abstraktnou triedou `WorkRequest`. Pre začlenenie práce do exekúcie je nutné ju zaradiť do fronty, interne reprezentovanú ako tabuľka v Android databáze Room. Každú prácu je nutné zadefinovať ako jednorazovú alebo periodicky sa opakujúcu. Každá práca môže byť zaradená do fronty ako unikátna so špecifickým názvom, vďaka čomu sa zabráni duplikovaniu rovnakej práce a prácu je následne možné identifikovať a manipulovať s ňou podľa potreby. Každej práci je taktiež možné priradiť jednu alebo viacero značiek, vďaka ktorým je možné manipulovať s viacero prácami súčasne.

Každej inštancii je možné taktiež pridať parametre v podobe primitívnych dátových typov alebo typu `String`. V rámci exekúcie práce v metóde `doWork` slúži návratový typ ako identifikátor výsledku, t. j. úspech, neúspech alebo identifikátor značiaci nutnosť prácu zopakovať. Jednorazová práca sa exekuuje čo najskôr, pri periodickej práci je nutné zadefinovať periodicitu práce, minimálny interval opakovania je 15 minút. Každej práci je možné taktiež nastaviť obmedzenia pre beh, napríklad nutnosť pripojenia do siete pomocou technológie Wi-Fi alebo nutnosť zariadenia nebyť v režime pre slabú batériu a podobne.

Všetky Android zariadenia podľa [27] disponujú od API verzie 23 funkcionalitami, ktoré pomáhajú šetriť energiu. Takzvaný mód Doze sa aktivuje v prípade, že je zariadenie nenapájané, nehybné a má vypnutú obrazovku. Systém v móde Doze znemožní aplikáciám používať sieť a odkladá všetky práce na neskôr vrátane prác vytvorených knižnicou WorkManager. Systém následne v tzv. oknách údržby[25] umožní aplikáciám prístup do siete a exekúciu prác. Okná údržby sa plánujú v závislosti na dlhotrvajúcej neaktivite užívateľa čím ďalej, tým menej často. Ďalšou funkciounalitou je tzv. App Standby, ktorá zabráňuje prístup do siete aplikáciám, ktoré užívateľ nepoužíva.



Obrázok 2.4: Ukážka fungovania Android funkcionality Doze. Okná údržby sa spúšťajú v závislosti na trvajúcej neaktivite užívateľa menej často. [27]

[25] angl. maintenance windows

Kapitola 3

Analýza

Táto kapitola sa zaoberá analýzou budúcich potrieb Android aplikácie a analýzou požiadaviek na server. V prvej časti je popísané fungovanie Android aplikácie s ohľadom na server, následne sú na základe potrieb spísané funkčné a nefunkčné požiadavky na server a záver kapitoly je venovaný dekomponovaniu funkčných požiadaviek do jednotlivých prípadov použitia.

3.1 Charakteristika a požiadavky Android aplikácie

Počas trvania projektu v rámci predmetu BI-SP1 bola vytvorená Android aplikácia v režime bez komunikácie s interným serverom, ktorý by uchovával dáta a umožnoval by riadenie aplikácie mimo zariadenia. Vznikla teda potreba realizovať server na podporu Android aplikácie Educhild. Nasledujúca časť opisuje funkčnosti vyžadujúce kooperáciu servera, ktoré boli zistené reverznou analýzou Android aplikácie, a uvádza potreby aplikácie voči serveru.

3.1.1 Prihlasovanie a registrácia

Firebase je platforma, ktorá je určená pre vyvíjanie mobilných a webových aplikácií, a poskytuje produkty ako monitorovanie chýb a výkonu aplikácie, pomáha distribuovať mobilné aplikácie, poskytuje autentizačné služby, hostingové a cloudové služby alebo odosielanie notifikácií, atď. [28] V aplikácii Educhild sa okrem iného používa táto platforma pre autentizačné služby. Autentizácia prebieha na základe tokenov, ktoré užívateľ obdrží po úspešnej autentizácii. Podľa [29] je možné získať na základe tokenu okrem iného údaje ako:

- id užívateľa;
- e-mail, ak je dostupný;

- verifikácia e-mailu;
- údaje vydávateľa tokenu;
- užívateľské meno a profilovú fotografiu.

Na základe týchto údajov je požadované vytvoriť autentizačný proces na serveri spolu s procesom registrácie a prihlasovania. Počas registrácie je nutné zaregistrovať na serveri rodiča.

3.1.2 Rodič a autor kvízov

Každý užívateľ, ktorý sa zaregistruje v mobilnej aplikácii, musí mať rolu rodiča. Po vytvorení rodiča by mala bezprostredne nadchádzať registrácia aspoň jedného dieťaťa. Rodič si po vytvorení prvého dieťaťa uchováva osobné preferencie, ktoré sa využijú pri registrovaní ďalších detí, ak táto možnosť bude v budúcnosti k dispozícii. Momentálne sa uchovávajú osobné preferencie na voľbu predvolených blokovaných aplikácií. U rodiča sa taktiež uchováva predvolený jazyk aplikácie.

Každý rodič bude mať možnosť vytvárať vlastné kvízy vo webovej aplikácii Educhild. V tom prípade rodič nadobudne rolu autora. Autor však nemusí byť rodič, ak sa registruje cez webovú aplikáciu a neplánuje si vytvárať rodičovský účet.

3.1.3 Dieťa

Dieťa je základnou entitou v Android aplikácii potrebnou k realizácii projektu. Aplikácia si zaznamenáva meno a fotografiu dieťaťa, potrebné pre budúce účely aplikácie, kde rodič bude môcť spravovať viacero detí v rámci jedného zariadenia.

Aplikácia si taktiež uchováva maximálny čas, ktorý je možné stráviť dieťaťom v aplikáciách označených ako blokované a reálne strávený čas v týchto aplikáciách, ktorý sa resetuje pri splnení podmienok určených aplikáciou. Po prekročení maximálne stráveného času aplikácia zablokuje prístup do blokovaných aplikácií. Aplikáciu bude možné využívať aj bez definovaného maximálneho času stráveného v blokovaných aplikáciách, a to využívaním iba vzdelávacej časti aplikácie a prehliadaním aplikačných štatistík dieťaťa.

Každému dieťaťu je možné nastaviť predvolené nastavenia blokovaných aplikácií, konkrétne zablokovanie všetkých aplikácií, ktoré sú zaradené v kategórii hry alebo sociálne siete. Taktiež je možné automaticky zablokovať všetky novo nainštalované aplikácie. U každého dieťaťa je potrebné zaznamenávať rok a mesiac narodenia pre účely relevantného obsahu vo vzdelávacej časti aplikácie.

3.1.4 Monitorované aplikácie

Pre účely správy blokovaných aplikácií je nutné zaznamenávať všetky aplikácie určené k monitorovaniu, t. j. aplikácie, ktoré nie sú systémové a zároveň majú meno a ikonu, a nejedná sa o aplikáciu Educhild. Pri týchto aplikáciách je nutné zaznamenávať blokovaný status a status o tom, či je aplikácia nainštalovaná (aplikácia sa monitoruje istú dobu po odinštalácii kvôli štatistikám). Každá aplikácia má unikátne id aplikácie podobné názvosloviu balíkov v programovacom jazyku Java [30].

Pre správne zobrazenie štatistík je nutné zaznamenávať kategóriu monitorovaných aplikácií. Kategórie aplikácií sú určené fixne konštantami z triedy `android.content.pm.ApplicationInfo`. Podľa [31] sú kategórie aplikácií v systéme Android zavedené od API verzie 26. Minimálna podporovaná verzia API aplikácie Educhild je 21 k dátumu zverejnenia práce. Z toho dôvodu je nutné sprostredkovať zariadeniam s API platformy Android nižším ako 26 informácie o kategórii aplikácii pomocou servera.

Na základe potrieb webovej aplikácie je nutné sprostredkovať webovej aplikácii taktiež názov aplikácie a ikonu aplikácie.

Konštanta	Hodnota	Kategória aplikácie
`CATEGORY_UNDEFINED`	-1	Nedefinovaná
`CATEGORY_GAME`	0	Hry
`CATEGORY_AUDIO`	1	Hudba, hudobné prehrávače
`CATEGORY_VIDEO`	2	Video, filmy, streamy
`CATEGORY_IMAGE`	3	Obrázky, galéria, fotoaparát
`CATEGORY_SOCIAL`	4	Sociálne siete, komunikácia, e-maily
`CATEGORY_NEWS`	5	Noviny, časopisy, šport
`CATEGORY_MAPS`	6	Mapy, navigácia
`CATEGORY_PRODUCTIVITY`	7	Pracovná aplikácia, úložisko

Tabuľka 3.1: Ukážka použitých kategórií aplikácií podľa konštánt v [31]

3.1.5 Štatistiky používania aplikácií

Aplikácia si zaznamenáva históriu používania monitorovaných aplikácií. Z histórie je možné vytvoriť požadované štatistiky. V aplikácii sa zobrazujú štatistiky podľa názvu aplikácií alebo podľa kategórií aplikácií. Z oboch typov štatistík je možné zobraziť interval histórie za posledný deň, posledný týždeň a podledných 10 dní.

Štatistiky sa resetujú vždy o 00:00, a to každý deň, respektíve v pondelok pri týždňových štatistikách. V prípade potreby bude existovať možnosť zobraziť mesačné štatistiky alebo iné štatistiky, a to aj spätne využitím dát zo servera.

3.1.6 Rozvrh používania blokovaných aplikácií

Okrem prekročenia času na hranie sa môžu blokované aplikácie zablokovať v konkrétnych časových intervaloch bez ohľadu na reálne strávený čas v blokovaných aplikáciách. Každý rozvrh obsahuje:

- názov, ktorý nemusí byť unikátny;
- čas odkedy a dokedy je rozvrh aktívny;
- dni, kedy je rozvrh aktívny;
- status o tom, či je rozvrh aktivovaný.

Čas je uchovávaný s presnosťou na sekundy. Ak je čas začiatku platnosti rozvrhu neskôr ako čas konca platnosti rozvrhu, rozvrh je aktívny aj nasledujúci deň do konca platnosti rozvrhu bez ohľadu na to, či je rozvrh v nasledujúci deň aktívny. Aktívne dni sa uchovávajú ako jedna číselná hodnota, kde v binárnom zápise každý bit reprezentuje jeden deň, v poradí od pondelka od najmenej významného bitu[26].

Deň	Pondelok	Utorok	Streda	Štvrtok	Piatok	Sobota	Nedeľa
Konštanta	1	2	4	8	16	32	64

Tabuľka 3.2: Ukážka konštánt dní, ktorých ak je binárny súčin s reálnou hodnotou rovný rovnakej konštante, značí, že je rozvrh v daný deň aktívny

3.1.7 Push notifikácie

Android aplikácia bude synchronizovať obsah so serverom v pravidelných intervaloch. Pre zabezpečenie aktuálnosti obsahu je ale nutné notifikovať aplikáciu o prichádzajúcich zmenách zo strany servera, kde mohol rodič modifikovať obsah vo webovej aplikácii Educhild.

Potreba notifikovania je nutná najmä, ak sa Android aplikácia nachádza v detskom móde, kde je potrebné zaistiť aktuálne informácie o blokovaných aplikáciách a maximálnom čase používania blokovaných aplikácií dieťaťom. V rodičovskom móde aplikácie, t. j. ak rodič modifikuje nastavenia v Android zariadení, bude synchronizácia konkrétneho obsahu prebiehať pri prechode na konkrétnu obrazovku rodičom.

3.2 Analýza požiadaviek na server

V tejto časti sú na základe požiadaviek Android aplikácie a webovej aplikácie uvedené funkčné a nefunkčné požiadavky, nutné pre návrh a implementáciu servera. Požiadavky sú vždy zotriedené podľa priority od najvyššej po najnižšiu.

[26] v binárnom zápise bit umiestnený najviac vpravo

3.2.1 Funkčné požiadavky

Nasledujúce požiadavky špecifikujú funkcionality potrebné na realizáciu servera. Každá požiadavka má svojho aktéra, podmienky na danú funkcionalitu a identifikátor. Každá požiadavka má taktiež uvedenú prioritu v rámci projektu a predpokladanú zložitosť funkcionality.

F1: Správa užívateľa

Systém bude umožňovať užívateľovi registrovať sa v prípade, že v systéme nie je zaregistrovaný. Registrácia a prihlasovanie bude prebiehať na základe platného autentizačného tokenu Firebase, ktorý užívateľ obdrží na základe registrácie alebo prihlásenia na platforme Firebase. Užívateľ bude môcť pomocou servera kedykoľvek odstrániť účet vrátane všetkých údajov, vrátane účtu Firebase [32]. Po odstránení účtu sa bude možné opäť zaregistrovať.

Systém bude evidovať užívateľov podľa unikátneho identifikátora a e-mailu, poskytovaných platformou Firebase, ale nebude poskytovať zmenu e-mailu alebo hesla, ktorá je možná pomocou aplikačného rozhrania Firebase v klientskej aplikácii [33]. Užívateľ si bude môcť pri registrácii zvoliť typ konta, a to rodičovské na využívanie mobilnej aplikácie Educhild alebo autorské, ktoré bude slúžiť na tvorbu kvízov prostredníctvom webovej aplikácie Educhild, prípadne kombináciu oboch typov účtov.

Priorita: vysoká
Zložitosť: stredná

F2: Správa osobných údajov a preferencií

Rodič bude môcť na serveri pridávať a odoberať deti a spravovať ich údaje. Systém si bude uchovávať u každého dieťaťa údaje meno, mesiac a rok narodenia, maximálny možný čas strávený dieťaťom v aplikáciách označených ako blokované a profilovú fotografiu. Rodič bude môcť dieťatu nastaviť predvolenú blokáciu aplikácií po nainštalovaní do zariadenia alebo predvolenú blokáciu všetkých aplikácií s kategóriou typu sociálna sieť alebo hry. Rodič bude mať umožnené uplatniť svoje preferencie na všetky deti súčasne. Systém si bude uchovávať rodičom predvolený jazyk systému.

Priorita: vysoká
Zložitosť: stredná

F3: Správa aplikačnej histórie

Systém bude spravovať záznamy o navštívených aplikáciách dieťaťom v zariadení Android, nutné pre správu štatistík. Každý záznam musí obsahovať identifikačné údaje o konkrétnej aplikácii a časové vymedzenie užívania konkrétnej aplikácie.

Priorita: vysoká
Zložitosť: nízka

F4: Správa monitorovaných aplikácií

Systém bude evidovať každú aplikáciu na mobilnom zariadení užívanú dieťaťom, ktorá je určená k monitorovaniu. Každej aplikácii sa bude evidovať stav, či je aplikácia určená k blokovaniu a inštalačný stav. Android aplikácia bude automaticky odosielať informácie o nainštalovaných aplikáciách na server. Ak bude aplikácia odinštalovaná, systém odstráni údaje o aplikácii v prípade, že už nebude potrebná pre štatistické účely. Rodič bude môcť u dieťaťa upravovať blokovaný status aplikácie.

Priorita: vysoká
Zložitosť: nízka

F5: Správa rozvrhov pre používanie blokovaných aplikácií

Systém umožní rodičovi pridávať, modifikovať a odstraňovať rozvrhy, na základe ktorých dieťa nebude môcť používať aplikácie označené ako blokované v prípade aktívneho rozvrhu. Systém si bude zaznamenávať názov rozvrhu, čas a dni, v ktorom je rozvrh aktívny a status, či je rozvrh aktívny.

Priorita: vysoká
Zložitosť: nízka

F6: Prehľad štatistík

Rodič bude môcť prehliadať štatistiky o čase strávenom dieťaťom v aplikáciách. K dispozícii budú štatistiky podľa názvu aplikácie a podľa kategórie aplikácie. Každý typ štatistík bude dostupný v intervale dennom, týždennom a 10-dňovom. Štatistiky sa počítajú vždy od polnoci daného dňa, respektíve od začiatku týždňa v týždňových štatistikách. Systém musí byť navrhnutý na nenáročné doplnenie ďalšieho intervalu, v ktorom sa budú zobrazovať štatistiky.

Priorita: stredná
Zložitosť: stredná

F7: Aktualizácia štatistík

Systém bude automaticky spravovať štatistiky podľa histórie navštívených aplikácií v mobilnom zariadení dieťaťa tak, aby boli vždy aktuálne a zosynchronizované s aplikačnou históriou.

Priorita: stredná
Zložitosť: vysoká

F8: Notifikovanie o zmene nastavení dieťaťa

Systém bude odosielať Android aplikácii informáciu o zmene údajov v prípade, že údaje boli zmenené v inom klientovi, než v Android aplikácii. Systém bude odosielať iba údaje, ktorých zmena môže ovplyvniť používanie zariadenia dieťaťa, a to maximálny čas strávený dieťaťom v blokovaných aplikáciách, na-

stavenie automatického blokovania aplikácií, rozvrhy používania blokovaných aplikácií a zmena stavu blokovania monitorovanej aplikácie. Systém nebude odosielať notifikácie o zmene ostatných údajov.

Priorita: stredná
Zložitosť: vysoká

F9: Zber aplikačných detailov

Systém bude uchovávať detaily o Android aplikáciách dostupné z mobilného zariadenia, ktoré sú potrebné pre správne fungovanie webovej aplikácie. Všetky informácie o aplikáciách budú anonymné, t. j. systém nebude zaznamenávať, z ktorého zariadenia boli informácie sprostredkované. Systém ale môže v prípade budúcej potreby obmedziť prístup klientskému zariadeniu, ktoré bude odosielať nepravdivý alebo zámerne modifikovaný obsah. Systém bude uchovávať názov aplikácie, a to aj v rôznych jazykoch v prípade potreby, kategóriu aplikácie a ikonu aplikácie. Systém musí byť navrhnutý aj na získavanie informácií z iných zdrojov v prípade potreby.

Priorita: nízka
Zložitosť: nízka

3.2.2 Nefunkčné požiadavky

Nefunkčné požiadavky špecifikujú celkové nároky na fungovanie systému. Nasledujúce požiadavky stanovujú nároky na fungovanie servera.

N1: Autentizácia pomocou tokenov

Systém bude autentizovať užívateľov pomocou autentizačných tokenov sprostredkovaných platformou Firebase. Systém musí byť zameniteľný, v prípade potreby bude schopný zmeniť poskytovateľa autentifikačných služieb, eventuálne sa bude môcť postarať o autentizačný proces sám. Systém môže v prípade potreby klientov vytvárať vlastnú reláciu, uloženú do súborov cookies, schopnú na dlhší čas nahradiť autentizačný token.

N2: Zabezpečené pripojenie

Systém musí kvôli bezpečnosti a správnemu fungovaniu webového klienta zabezpečiť šifrované pripojenie pomocou protokolu HTTPS. Systém bude mať platný certifikát podpísaný certifikačnou autoritou a bude odolný voči útokom v internetových prehliadačoch.

N3: Asynchrónny tok požiadaviek

Systém musí zabezpečiť mechanizmus na spracovanie požiadaviek tak, aby požiadavky odoslané oneskorene oproti času vytvorenia, nemodifikovali obsah zmenený požiadavkami, ktoré boli vytvorené neskôr ako pôvodné požiadavky.

Respektíve, systém musí zabezpečiť, aby bola platná vždy najnovšie vytvorená požiadavka, bez ohľadu na čas odoslania.

N4: Konfigurovateľnosť aplikácie
Systém musí byť konfigurovateľný a spustiteľný na rôznych prostrediach bez nutnosti zásahu v zdrojovom kóde.

N5: Monitorovanie systému a sledovanie chýb
Systém bude monitorovať zdravie a stav aplikácie a bude hlásiť chyby a výnimky za behu programu. Systém poskytne výpis zásobníka a ďalšie detaily chyby.

N6: Prehľad koncových bodov
Systém bude poskytovať aktuálnu dokumentáciu o všetkých koncových bodoch a aplikačnom rozhraní systému. Dokumentácia bude dostupná online po autentizácii užívateľa.

3.3 Prípady použitia

Nasledujúce prípady použitia rozdeľujú funkčné požiadavky na jednotlivé úlohy v systéme, pomocou ktorých aktéri dosahujú v systéme istý cieľ. Každá funkčná požiadavka by mala byť realizovaná aspoň jedným prípadom použitia. Na záver podkapitoly je zhrnuté realizovanie funkčných požiadaviek prípadmi použitia a je zobrazený model prípadov použitia spolu s aktérmi, ktorí dané prípady použitia iniciujú.

UC1: Vytvorenie účtu
Umožňuje užívateľovi vytvoriť účet na serveri, ak doposiaľ neexistuje. Vytvorený účet je nutnou podmienkou na registráciu rodiča alebo autora kvízov. Pre vytvorenie účtu na serveri je nutné získať údaje z platformy Firebase na základe autentizačného tokenu.

UC2: Registrácia rodiča
Užívateľ sa zaregistruje ako rodič, a tým bude môcť využívať Android aplikáciu Educhild so vzdialenou správou. Po vytvorení rodičovského účtu by malo bezprostredne nadchádzať zaregistrovanie aspoň jedného dieťaťa.

UC3: Registrácia autora
Užívateľ sa zaregistruje ako autor kvízov, čím mu bude umožnené vytvárať a zdieľať nové kvízy pre aplikáciu Educhild.

UC4: Vymazanie účtu
Umožňuje vymazanie celého účtu užívateľom vrátane všetkých údajov, prípadne vymazanie rodičovskej alebo autorskej časti účtu jednotlivo. Pri vymazaní účtu sa taktiež vymaže užívateľov účet na platforme Firebase. Server si po vymazaní účtu neuchováva údaje užívateľa.

UC5: Vytvorenie dieťaťa
Umožňuje vytvorenie dieťaťa pre konkrétneho rodiča, a tým aj nastavenie rozvrhov, monitorovaných aplikácií a správu aplikačnej histórie, štatistík a kvízov dieťaťa. Na používanie Android aplikácie je nutné vytvorenie aspoň jedného dieťaťa.

UC6: Správa preferencií rodiča
Umožňuje nastaviť preferencie rodiča pri vytváraní nového dieťaťa. Preferencie sa automaticky nastavia po vytvorení prvého dieťaťa.

UC7: Správa dieťaťa
Umožňuje rodičovi zobraziť a upraviť nastavenia dieťaťa pre Android aplikáciu. Pri zmene nastavení z webovej aplikácie bude zaslaná notifikácia Android zariadeniu.

UC8: Vymazanie dieťaťa
Rodič vymaže všetky údaje o dieťati vrátane jeho nastavení. Po vymazaní nie je možné obnoviť údaje. Pri vymazaní dieťaťa z webovej aplikácie sa odošle notifikácia Android zariadeniu. Po odstránení všetkých detí nie je možné využívať Android aplikáciu.

UC9: Pridanie záznamu do aplikačnej histórie
Zariadenie zašle záznam o aktivite dieťaťa v monitorovanej aplikácii na server, kde sa záznam uloží a aktualizuje aplikačné štatistiky.

UC10: Čítanie záznamov aplikačnej histórie
Umožňuje synchronizáciu záznamov po prihlásení rodičom do Android aplikácie a poskytuje detailný prehľad o aktivite dieťaťa vo webovej aplikácii.

UC11: Pridanie monitorovanej aplikácie
Android zariadenie po registrácii alebo pri nainštalovaní novej aplikácie do zariadenia dieťaťom odošle nový záznam o monitorovanej aplikácii na server spolu s počiatočným stavom blokovania aplikácie podľa predvolených nastavení.

UC12: Úprava blokovaného stavu monitorovanej aplikácie
Rodič upraví stav blokovania aplikácie vo webovej aplikácii alebo priamo v Android aplikácii. Ak rodič upraví stav vo webovej aplikácii, server odošle notifikáciu Android zariadeniu o zmene stavu.

UC13: Pridanie rozvrhu pre používanie blokovaných aplikácií
Rodič vytvorí dieťaťu rozvrh, počas ktorého mu nebude umožnené používať aplikácie označené ako blokované. Rodič môže pridať rozvrh, ktorý sa časovo prekrýva s iným rozvrhom. Rodič nemôže pridať rozvrh, ktorý nie je aktívny žiadny deň. Ak rodič vytvoril nový rozvrh vo webovej aplikácii, server odošle Android zariadeniu notifikáciu o pridaní nového rozvrhu dieťaťu.

UC14: Správa rozvrhu pre používanie blokovaných aplikácií
Umožňuje rodičom zobrazovať a upravovať rozvrh dieťaťa, počas ktorého mu nebude umožnené používať aplikácie označené ako blokované. Ak rodič upravil rozvrh vo webovej aplikácii a obsah na serveri sa modifikoval, server odošle Android zariadeniu notifikáciu o zmene rozvrhu.

UC15: Vymazanie rozvrhu pre používanie blokovaných aplikácií
Rodič vymaže dieťaťu rozvrh, počas ktorého mu nebude umožnené používať aplikácie označené ako blokované. Ak rodič vymazal rozvrh vo webovej aplikácii a rozvrh sa na serveri vymazal, server odošle Android zariadeniu notifikáciu o odstránení rozvrhu.

UC16: Zobrazenie štatistík používania aplikácií
Rodič si zobrazí štatistiky o používaní aplikácií podľa názvu aplikácií alebo podľa kategórií, a to za súčasný deň, týždeň alebo 10 dní.

UC17: Mazanie štatistík a monitorovaných aplikácií
Systém pravidelne premaže neaktuálne štatistiky používaných aplikácií dieťaťa a záznam o aplikácii, ak aplikácia už nie je v Android zariadení nainštalovaná a nie je naďalej potrebná pre účely zobrazovania a spravovania štatistík.

UC18: Synchronizácia štatistík
Systém pravidelne zosynchronizuje všetky štatistiky tak, aby boli v súlade so záznamami z aplikačnej histórie a zobrazovali údaje z časového obdobia, na ktoré sú určené.

UC19: Odoslanie notifikácie o zmene nastavení aplikácií
Umožňuje odosielanie notifikácii Android zariadeniu o modifikácii údajov na serveri.

UC20: Pridanie záznamu o aplikačných detailoch

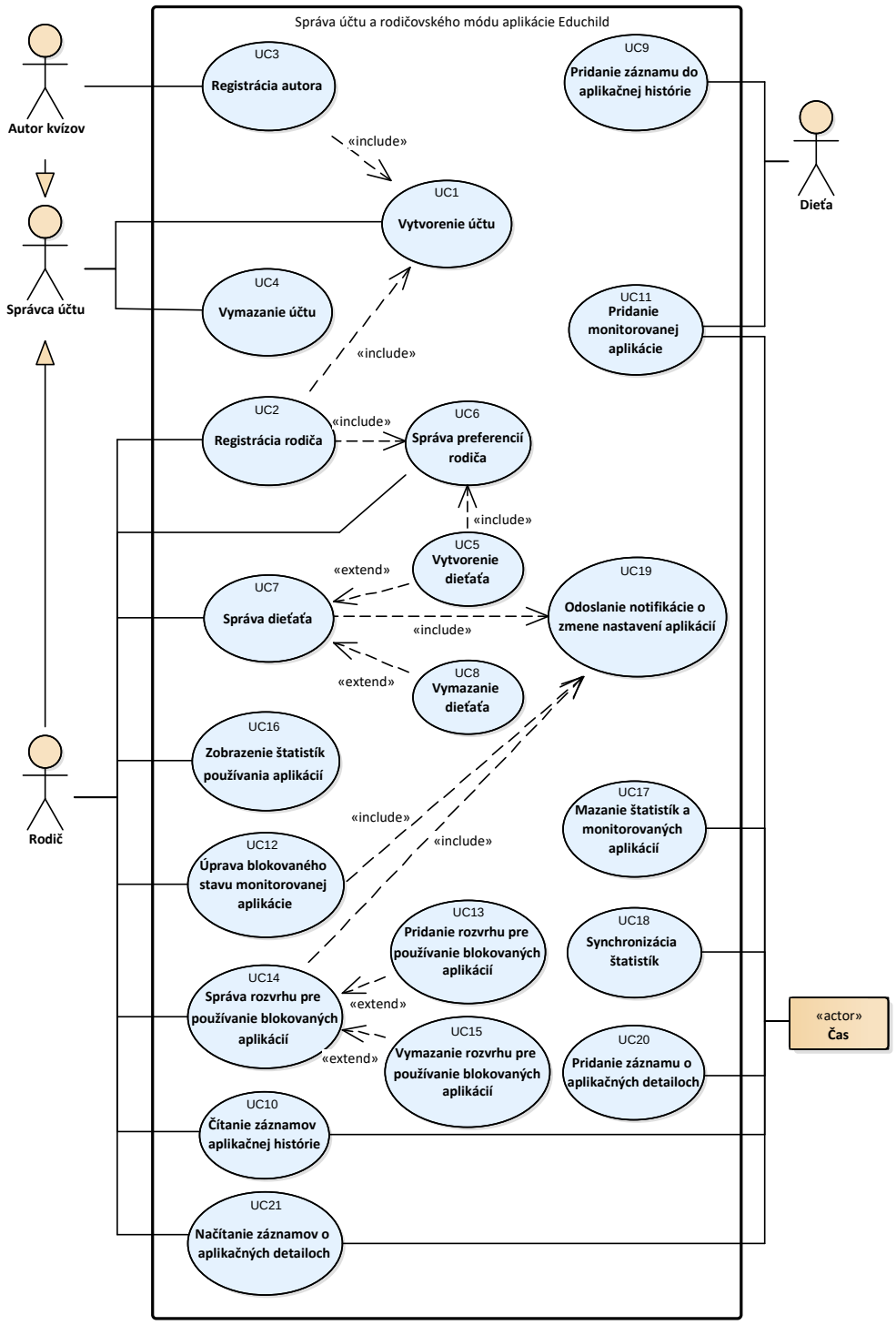
V prípade, že sa v systéme nenachádzajú detaily o aplikácii, ktorou Android zariadenie disponuje, systém pridá informácie o danej aplikácii, čím môže webová aplikácia správne zobraziť obsah a systém môže správne zaznamenávať štatistiky podľa kategórie aplikácie.

UC21: Načítanie záznamov o aplikačných detailoch

Umožňuje načítanie aplikačných detailov v prípade, že sa rodič nachádza vo webovej aplikácii, ktorá bežne nemá prístup k informáciam, ktorými Android zariadenia disponujú. Údaje môžu použiť aj niektoré typy Android zariadení, ktoré nemajú prístup k niektorým informáciám o aplikáciách.

	F1	F2	F3	F4	F5	F6	F7	F8	F9
UC1	+								
UC2	+								
UC3	+								
UC4	+								
UC5		+							
UC6		+							
UC7		+							
UC8		+							
UC9			+						
UC10			+						
UC11				+					
UC12				+					
UC13					+				
UC14					+				
UC15					+				
UC16						+			
UC17							+		
UC18							+		
UC19								+	
UC20									+
UC21									+

Tabuľka 3.3: Kontrola splnenia všetkých funkčných požiadaviek



Obrázok 3.1: Model prípadov použitia

Kapitola 4

Návrh a realizácia

Táto kapitola je venovaná návrhu a realizácii servera a komunikácie so serverom v Android aplikácii. V prvej časti je ukázaná základná architektúra servera a sú popísané komponenty aplikácie. V ďalších častiach sú ukázané návrhy databázy a aplikačného rozhrania webových služieb. Následne je popísaná tvorba samotného aplikačného servera a v podkapitole DevOps je popísaná forma vývoja servera. Na záver je ukázaný návrh a realizácia časti Android aplikácie súvisiacej so serverom.

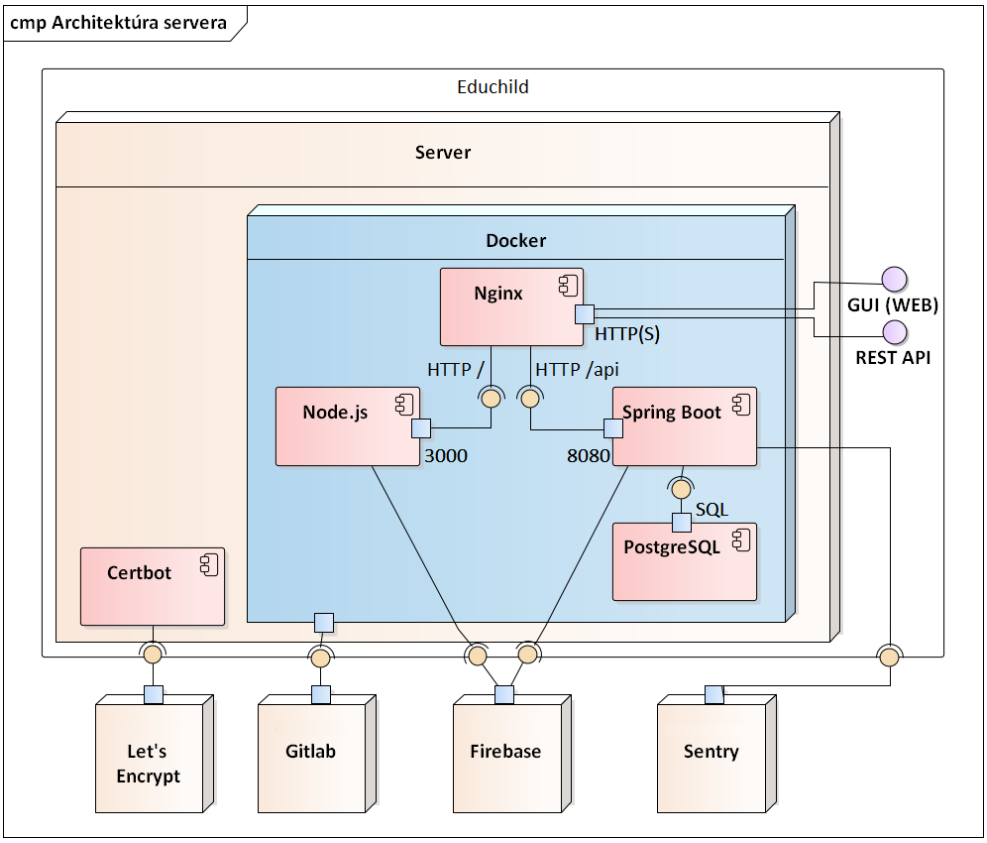
4.1 Architektúra servera

Táto podkapitola je venovaná základnému návrhu architektúry servera. V tejto časti sú popísané jednotlivé komponenty aplikácie Educhild nutné pre chod servera, prepojenie jednotlivých komponentov, externé komponenty a zabezpečenie servera. Pri jednotlivých komponentách sú taktiež popísané základné princípy fungovania komponenty, nutné pre pochopenie architektúry.

4.1.1 Docker

V počiatočnej fáze bol projektu poskytnutý jeden virtuálny serverový hosting s operačným systémom CentOS, v ktorom musia byť umiestnené všetky komponenty. Preto bola využitá platforma Docker, ktorá umožňuje rozdeliť aplikácie do kontajnerov spúšťaných v izolovanom prostredí [34]. Použitím kontajnerov sa zjednocujú používané verzie a inštalácia jednotlivých aplikácií v produkčnom, testovacom a lokálnom prostredí. Docker využíva funkčnosti jadra OS Linux a nepotrebuje pre beh vlastný virtuálny stroj, vďaka čomu je nenáročný na systémové prostriedky [35].

Každému kontajneru môžu byť sprístupnené siete, vďaka ktorým môže komunikovať s ostatnými kontajnermi. Pre komunikáciu kontajnera s externým prostredím je nutné špecifikovať konkrétne exponované porty, čo robí aplikáciu bezpečnejšou. Pre konfiguráciu všetkých kontajnerov súčasne slúži nástroj



Obrázok 4.1: Architektúra servera

Docker Compose, ktorý dokáže spravovať súčasne všetky kontajnery definované a nakonfigurované v rámci súboru typu YAML, čo je formát pre serializáciu štrukturovaných dát. [36] Konfigurácia bola riešená v spolupráci s Petrom Šímom v rámci jeho bakalárskej práce zameranej na detskú časť aplikácie Educhild. Testovacia konfigurácia nástroja je zobrazená v dodatku C.

4.1.2 PostgreSQL

Jedným z použitých kontajnerov na serveri je relačná databáza PostgreSQL. Pre potreby aplikácie Educhild nie je prvoradý výber databázy, a preto nebol kladený dôraz na výber databázy. Databáza bola vybraná na základe jej licencie FOSS[27]. Ďalšou uvažovanou databázou bola databáza MySQL, ktorá je pre potreby aplikácie dostačujúca, ale napokon bola zvolená databáza PostgreSQL pre jej podporu objektových typov a rozsiahlejšiu podporu indexov. [37]

[27] slobodný a otvorený softvér, angl. free and open-source software

Pre vytvorenie databázy sa využíva oficiálny obraz z úložiska Docker Hub. Všetky súbory databázy musia byť uložené aj po zastavení Docker kontajnera. Pre tento účel sa používa mechanizmus Docker Volume, ktorý uloží zvolené súbory do konkrétnych perzistentných súborov alebo do anonymných súborov, vygenerovaných pri spustení. Pre databázu, ktorej súbory nie je nutné priamo spravovať, je využitý anonymný súbor.

4.1.3 Spring Boot

Pre obsluhu klientov a spracovávanie požiadaviek sa používa aplikačný rámec Spring Boot. Výber vhodného aplikačného rámca pre webový server sa podrobne analyzuje v kapitole 1. Webový server v súčasnosti obsluhuje požiadavky webového klienta a Android aplikácie. Jeho základné funkcionality sú autentizácia a autorizácia, spracovanie dát, odosielanie notifikácií Android zariadeniu, vykonávanie načasovaných synchronizačných úloh. Aplikačný rámec používa vnorený webový server Tomcat.

Pre vytvorenie obrazu webového servera pre Docker sa používa skriptovací súbor nazývaný `Dockerfile`, v rámci ktorého sa zadefinujú príkazy potrebné pre zostavenie a spustenie aplikácie. Obraz sa sťahuje priamo z platformy GitLab, kde sa v rámci DevOps zostavuje aktuálny obraz aplikačného rámca. Pre zostavenie a správu aplikácie sa používa nástroj Maven, ktorý je typickým nástrojom pre vývoj v jazykoch založených na jazyku Java. Výberu vhodného nástroja však nebola na začiatku vývoja venovaná dostatočná pozornosť a v súčasnosti by bol uprednostnený nástroj Gradle, ktorý je jednoduchší, rýchlejší [38] a jeho použitím by sa zjednotilo použitie zostavovacieho nástroja s vývojom Android aplikácie Educhild, čím by sa uľahčila práca vývojárov.

4.1.4 Nginx

V rámci jednej domény musí server prerozdeliť požiadavky smerujúce na webový server s aplikačným rámcom Spring Boot, zaobstarávajúci biznis logiku aplikácie, zatiaľ čo požiadavky smerujúce na platformu Node.js spracúvajú požiadavky webového klienta. Pre tento účel musel byť na serveri zavedený reverzný proxy server, ktorý dokáže prerozdeliť požiadavky na rôzne platformy. Požiadavky smerujúce na aplikačný rámec Spring Boot musia začínať cestou `/api`, ostatné požiadavky smerujú na platformu Node.js.

Pre účely projektu bol vybraný webový server Nginx pre jeho – podľa vývojára – jednoduchšiu konfiguráciu oproti webovému serveru Apache, hlavným konkurentom platformy Nginx. Aplikácia však v súčasnosti využíva minimálne funkcionality tohto webového servera, a preto na výber reverzného proxy servera nie je kladený dôraz. V budúcnosti môže byť využitý webový server Nginx taktiež pre kešovanie alebo v prípade úspešnosti projektu pre vyvažovanie záťaže, kde sa rozloží záťaž na jednotlivé servery. [39]

Nginx ako koncový bod musí zaobstarať dešifrovanie a šifrovanie komunikácie pomocou protokolu HTTPS. K tomu je nutný SSL alebo TLS certifikát podpísaný dôveryhodnou autoritou. Aplikácia využíva bezplatné certifikáty vydané certifikačnou autoritou Let's Encrypt, ktoré je nutné obnovovať každé 3 mesiace. Obnova sa vykonáva automaticky programom Certbot pomocou naplánovaných úloh v systéme (tzv. cron job). Server automaticky presmerováva nezašifrovanú komunikáciu pomocou protokolu HTTP na port 443, t. j. na šifrované pripojenie pomocou protokolu HTTPS. Okrem toho webový server zabezpečuje jednoduché overenie prístupu[28], ktoré je potrebné pre webové rozhranie na testovacom prostredí a pre zabezpečenie prístupu do dokumentácie koncových bodov.

V počiatočnej architektúre servera nebol zahrnutý reverzný proxy server ani obsluha webového klienta. Z toho dôvodu muselo byť šifrovanie požiadaviek zabezpečené aplikačným rámcom Spring Boot, čo viedlo v konečnom dôsledku k zbytočnej implementácii. Zabezpečené pripojenie bolo nakoniec z rámca odstránené z dôvodu jeho nepotrebnosti pri komunikácii v lokálnej sieti (t. j. medzi kontajnerom obsahujúcim Nginx a Spring Boot). Rovnako došlo k odstráneniu implementácie jednoduchého overenia prístupu, ktoré zabezpečuje webový server Nginx.

4.1.5 Node.js

Prostredie Node.js zabezpečuje webové rozhranie aplikácie Educhild. Naň smerujú všetky požiadavky nezačínajúce cestou `/api`. Webovej aplikácii sa ďalej venuje bakalárska práca Jana Stejskala.

4.2 Databázový model

Tvorba databázového modelu je jednou z najdôležitejších činností pri tvorbe softvérového systému. Kvalitný model predurčuje návrh webového servera a čiastočne aj návrh aplikačného rozhrania koncových bodov. Databázový model znázorňuje logickú štruktúru databázy, vzťahy medzi entitami a obmedzenia určujúce spôsob ukladania a pristupovania k dátam. Databázový model aplikácie zobrazujúci najdôležitejšie entity a vzťahy v rámci administratívnej a rodičovskej časti aplikácie je znázornený v dodatku D. Návrh databázy prebiehal v súlade s modelom v Android aplikácii, popísaným v podkapitole 3.1.

Databázový model slúži taktiež na následné vygenerovanie SQL skriptu obsahujúci jazyk pre definíciu dát[29], z ktorého sa vytvárajú databázové objekty. Keďže sa prvotná verzia databázy vytvárala spolu s webovým aplikačným rámcom, bol pre vytvorenie databázy využitý nástroj Hibernate, ktorý je využívaný aplikačným rámcom Spring Boot na objektovo-relačné mapovanie.

[28] angl. basic access authentication
[29] angl. Data Definition Language, DDL

Nástroj Hibernate automaticky z mapovania vygeneruje SQL skript pre vytvorenie schémy databázy. Vďaka tomuto riešeniu je zredukované množstvo písania nadbytočného kódu a je znížená chybovosť vývojárov. Toto riešenie však nie je odporúčané využiť ďalej pri migrácii databázy na novšie verzie na produkčnom prostredí [40], a preto budú pri migrácii databázy použité migračné skripty.

Jednou z najdôležitejších tabuliek databázy je `user_details`, ktorá ukladá záznamy každého užívateľa a umožňuje ho identifikovať. Na identifikáciu sa v súčasnosti využíva unikátny identifikátor z platformy firebase. Každému užívateľovi je momentálne umožnené uložiť jeden token slúžiaci na odosielanie správ konkrétnemu zariadeniu o nutnosti aktualizácie údajov. V budúcnosti pri plánovaní použitia jedného účtu vo viacerých zariadeniach bude umožnené užívateľovi uložiť viacero tokenov súčasne. Z platformy Firebase je taktiež poskytnutý e-mail a meno užívateľa, ktoré sa ukladajú do internej databázy pre možnosť prípadného odosielania e-mailov alebo zmenu poskytovateľa autentizácie.

Každý užívateľ môže mať následne vytvorený jeden záznam v tabuľke `author` a v tabuľke `parent`, podľa toho, či sa užívateľ rozhodne vytvárať kvízy alebo využívať Android aplikáciu, alebo oboje súčasne. V pôvodnom návrhu boli údaje o užívateľovi zlúčené v tabuľke s údajmi o rodičovi, čím nebolo možné rozlíšiť aktéra a jeho záujmy v aplikácii. Rodič si ďalej bude môcť vytvárať v aplikácii neobmedzené množstvo detí, ktorým sa budú zaznamenávať ich nastavenia, rozvrhy, monitorované aplikácie a štatistiky.

Do tabuliek, ktorých modifikácia dát je možná z viacero zdrojov súčasne (konktétne `parent`, `child`, `timetable`, `monitored_application`), bol pridaný údaj `updated` značiaci čas poslednej modifikácie pre znemožnenie prepísania aktuálnych dát starými dátami. Tento problém bude ďalej riešený v podkapitole 4.4. Z bezpečnostných dôvodov – ďalej riešených v podkapitole 4.3 – bol entitám `child`, `timetable` a `application_history` pridelený univerzálny unikátny identifikátor[30] pre externú komunikáciu. Tento identifikátor – rovnako ako napríklad atribút `package_name` v tabuľke `monitored_application` – mohol byť použitý ako primárny kľúč, ale použitie dátového typu varchar ukladajúceho znakové reťazce a dátového typu uuid ako primárneho kľúča nie je vhodné z výkonnostných dôvodov a zvyšuje nároky pri reorganizovaní indexov tabuľky. Z toho dôvodu je oproti prvotnému návrhu databázy vo všetkých tabuľkách ako primárny kľúč preferovaný automaticky inkrementovaný numerický dátový typ.

V počiatočnom návrhu boli autorom preferované zložené primárne kľúče z dôvodu lepšej organizácie dát (napríklad relácia dieťaťa mala vo svojom primárnom kľúči zahrnutý primárny kľúč z tabuľky rodiča, atď.). V konečnom dôsledku to viedlo k tomu, že niektoré entity mali primárny kľúč zložený až zo štyroch údajov, čo negatívne ovplyvnilo zložitosť modela a taktiež tabuľky

[30] angl. Universally unique identifier, UUID

dekomponujúce entitné vzťahy M:N, ktoré obsahovali primárny kľúč zložený z väčšieho množstva atribútov. Z toho dôvodu boli všetky zložené primárne kľúče premodelované na jednoduché primárne kľúče.

V prvotných návrhoch bolo úsilie znížiť počet dát obsiahnutých v tabuľke pre rozvrh dieťaťa na hranie tým, že rovnaké rozvrhy budú spoločné pre viacero detí, čím vznikla väzba typu M:N medzi dieťaťom a rozvrhom. To sa však ukázalo ako neefektívne riešenie, keďže pri editácii údajov sa musel vytvárať nový záznam a vznikali zbytočné výpočetné operácie pri kontrole existencie väzby rozvrhu s nejakým dieťaťom po tom, čo bola nejaká väzba odstránená. Po tom, čo sa do rozvrhu pridal atribút názov rozvrhu – čím sa každý rozvrh stal jedinečným –, nebolo možné ďalej udržiavať väzbu M:N. Tým sa ukázalo, že dané riešenie nebolo pripravené pre budúce zmeny a v súčasnosti má každé dieťa pridelené unikátne záznamy o rozvrhu. V ďalšom návrhu mal atribút pre názov rozvrhu pridelené obmedzenie UNIQUE určujúce unikátnosť pre každé dieťa. Tento návrh sa ukázal byť nesprávny po tom, čo sa v Android aplikácii nastavovali implicitne rovnaké názvy pre všetky rozvrhy. Časť prvotného návrhu je zobrazená na obrázku 4.2.

V súčasnom databázovom modeli sa nachádzajú vo všetkých typoch štatistík údaje o strávenom čase dieťaťa pri používaní konkrétnej aplikácie alebo kategórie. V databáze sa napriek tomu tiež uchováva celkovo strávený čas za dané obdobie. Tento údaj je možné dopočítať z ostatných údajov, a teda je redundantný a teoreticky môžu vznikať aktualizačné anomálie. Z praktických dôvodov je ale nutné sprostredkovať užívateľovi údaje o celkovo strávenom čase v čo najkratšom okamihu, rovnako ako nie je žiaduce opätovné prepočítavanie rovnakej hodnoty. Na aplikačnej úrovni webového servera je zabezpečené okamžité prepočítanie celkového času pri pridávaní a odoberaní konktrétnych záznamov, a teda sú tieto redundantné údaje zakaždým aktualizované.

4.3 Návrh aplikačného rozhrania webových služieb

Táto podkapitola je venovaná návrhu aplikačného programového rozhrania webových služieb aplikačného rámca Spring Boot. Prvotný návrh API bol riešený pred samotným návrhom webového servera, aby reflektoval potreby zo strany klienta a nebol obmedzovaný samotným návrhom servera. Pre tvorbu rozhrania bol využitý architektonický štýl REST[31]. Ďalšou uvažovanou metódou pre tvorbu webových služieb bol protokol SOAP, ktorý má oproti princípu REST okrem iného vyššiu latenciu a vyššie nároky na spracovanie v mobilných zariadeniach [41]. Platforma Android navyše poskytuje množstvo knižníc určených primárne pre prácu s RESTful webovými službami, ako aj s formátom JSON, ako sú napríklad Retrofit, Volley, Moshi alebo Gson. Obsah tela požiadaviek a odpovedí správ bude primárne riešený formátom JSON, hoci RESTful

[31] Representational State Transfer

Obrázok 4.2: Ukážka časti prvotného databázového návrhu. Tabuľky majú zložené primárne kľúče, tabuľka rozvrhu je vo vzťahu s tabuľkou dieťaťa typu M:N a tabuľka rodiča obsahuje taktiež všeobecné informácie o užívateľovi.

webové služby môžu umožňovať prijímanie a odosielanie správ vo viacerých formátoch.

Ak webové služby implementujú architektonický vzor REST, nazývame tieto služby RESTful. REST API je špecifické tým, že každý zdroj je jed-

noznačne určený svojím URI[32], ktorý reprezentuje daný zdroj. Každý zdroj potrebuje pre prácu s dátami realizovať operácie CRUD[33]. Tieto operácie sú realizované protokolom HTTP pomocou metód POST pre inicializáciu nového zdroja, GET pre získanie aktuálneho stavu zdroja, PUT pre vytvorenie alebo aktualizáciu zdroja a DELETE pre odstránenie zdroja. Pre realizáciu REST API je taktiež možné využiť aj ostatné HTTP metódy, ako napríklad PATCH pre čiastočnú úpravu stavu zdroja. Pre reprezentáciu vzťahov medzi zdrojmi sa využívajú odkazy v odpovediach, jedná sa o tzv. princíp HATEOAS[34]. [42]

RESTful webové služby sú bezstavové. Keďže je celé API webových služieb aplikácie Educhild privátne, je nutné každú požiadavku autentizovať. V aplikácii budú existovať zdroje, ktoré musia byť dostupné taktiež bez registrovaného užívateľa, ako napríklad zistenie informácií o nainštalovaných aplikáciách zo servera alebo sťahovanie kvízov v rámci detskej časti aplikácie. Aby tieto zdroje neboli dostupné verejne, bol zavedený tzv. API kľúč[35], ktorý je nutné zadať do HTTP hlavičky `X-API-KEY`. Autentizácii pomocou API kľúča sa ďalej venuje bakalárska práca Petra Šímu. Pre ostatné zdroje, ktoré vyžadujú zaregistrovaného užívateľa, bude autentizácia riešená pomocou platfotmy Firebase, v ktorej sú uložené prihlasovacie údaje užívateľov, ktorá vygeneruje autentizačný token, ktorý je nutné vložiť do HTTP hlavičky s kľúčom `X-Authentication-Firebase`.

Návrh URI je dôležitou súčasťou tvorby REST API. Každé URI má vhodne reprezentovať svoje zdroje a umožňovať vykonať požadované metódy. Kvalitný návrh by mal byť nezávislý od danej platformy (t. j. od webového alebo mobilného klienta) a mal by byť mať možnosť sa vyvíjať a pridávať funkcie nezávisle na klientských aplikáciách. [43] V nasledujúcej časti je zobrazený návrh REST API správcovskej a rodičovskej časti. Vždy je zobrazená HTTP metóda, URI a popis služby, prípadne viacero služieb súčasne.

`POST /parents`
`POST /authors`

Vytvorí nový rodičovský, resp. autorský účet na základe autentizačného tokenu. V prípade, že je účet už vytvorený, systém vráti údaje existujúceho rodiča alebo autora. Ak v systéme nie je evidovaný užívateľský účet, systém vytvorí daný účet.

`GET /parents/me`
`GET /authors/me`

Vráti existujúci účet rodiča, resp. autora na základe autentizačného tokenu. V prípade neexistujúceho účtu bude vrátený status 404.

[32] jednotný identifikátor zdroja, angl. Uniform Resource Identifier
[33] operácie vytvorenia, čítania, úpravy a odstránenia
[34] Hypermedia as the Engine of Application State
[35] identifikátor pre overenie klienta volajúceho API

`PUT /parents/me`
`PUT /authors/me`
Upraví entitu existujúceho rodiča, resp. autora na základe tela požiadavky vo formáte JSON.

`DELETE /parents/me`
`DELETE /authors/me`
Vymaže rodičovský, resp. autorský účet, ak existuje, vrátane všetkých údajov a nastavení detí rodiča alebo autora. Nevymaže sa užívateľský účet.

`POST /parents/me/children`
Vytvorí nové dieťa podľa tela zadanom v požiadavke. V odpovedi vráti telo s údajmi o dieťati vrátane identifikátora slúžiaceho pre ďalšiu komunikáciu.

`GET /parents/me/children`
Vráti zoznam všetkých detí daného rodiča.

`GET /children/{childId}`
Vráti údaje dieťaťa s identifikátorom vyplneným v parametri `childId`, ak existuje.

`PUT /children/{childId}`
Upraví entitu existujúceho dieťaťa na základe tela požiadavky.

`DELETE /children/{childId}`
Vymaže dieťa zo systému, ak existuje, vrátane všetkých jeho údajov a nastavení.

`POST /children/{childId}/image`
Nahrá do systému profilovú fotografiu dieťaťa a vymaže predchádzajúcu fotografiu, ak exituje. Podporovaný formát fotografií je png a jpeg. Fotografia sa vkladá ako parameter `image` s typom MIME.

`GET /children/{childId}/image`
Vráti profilovú fotografiu dieťaťa s jej typom, ak je v systéme nahraná.

`DELETE /children/{childId}/image`
Odstráni profilovú fotografiu zo systému, ak je v systéme nahraná.

`POST /children/{childId}/timetables`
Pridá dieťaťu nový rozvrh na blokovanie používania zvolených aplikácií podľa tela správy, ktorý je jedinečný pre dieťa. V odpovedi vráti vytvorený rozvrh s údajmi o dieťati vrátane identifikátora slúžiaceho pre ďalšiu komunikáciu.

`GET /children/{childId}/timetables`

Vráti zoznam všetkých rozvrhov na blokovanie používania zvolených aplikácií, ktoré sú pridelené danému dieťaťu.

`GET /timetables/{timetableId}`

Vráti konkrétny rozvrh dieťaťa podľa identifikátora v parametri `timetableId`, ak existuje.

`PUT /timetables/{timetableId}`

Upraví entitu existujúceho rozvrhu na základe tela požiadavky.

`DELETE /timetables/{timetableId}`

Odstráni rozvrh zo systému podľa identifikátora `timetableId`, ak existuje, a tým ho aj odstráni zo zoznamu rozvrhov daného dieťaťa.

`POST /children/{childId}/applications`

Pridá dieťaťu novú Android aplikáciu určenú na monitorovanie alebo upraví vlastnosti existujúceho záznamu.

`GET /children/{childId}/applications?page={page}&size={size}`

Vráti zoznam evidovaných Android aplikácií určených na monitorovanie. V požiadavke je možné zadať parameter `page` pre číslo strany určenej na zobrazenie a parameter `size` určujúci počet entít zobrazených v rámci jednej požiadavky. Predvolený počet je 200 entít na jednu požiadavku.

`PUT /children/{childId}/applications/{packageName}`

Upraví entitu existujúcej monitorovanej aplikácie, identifikovanú podľa unikátneho názvu balíka aplikácie.

`DELETE /children/{childId}/applications/{packageName}`

Odstráni dieťatu záznam o monitorovanej aplikácii, ak existuje.

`POST /children/{childId}/applications/history`

Pridá záznam o strávenom čase v aplikácii na mobilnom zariadení.

`GET /children/{childId}/applications/history`
`/{date}?page={page}&size={size}`

Vráti záznamy o strávenom čase v aplikáciách z istého dňa. Rovnako ako pri listovaní zaevidovaných Android aplikácií, sa k záznamom pristupuje pomocou stránkovania.

```
GET /children/{childId}/statistics/daily?page={page}&size={size}
GET /children/{childId}/statistics/daily/{date}
GET /children/{childId}/statistics/week
GET /children/{childId}/statistics/ten_days
```

Požiadavky vracajú informácie o strávenom čase v aplikáciách za posledné dni (stránkovane), konkrétny deň, posledný týždeň alebo posledných 10 dní.

```
GET /children/{childId}/statistics/daily/{date}/category
GET /children/{childId}/statistics/daily/{date}/category
    /{categoryId}
GET /children/{childId}/statistics/daily/{date}/application
GET /children/{childId}/statistics/daily/{date}/application
    /{packageName}
GET /children/{childId}/statistics/week/category
GET /children/{childId}/statistics/week/category/{categoryId}
GET /children/{childId}/statistics/week/application
GET /children/{childId}/statistics/week/application/{packageName}
GET /children/{childId}/statistics/ten_days/category
GET /children/{childId}/statistics/ten_days/category/{categoryId}
GET /children/{childId}/statistics/ten_days/application
GET /children/{childId}/statistics/ten_days/application
    /{packageName}
```

Požiadavky vracajú stránkované alebo konkrétne záznamy o štatistikách, koľko času strávilo dieťa v Android aplikáciach. Časy v aplikáciách sú agregované podľa konkrétnej aplikácie alebo kategórie aplikácie podľa tabuľky 3.1 za dané obdobie (t. j. za deň, týždeň alebo 10 dní).

```
POST /applications/{packageName}?lang={lang}
```

Vloží do systému všeobecné informácie o aplikácii v danom jazyku.

```
POST /applications/{packageName}/image
```

Vloží do systému ikonu aplikácie.

```
GET /applications/{packageName}?lang={lang}
```

Vráti informácie o konkrétnej aplikácii v danom jazyku, ak sa v systéme takéto údaje nachádzajú. Alternatívne sa vrátia informácie v jazyku zariadenia užívateľa, ak je požiadavka autentizovaná užívateľom, prípadne informácie v anglickom jazyku.

```
GET /applications/{packageName}/image
```

Vráti ikonu danej aplikácie, ak je ikona v systéme uložená.

`DELETE /accounts/me`
Odstráni celý účet vrátane rodičovskej a autorskej časti, všetkých údajov a nastavení, ako aj autentizačné údaje uložené na platforme Firebase.

`POST /accounts/me/firebase/token`
Zaregistruje k účtu token určený pre push notifikácie do Android zariadenia.

V každej odpovedi je odoslaný stavový kód HTTP podľa jeho štandardného významu. Medzi najdôležitejšie používané stavové kódy patria `200 OK`, ktorý sa použije v prípade úspešnej metódy GET, PUT alebo DELETE; stavový kód `201 Created` je použitý v prípade úspešnej metódy POST. Ak požadovaná entita nebola nájdená, je vrátený stavový kód `404 Not Found`. V prípade, že nebol zadaný platný autentizačný token, je zakaždým vrátený kód `403 Forbidden`.

Vo finálnom návrhu sa nachádzajú zdroje, ktoré vo svojom identifikátore obsahujú cestu `/me` (`/accounts/me`, `/parents/me`, `/authors/me`). Táto cesta referuje na užívateľa, ktorý bol identifikovaný na základe autentizačného tokenu. V prvotných návrhoch sa namiesto cesty `/me` používal numerický identifikátor. Tento identifikátor sa ukázal byť redundantný a zvyšujúci zložitosť návrhu, pretože kazdý užívateľ môže vlastniť maximálne jeden účet každého typu, a teda je na základe súčastnej cesty jednoznačne identifikovateľný.

V raných riešeniach návrhu sa do premenných v zdroji slúžiace ako identifikátor vkladali numerické hodnoty, ktoré boli získavané z primárnych kľúčov v tabuľkách databázy. Primárne kľúče sú generované databázou inkrementálne, čím dokáže zistiť potenciálny útočník identifikácie ostatných užívateľov, čo ohrozuje zabezpečenie systému. Okrem toho bolo možné získať prehľad o aktuálnom stave počtu entít v databáze, čo môže mať z biznisového hľadiska negatívny dopad. V novších verziách sa ako identifikátor v zdrojoch využíva unikátny identifikátor UUID, zložený z náhodne vygenerovaného 128-bitového čísla. Nevýhodami tohto riešenia sú zhoršená čitateľnosť (API ale nie je určené pre koncového užívateľa) a možnosť kolízie, ktorá je ale extrémne nízka a je ošetrená na databázovej úrovni pridelením obmedzenia o unikátnosti.

Keďže je správa užívateľov riešená platformou Firebase, teoreticky nie je nutná registrácia užívateľa na internom serveri a užívateľ sa zaeviduje v databáze až v prípade prvej odoslanej požiadavky na rodiča alebo autora. To reflektovalo aj prvotný návrh riešenia API, v ktorom vykonávali metódy GET a POST pri dopytovaní rodiča rovnakú logiku. Toto riešenie znemožňovalo evidenciu využívania jednotlivých fragmentov aplikácie užívateľmi a vnášalo priestor vzniku chýb, keďže pri vyžadovaní objektu sa objekt mohol práve vytvoriť. Tento návrh taktiež porušoval zásady RESTful návrhu. V súšasnej verzii je vyžadované explicitné vytvorenie rodičovského alebo autorského účtu pomocou metódy POST.

Dokumentácia API je riešená pomocou aplikačného rámca Swagger, ktorý automaticky vygeneruje dokumentáciu na základe implementácie ovládača

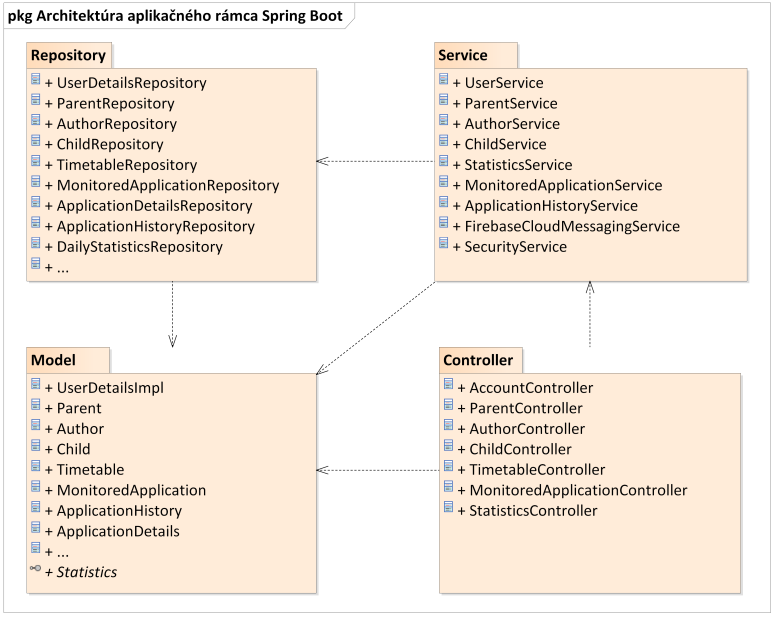
v aplikačnom rámci Spring. Nástroj zobrazuje požadovanú formu požiadavky, povinnosť jednotlivých atribútov a zobrazuje ukážkovú odpoveď zo servera. Po vložení autentizačného tokenu je taktiež možné jednotlivé požiadavky priamo vyskúšať vo webovom rozhraní voči reálnemu testovaciemu serveru.

4.4 Návrh a implementácia aplikačného servera

Po tom, čo sa vykonali prvotné návrhy architektúry servera, databázy a aplikačného programového rozhrania webových služieb, je možné realizovať návrh aplikácie pomocou aplikačného rámca Spring Boot, ktorý zaobstaráva biznis logiku celého servera. Táto podkapitola je venovaná návrhu a realizácii aplikačného servera pomocou aplikačného rámca Spring Boot.

4.4.1 Logická architektúra aplikácie

Aplikácia je rozdelená do viacerých balíčkov, ktoré obsahujú triedy majúce podobný účel. Aplikácia podľa obrázka 4.3 obsahuje 4 základné balíky. Radenie tried do balíkov zabezpečuje ľahšiu udržateľnosť a rozšíriteľnosť tak, aby zmeny neovplyvnili nesúvisiace časti systému. Pre jednoduchšiu lokalizáciu chýb sú triedy a balíky navrhnuté tak, aby neobsahovali cyklické závislosti.



Obrázok 4.3: Rozdelenie tried aplikačného rámca do balíčkov podľa funkcionality tried.

Balík `Model` obsahuje všetky entity aplikácie určené na uchovávanie dát. Triedy balíka `Repository` majú za úlohu zabezpečenie CRUD operácií nad da-

tabázou. Tieto balíky sa radia pod perzistentnú vrstvu aplikácie. Aplikačná vrstva aplikácie je zabezpečená triedami v balíku Service, ktoré zabezpečujú logiku aplikácie. Balík `Controller`, patriaci do prezentačnej vrstvy aplikácie, slúži na sprostredkovanie REST API. V súčasosti je logická architekrúra systému organizovaná striktne, t. j. ovládače (Controllers) nemôžu priamo komunikovať s úložiskom (Repository), a to hlavne kvôli použitiu autorizačných procesov v servisnej vrstve, ďalej riešených v oddiele 4.4.4. Oproti relaxovane vrstvenej architektúre môže dané riešenie v jednoduchých operáciach zvyšovať komplexnosť systému.

Aplikačný rámec Spring umožňuje automaticky spravovať a vytvárať závislosti pomocou kontajnera Spring IoC[36], ktorý využíva princíp obráteného riadenia[37] [13]. Pre využitie automatického spravovania komponentov je nutné priradiť danej triede anotáciu `@Component`. Tým sa automaticky vytvorené objekty triedy stávajú tzv. *bean* a sú dostupné k injektovaniu v ostatných komponentách. Všetky objekty *bean* je možné nakonfigurovať, a to buď pomocou konfiguračného súboru xml, alebo v konfiguračných triedach, majúcich anotáciu `@Configuration`. V serverovej aplikácii Educhild je preferovaná konfigurácia pomocou tried z dôvodu väčšej prehľadnosti a zníženia redundancie kódu.

V aplikačnom rámci Spring existujú špeciálne prípady komponentov. Komponent, ktorého trieda obsahuje anotáciu `@Repository`, slúži primárne pre prácu s databázou a je schopná transformovať špecifické výnimky vyhodené z databázovej vrstvy. Všetky triedy z balíka `Repository` preto budú zadefinované ako komponenty pomocou uvedenej anotácie. Pre pridanie tried obsahujúce biznis logiku aplikácie do aplikačného kontextu aplikačného rámca Spring slúži anotácia `@Service`, ktorou sú označené všetky triedy z balíka `Service`.

4.4.2 Perzistentná vrstva

Do perzistentnej vrstvy aplikácie patria všetky triedy z balíkov `Model` a `Repository` a slúžia primárne pre prácu s uchovávaním dát. Pre umožnenie ukladania dát z aplikácie do databázy bola využitá metóda objektovo-relačného mapovania, ktorá transformuje triedy a jej atribúty na databázové relácie, atribúty a relačné vzťahy. Metóda ORM je v súčasnosti vzhľadom na jednoduchší model aplikácie vhodná a významne redukuje tvorbu nadbytočného kódu. Napriek tomu prináša nevýhody, akou je napríklad náročnejšie ladenie a hľadanie chýb vygenerovaného SQL kódu.

Pre zavedenie ORM je použité rozhranie JPA[38], ktoré je špecifikáciou jazyka Java [13] a umožňuje namapovať triedu na reláciu pomocou anotácií. Pre implementáciu rozhrania bol použitý aplikačný rámec Hibernate

[36] Inversion of Control
[37] angl. Dependency inversion
[38] Jakarta Persistence

```
@Entity
data class Child(
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Column(name = "child_id")
  val id: Int = 0, //default value avoids attribute to be nullable
  @Column(unique = true, nullable = false)
  val uuid: UUID = UUID.randomUUID(),
  @ManyToOne
  @JoinColumn(name = "parent_id", nullable = false)
  val parent: Parent,
  @Column(length = CHILD_NAME_LENGTH, nullable = false)
  val name: String,
  @Column(name = "play_time_per_day", nullable = false)
  val playTimePerDay: Long,
  @Embedded //attributes of table child are also inside of embedded class
  val monitoredAppSettings: MonitoredAppSettings,
  ...
  @OneToMany(mappedBy = "child")
  val timetables: Set<Timetable> = emptySet(),
  @ManyToMany(mappedBy = "children")
  val quizzes: Set<Quiz> = emptySet(), //must be set to secure constraints
) : Updatable {
  init {
    require(playTimePerDay >= 0L || playTimePerDay == UNSPECIFIED) {
      "playtimePerDay must be positive or must equal $UNSPECIFIED
      if not specified"
    }
  }
  ...
}
```

Zdrojový kód 4.1: Ukážka časti zdrojového kódu dátovej triedy reprezentujúcej dieťa so zameraním sa na objektovo-relačné mapovanie entity.

ako implicitná súčasť aplikačného rámca Spring Boot a jeho závislosti `spring-boot-starter-data-jpa`. Pre prácu s JPA museli byť do projektu nainštalované doplnky pre prácu s jazykom Kotlin. Pri vytvátaní triedy určenej na uchovávanie dát je v Kotline obzvlášť výhodné použiť zvláštny typ triedy označovaný ako `data class`, ktorý zabezpečí okrem iného automatickú rovnosť dvoch inštancií (t. j. prepíše metódy `equals` a `hashCode`, ktoré sa využívajú pri komparácii objektov) s rovnakými atribútmi, implicitne vytvorí metódu `copy` určenú na klonovanie objektov alebo automaticky vygeneruje

```
CREATE TABLE child(
  child_id INTEGER GENERATED BY DEFAULT AS IDENTITY NOT NULL,
  date_of_birth date NOT NULL,
  auto_blocking_after_installation BOOLEAN NOT NULL,
  block_game_by_default BOOLEAN NOT NULL,
  block_social_by_default BOOLEAN NOT NULL,
  name VARCHAR(255) NOT NULL,
  play_time_per_day BIGINT NOT NULL,
  updated TIMESTAMP WITHOUT TIME ZONE NOT NULL,
  uuid UUID NOT NULL,
  parent_id INTEGER NOT NULL,
  CONSTRAINT "childPK" PRIMARY KEY (child_id)
);
ALTER TABLE child ADD CONSTRAINT UC_CHILDUUID_COL UNIQUE (uuid);
ALTER TABLE child ADD CONSTRAINT "FK7dag1cncltpyhoc2mbwka356h"
    FOREIGN KEY (parent_id) REFERENCES parent (parent_id);
```

Zdrojový kód 4.2: Ukážka automaticky vygenerovaného DDL kódu pre vytvorenie tabuľky dieťaťa v jazyku SQL

relevantný textový výstup pomocou metódy `toString`. Súčasťou dátovej triedy je taktiež jediný konštruktor, ktorý obsahuje všetky atribúty dátovej triedy, čo je v konflikte s podmienkami pre vytvorenie entity určenej k mapovaniu, ktorá musí obsahovať aspoň jeden verejný alebo chránený bezparametrický konštruktor. [44] Preto bol importovaný doplnok `kotlin-maven-noarg` od oficiálnych vydávateľov jazyka Kotlin, ktorý automaticky zabezpečí bezparametrický konštruktor pre triedy. Všetky triedy v jazyku Kotlin sú – oproti jazyku Java – implicitne finálne, čo znamená, že nemôžu byť predkom žiadnej triedy a pre umožnenie dedenia je nutné označiť triedy kľúčovým slovom `open`. Podmienkou ORM je, že entitná trieda nesmie byť finálna. To je zabezpečené doplnkom `kotlin-maven-allopen` od rovnakého vydávateľa, ktorý automaticky nastaví triedy ako nefinálne.

Všetky entity boli namapované podľa databázového modela zobrazeného v dodatku D a opísaného v podkapitole 4.2. Pri modelovaní atribútov bolo nutné dbať na správne označenie toho, či atribút môže nadobudnúť hodnotu null, pretože jazyk Kotlin – oproti jazyku Java – rozdeľuje dátové typy na nulové a nenulové. Pri mapovaní atribútov bolo nutné dbať taktiež na správne označenie unikátnosti atribútov (ktorých absencia viedla v testovacej fáze k abnormálnemu správaniu), správnu dĺžku reťazcov a rozsah číselných hodnôt, a správnu konverziu dátumu a času na SQL formát (nutnosť rozlišovať medzi typmi `DATE`, `TIME`, `TIMESTAMP`). Pri modelovaní vzťahov M:N je nutné dbať na dátové typy umiestnené pod kolekciami. V prvotnom návrhu bolo

pre kolekcie využité rozhranie typu `List`, ktoré nezabezpečilo v dekompovanej tabuľke unikátnosť prvkov, ktorá bola žiadaná. Pre zabezpečenie unikátnosti prvkov v relácii bol využitý typ `Set` reprezentujúci množinu.

Pre manipuláciu s dátami v databáze sa využíva vzor DAO[39], ktorý zavádza CRUD operácie nad databázou a poskytuje jednotné rozhranie pre prácu s viacerými databázovými zdrojmi. Pre zníženie písania nadbytočného kódu sa v aplikačnom rámci Spring využíva rámec Spring Data Repository ponúkajúce rozhranie `Repository`, ktoré automaticky vygeneruje SQL kód na základe mennej konvencie metód. Generickými parametrami rozhrania sú typ entity, nad ktorou sa budú vykonávať operácie a dátový typ primárneho kľúča. Pre vytvorenie SQL dotazu je nutné nazvať metódu podľa špecifickej operácie (napríklad `find..By`, `save`, `delete..By`). [45] Všetky kľúčové slová, ktoré je možné v názve použiť, je možné nájsť v dokumentácii aplikačného rámca Spring. Pre dotazovanie je možné taktiež využiť vlastný SQL dotaz pomocou jazyka SpEL alebo pomocou natívneho dotazu.

```
@Repository
interface MonitoredApplicationRepository
  : JpaRepository<MonitoredApplication, Int>
{
  fun findByChildAndPackageName(child: Child, packageName: String)
    : MonitoredApplication?
  fun deleteByChildAndPackageName(child: Child, packageName: String)
  fun findAllByChild(child: Child, pageable: Pageable)
    : Page<MonitoredApplication>
  fun deleteAllByChild(child: Child)
}
```

Zdrojový kód 4.3: Ukážka DTO pre dotazovanie nad entitou s monitorovanými aplikáciami dieťaťa. Väčšina CRUD operácii je už deklarovaná v rozhraní JpaRepository.

Potomkom rozhrania `Repository` je rozhranie `CrudRepository`, ktoré ponúka preddeklarovanú väčšinu CRUD operácií. Aj preto vo vlastnom rozhraní, v ktorom je nutné zadať generické parametre rozhrania, nie je nutné deklarovať žiadne ďalšie operácie. Pre zvýšenie čitateľnosti kódu boli preto mnoho rozhraní zlúčené do jedného súboru, čo jazyk Kotlin oproti jazyku Java umožňuje. Ďalšou výhodou oproti jazyku Java je zabudovaná kontrola nulovosti, vďaka čomu nie je nutné využívať na kontrolu nulovosti typ `Optional`, čo je štandardné riešenie v jazyku Java pri práci s rozhraním `Repository`.

Spring Data Repository ponúka taktiež možnosť vyhľadávať záznamy stránkovane. Ako posledný parameter dotazovacej metódy je možné pridať triedu

[39] Data Access Object

implementujúcu rozhranie `Pageable`, v ktorom je možné definovať žiadanú stránku, počet dotazov na stránku alebo spôsob triedenia výsledkov. Daná metóda následne vráti danú stránku so žiadanými vlastnosťami.

4.4.3 Aplikačná vrstva

Aplikačná vrstva, ktorá zaobstaráva biznis logiku aplikácie, je rozdelená do viacerých servisných tried podľa obrázka 4.3. Väčšina metód zaobstaráva bežné spracovanie dát doručených z REST API, ktoré následne uloží do databázy. Tieto metódy majú anotáciu `@Transactional`, ktorá zabezpečí transakčné spracovanie metódy, ktorá v prípade vyhodenia výnimky vráti databázu do predchádzajúceho stavu[40], respektíve zabezpečí potvrdenie zmien[41], čím sa zabezpečí konzistencia dát v databáze.

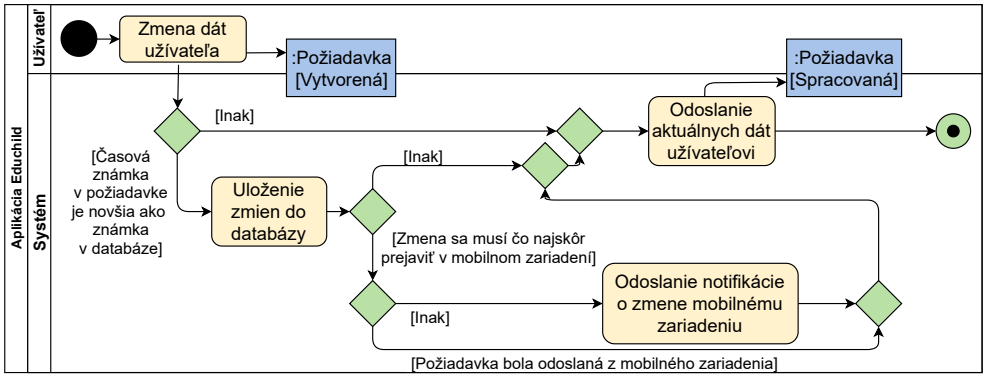
Počas vývoja aplikácie zriedkavo vznikal problém súvisiaci s prepisovaním aktuálnych dát neaktuálnymi dátami pri aktualizačných operáciách. To bolo spôsobené tým, že požiadavky z Android aplikácie sa odosielali na server oneskorene, zatiaľ čo rodič aktualizoval dáta na webe. Po príchode dát z Android zariadenia sa prepísali novšie zmeny, ktoré vykonával rodič na webe. Daný problém je ale zriedkavý a nepravdepodobný, pretože požiadavky z Android aplikácie sa odosielajú čo najskôr a rodič si je vedomý zmien, ktoré vykonal. Problém bol odstránený pridaním časovej známky modifikácie do každej požiadavky, ktorá bola schopná premazať aktuálne dáta. Daná známka je následne uchovaná v databáze.

Atypickou servisovou triedou je trieda zaobstarávajúca odosielanie notifikácií do Android zariadenia. Tieto notifikácie sa odosielajú pri entitách, ktoré sú potrebné aktualizovať sa čo najrýchlejčie pre správne fungovanie Android aplikácie (obzvlášť počas detského módu aplikácie, kde môže nastavenia zariadenia ovládať rodič vzdialene). Jedná sa o entity rodiča, dieťata, monitorovaných aplikácií a rozvrhu. Notifikovanie Android zariadenia sa udeje vždy, keď nová zmena nastala z iného zdroja (t. j. z webu). Pre odoslanie notifikácie musí mať rodič zaregistrovaný na serveri Firebase token pre push notifikácie, ktorý sa odosiela na server ihneď po prihlásení sa v zariadení. Odosielanie notifikácií je riešené mapou, ktorej kľúče obsahujú typ operácie, ktorú je potrebné na Android zariadení vykonať. Hodnoty kľúčov sú identifikátory potrebné pre rozpoznanie entity, ktorú je nutné aktualizovať.

4.4.4 Zabezpečenie aplikácie

Jednou zo základných nefunkčných požiadaviek každého systému je zabezpečenie aplikácie proti neoprávnenému prístupu. Jednou zo základných vecí, ktoré je potrebné vyriešiť v každej HTTP požiadavke, je autorizácia a autentizácia klienta. Pre autorizáciu užívateľa sa v aplikačnom rámci Spring

[40] angl. rollback
[41] angl. commit



Obrázok 4.4: Ukážka diagramu aktivít hladkého priebehu pri aktualizácii dát užívateľa.

využívajú role a oprávnenia. Každá rola môže obsahovať niekoľko oprávnení. V aplikácii sa pri každom invokovaní požiadavky vytvorí kontext[42], v ktorom sa uchovávajú všetky autentizačné role a oprávnenia, ktoré požiadavka nadobudne. Následne sa môže využívať kontext v jednotlivých triedach a metódach pre zaistenie prístupu.

V aplikácii je možné získať role a oprávnenia dvomi spôsobmi. Prvý spôsob je, že po autentizácii užívateľa sa do kontextu predajú všetky pridelené právomoci, ktoré má uložené vo svojom užívateľskom profile. V aplikácii tieto informácie uchováva trieda `UserDetailsImpl`, ktorá implementuje rozhranie aplikačného rámca Spring Security `UserDetails`. Následne sa k údajom pristupuje pomocou servisnej triedy, ktorá implementuje rozhranie `UserDetailsService` z rovnakého rámca. Výhodou použitia rozhraní aplikačného rámca je, že čokoľvek, čo vráti servisná trieda `UserDetailsService`, môže byť získané z bezpečnostného kontextu [13]. Užívateľ môže momentálne disponovať rolami

- `USER`, ktorá umožňuje vykonávať operácie súvisiace so správou účtu;
- `PARENT`, prideľujúca všetky oprávnenia prístupne rodičovi, ako nastavovanie oprávnení dieťaťu;
- `AUTHOR`, slúžiaca pre operácie súvisiace s tvorbou kvízov;
- `CHILD`, pomocou ktorej môžu oprávnenia pre dieťa napríklad ukladať záznamy o navštívených aplikáciách.

Faktom je, že v súčasnosti má každý serverový užívateľ pridelenú každú z vyššie uvedených rolí a všetky obmedzenia prístupu pre dieťa sú riešené v logike klientských aplikácií. Tento návrh ale explicitne rozdeľuje aktérov v systéme a umožňuje budúce rozšírenie systému, kde napríklad každé dieťa bude môcť

[42]kontext je rozhranie typu `SecurityContext`

vlastniť samostatné prihlasovacie údaje, čím mu bude napríklad umožnené exekuovať kvízy taktiež na webe.

Ďalšie oprávnenia je možné získať tým, že sa do HTTP požiadavky vloží hlavička `X-API-KEY`, ktorej hodnota indikuje typ klienta. Na základe typu klienta môže bezpečnostný kontext nadobúdať rolu `ANDROID_DEVICE` alebo `WEB_CLIENT`. Tieto role umožňujú využívať nezaregistrovanému užívateľovi časť aplikácie, a to napríklad pre sťahovanie kvízov alebo na zisťovanie informácií o aplikáciách. Pôvodne miesto týchto dvoch rolí bola v systéme len jedna rola s názvom `ANY`, ktorá mala podobný účel. Webový a mobilný klient majú však rôzne potreby a pridelením jedinej role získavali viac oprávnení, než bolo potrebné, čím bolo zhoršené zabezpečenie systému. Rozdelenie rolí podľa typu klienta taktiež pomohlo zaistenie typu klienta v aplikačnej logike systému, na základe čoho je možné zistiť, či je nutné odosielať notifikácie kokrétnemu zariadeniu.

Zoznam rolí a ich oprávnení je možné nájsť v tabuľke 4.1. Názvy rolí a oprávnení boli zanechané v anglickom jazyku tak, ako je to v zdrojovom kóde aplikácie. Oprávnenia týkajúce sa kvízovej časti aplikácie boli navrhované Petrom Šimom.

Kontrola právomoci sa v aplikácii kontroluje pomocou anotácie metód `@PreAuthorize`, v ktorej sa pomocou SpEL definujú predikáty. V špecifickom prípade sa môže kontrola vykonávať v tele metódy pomocou rozhrania `SecurityContext`.

Autentizácia užívateľa sa vykonáva pomocou Firebase autentizačného tokenu a API kľúča. V aplikačnom rámci Spring Security prebieha overovanie pridaním filtra, ktorý je schopný v prípade úspešného overenia pridať autentizačné údaje do bezpečnostného kontextu. Filter sa vytvorí implementáciou rozhrania `OncePerRequestFilter`. Vo filtri kontrolujúci Firebase autentizačný token je token overovaný pomocou knižnice `firebase-admin`. Následne je token uložený v potomkovi triedy `AbstractAuthenticationToken`, v ktorej je uložená identita užívateľa. Kontrola autenticity užívateľa pri požadovaní entity sa verifikuje v anotácii metód `PostAuthorize`, v ktorej sa pomocou SpEL skontroluje, či vrátený objekt má rovnaké identifikácie, ako autentizačné údaje (t. j. kontroluje sa hodnota unikátneho identifikátora užívateľa oproti identifikácii z tokenu).

Počas testovania aplikácie bol zistený problém, kedy pri volaní istých metód nedochádzalo k vykonávaniu autentizácie a autorizácie, definovaných pomocou anotácií. Dôvodom bolo, že metóda bola zavolaná priamo z inej metódy, ktorá sa nachádzala v rovnakej triede. Pri priamom volaní metódy sa nezavolá tzv. proxy trieda, v ktorej je implementované správanie definované v anotáciách. Problém bol vyriešený kvalitnejším návrhom, v ktorom si metóda vyžiada všetky potrebné údaje v parametroch a nepotrebuje si ich vyhľadávať sama (pôvodný návrh porušoval princíp jednej zodpovednosti[43]).

[43]angl. Single-responsibility principle

V prípadoch, kedy bolo napriek tomu nutné zavolať metódu z rovnakej triedy, bola metóda zavolaná pomocou aplikačného kontextu, cez ktorý bolo možné pristúpiť k žiadanému objektu *bean*.

	USER	PARENT	AUTHOR	CHILD	ANDROID_DEVICE	WEB_CLIENT
account:write	+					
parent:read		+				
parent:write		+				
author:read			+			
author:write			+			
child:read		+		+		
child:write		+				
timetable:read		+		+		
timetable:write		+				
monitored_application:read		+		+		
monitored_application:write		+				
application_details:read		+			+	+
application_details:write					+	
child_statistics:read		+		+		
child_statistics:write						
application_history:read		+		+		
application_history:write					+	
quizzes:read		+	+	+		+
quizzes:write		+	+			
quiz_statistics:read		+		+		
quiz_statistics:write					+	
images:read		+		+	+	+
images:write		+			+	
push_notification:send						+

Tabuľka 4.1: Role a oprávnenia užívateľov na serveri. Prvý riadok znázorňuje všetky role na serveri, v prvom stĺpci sú vypísané všetky oprávnenia.

4.4.5 Prezentačná vrstva

Prezentačná vrstva slúži pre implementáciu REST API navrhnutého v podkapitole 4.3. Každá trieda implementujúca REST API musí mať anotáciu `RestController`. Zdroje a metódy rozhrania sú definované pomocou anotácií

tried a metód. Pre transformáciu tela vo formáte JSON bol využitý modul `jackson-module-kotlin`, ktorý umožňuje serializovať a deserializovať dátové triedy jazyka Kotlin do formátu JSON. Pre konverzu sú použité preddefinované konverzie, ale je možné si vytvoriť taktiež vlastý formát. Ten bol využitý pri serializácii a deserializácii dátumu narodenia dieťaťa, v ktorom je podľa analýzy považovaný iba rok a mesiac narodenia.

Všetky dátové triedy reprezentujúce požiadavku alebo odpoveď[44], sú nezávislé od tried z perzistentnej vrstvy. Pôvodne bola pre požiadavku aj odpoveď využitá rovnaká trieda, ale tento návrh sa časom ukázal byť ako nedostatočný pre rôzne potreby v tele požiadavky a odpovede. Každá trieda reprezentujúca odpoveď dedí od triedy `RepresentationModel`, vďaka čomu je možné do tela správy pridať jednoducho odkazy na ďalšie zdroje, čím sa uplatní prícip HATEOAS, nutný pre RESTful API.

Pôvodne sa všetky výnimky z nižších vrstiev zachovávali v ovládačoch, čo spôsobovalo neprehľadnosť kódu. V novom návrhu sa využíva mechanizmus zachytávania výnimiek, pomocou ktorého sa výnimky odchytia globálne a následne sa transformujú na odpovede s reprezentujúcimi chybovými kódmi. Vďaka tomu je možné riešiť v ovládačoch iba hladký priechod volania.

4.5 DevOps

Devops (zložený z anglických výrazov Development a Operations) je forma vývoja, ktorá si kladie za cieľ rýchly a flexibilný vývoj, a zabezpečenie obchodných procesov. Účinne integruje vývoj, dodávku a operácie, čím zabezpečuje plynulé spojenie týchto tradične oddelených činností. [46] Táto kapitola je venovaná integrácii vývojového prístupu Devops do projektu Educhild.

Automatizácia zostavenia aplikácie je jednou z najpodstatnejších činností DevOps. Zabezpečuje kompiláciu kódu, spravovanie závislostí, generovanie závislostí, spúšťanie testov a nasadzovanie aplikácie na rôzne prostredia. Aplikačný rámec Spring Boot používa nástroj Maven, ako bolo uvedené v podkapitole 4.1. Pre konfiguráciu aplikácie medzi rôznymi prostrediami sa využívajú profily aplikačného rámca Spring. V profiloch sú nastevené autentizačné údaje do databázy a požadovaný JDBC ovládač pre využívanú databázu, pripojenie k platforme Firebase a Sentry, prefix cesty adresy URL, a podobne.

Súčasťou DevOps je taktiež nutnosť verzovania kódu, databázy a dát. Pre verzovanie kódu je použitý nástroj Git, distribuovaný systém pre správu verzií. Výhodou nástroja Git oproti konkurenčnému nástruju SVN je jeho distribuovanosť a možnosť jednoduchého vytvárania a zlučovania vetiev. Repozitár Git je umiestnený na školskej platforme GitLab. V súčasnosti sú oproti počiatočnému spôsobu vývoja v repozitári umiestnené dve tzv. chránené vetvy, ktoré nie je možné vymazať a každá zmena musí podliehať žiadosti o zlúčenie a následnej revízii kódu privilegovanými osobami. Vetva `master` reprezentuje

[44] takéto objekty sú tiež nazývané ako objekty pre prenos dát, angl. Data transfer objects

budúce produkčné prostredie, vetva `test` reprezentuje testovacie prostredie. Verzovaniu databázy a dát sa venuje bakalárska práca Petra Šímu.

Po pridaní novej revízie do vzdialeného repozitára sa automaticky spúšťajú úlohy v rámci nástroja Gitlab CI/CD. V rámci tzv. nepretržitej integrácie[45] sa spúšťajú úlohy pre otestovanie úspešného zostavenia aplikácie a exekúciu všetkých jednotkových testov. Po úspešnom prevedení týchto krokov sú spustené úlohy v rámci nepretržitého doručovania[46], ktoré majú za úlohu vytvorenie obrazu aplikačného rámca, ktorý sa uloží do tzv. GitLab registra, odkiaľ je možné obraz stiahnnuť. Následné nasadenie na server prebieha pomocou SSH pripojenia, ktorého údaje na pripojenie sú uložené v chránených premenných prostredia.

Po pripojení sa invokuje stiahnutie obrazu na server pomocou tzv. Deploy tokenu vygenerovaného z platformy GitLab a následné spustenie kontajnera pomocou nástroja Docker Compose. Tento nástroj však znovu vytvára iba kontajnery, ktorých obraz nadobudol zmenu. Preto je možné nasadiť aplikačný server samostatne bez manipulácie webového kontajnera a naopak, čo je v súlade s mikroservisnou architektúrou DevOps. V počiatočných fázach projektu neexistovalo automatické nasadzovanie na server. To zapríčiňovalo oneskorené nasadzovanie, možnosť vnesenia chýb nedodržaním správnej postupnosti krokov a viedlo k dezorganizácii aktuálne nasadenej verzie na serveri.

Pri spúšťaní servera pomocou nástroja docker-compose (viď. dodatok C) je ako prvá spustená databáza, po ktorej nasleduje spustenie aplikácie Spring Boot. Nástroj invokuje spúšťanie aplikácie ihneď po tom, čo je spustená databáza. K databáze sa ale je možné pripojiť až niekoľko sekúnd po jej spustení. Medzitým ale aplikácia zahlási fatálnu chybu, pretože sa pri spúšťaní nemohla pripojiť k databáze. Tento problém bol vyriešený vytvorením skriptu, ktorý spustí aplikáciu až po tom, čo je prístupný port na pripojenie sa k databáze. Sktipt bol navrhnutý Petrom Šímom.

Dôležitou súčasťou DevOps je taktiež monitorovanie systému. Pre monitorovanie aplikačného servera bol vybraný nástroj Sentry z dôvodu jeho dominantného postavania na trhu [47] a preto, že väčšina nástrojov na zaznamenávanie chýb obsahuje veľmi podobné funkcionality. Vďaka tomuto nástroju bolo počas vývoja odhalených niekoľko chýb, ktoré by ostali nepovšimnuté z dôvodu neprehľadného a dočasného konzolového výpisu zo servera. Nástroj umožňuje separáciu jednotlivých prostredí, poskytuje integráciu pre aplikačný rámec Spring Boot a poskytuje štatistiky o čase a počte jednotlivých chýb vrátane ich dohľadania. Okrem toho je možné využiť priraďovanie chyby konkrétnemu vývojárovi, ako zasielanie informácii o chybách do e-mailu, čo je v súlade s metodikou DevOps. Okrem toho je taktiež umožnené vzdialené ladenie[47] kódu na testovacom serveri po pripojení sa pomocou SSH pripojenia

[45] angl. Continuous Integration
[46] angl. Continuous Delivery
[47] angl. debugging

a presmerovania portu[48].

Pre realizáciu DevOps je taktiež dôležizá tímová kolaborácia, ktorá bola zavedená ihneď po založení projektu. Pre správu životného cyklu aplikácie je využitý nástroj Redmine, ktorý slúži na prideľovanie a správu úloh a na utilizáciu. Aplikácia Trello bola využitá na prideľovanie čiastkových úloh a na kooperáciu pri úlohách určených pre viacero členov tímu. Ako komunikačný nástroj slúži aplikácia Slack a sú organizované synchronizačné schôdze minimálne raz za týždeň. Pre zdieľanie vedomostí a dokumentovanie bola na platforme Gitlab na serverovom projekte založená stránka Wiki, hoci kvôli jej neskorému zavedeniu v súčasnosti neobsahuje informácie potrebné pre prípadného nového člena projektu.

4.6 Komunikácia Android aplikácie so serverom

Táto časť je venovaná návrhu a realizácii komunikácie Android aplikácie so serverom a následnej synchronizácie dát v aplikácii. Android aplikácia je konzumentom REST API, ktorého návrh je popísaný v podkapitole 4.3.

4.6.1 HTTP klient

Na realizáciu bol využitý HTTP klient Retrofit s knižnicou Moshi pre konverziu formátu JSON. Retrofit je oproti konkurenčnému nástroju Volley jednoduchší pre deklaráciu API a ponúka vývojárovi väčšiu nezávislosť pre prácu s rozhraním. Pre konzumáciu REST API je nutné deklarovať štruktúru rozhrania, ktorého metódy obsahujú anotácie s URI zdroja a typom HTTP metódy. Každá metóda je deklarovaná s kľúčovým slovom jazyka Kotlin `suspend`, vďaka čomu musí byť metóda invokovaná z Kotlin korutiny, čím je garantované, že volanie API nebude exekuované z hlavného vlákna určeného najmä pre prácu s GUI. Každej požiadavke a odpovedi reprezentovaných vo forme Kotlin dátovej triedy, je automaticky vygenerovaný adaptér pre konverziu z a do JSON formátu. Všetky základné dátové typy sú zkonvertované automaticky, ale je taktiež možné napísať si vlastný konvertor pre dátový typ, ako napríklad pri potrebe uloženia roka a mesiaca narodenia pre dieťa.

Do každej HTTP požiadavky je nutné vložiť hlavičky slúžiace na autentizáciu užívateľa. Pre zautomatizovanie tohto kroku bol využitý interceptor, ktorý automaticky vloží potrebné hlavičky. Autentizačný token je získavaný pomocou knižnice `FirebaseAuth`, ktorá ho poskytne na základe prihlasovacích údajov užívateľa, ktoré si uchováva po dobu prihlásenia užívateľa.

Šifrované spojenie aplikácie zabezpešuje protokol HTTPS. Zariadenie si preveruje dôveryhodnosť certifikátu domény pomocou sytémových koreňových certifikačných autorít. Na začiatku vývoja, kedy serveru nebola pridelená doména, bolo nutné sa pripájať priamo pomocou IP adresy. Z toho dôvodu

[48] angl. port forwarding

bolo nutné používať vlastnoručne podpísaný certifikát vydaný pre IP adresu pomocou alternatívneho názvu subjektu[49]. Pre zaručenie dôveryhodnosti certifikátu v aplikácii musel byť certifikát ručne pridaný do tzv. SSL kontextu, vytvoreného pri vytváraní HTTP klienta. Po zavedení serverovej domény bol tento problém vyriešený a certifikát je verifikovaný certifikačnými autoritami.

4.6.2 Typy požiadaviek na server

Po tom, čo je v aplikácii nainštalovaný HTTP klient, je možné invokovať samotné požiadavky. V aplikácii je možné požiadavky kategorizovať do viacerých typov podľa času odosielania požiadavky a podľa toho, či je zaručené prijatie úspešnej odpovede (resp. opakovanie odosielania až do prvého úspešného prijatia odpovede).

Okamžité negarantované požiadavky

Prvý typ požiadaviek sú požiadavky, ktorých odpoveď je nutné získať okamžite. Tieto požiadavky sú žiadané pri registrácii, kde nie je možné vynechať istý registračný krok s pokračovaním v ďalšom kroku. Pri neúspešnom prevolaní metódy je vyhodená výnimka a užívateľ je informovaný o nemožnosti pokračovať v registrácii.

Medzi požiadavky, ktoré je nutné spracovať ihneď, patria taktiež požiadavky vyvolávané pre aktualizáciu obrazovky, ktorú si aktívne prehliada užívateľ (t. j. požiadavky volané metódou GET). Kotlin korutina – ktorá sa v aplikácii Educhild využíva pre všetkú asynchrónnu prácu –, sa spustí ihneď po vytvorení komponentu `ViewModel`. Po úspešnom prevolaní metódy sa výsledok uloží do internej Android databázy Room. Vysledky z databázy spracuje komponent `ViewModel` pomocou triedy `LiveData`, ktorá implementuje návrhový vzor pozorovateľ[50]. Pri neúspešnom prevolaní sa využijú dáta uložené v internej databáze, pretože Android aplikácia musí byť plne funkčná aj počas nedostupnosti pripojenia do siete. V pôvodnej implementácii aplikácie nebola uvažovaná možnosť aktualizácie dát po tom, čo bola načítaná obrazovka pre užívateľa a všetky metódy slúžiace pre čítanie dát z databázy pre zobrazenie užívateľovi museli byť prerobené tak, aby realizovali návrhový vzor pozorovateľa (t. j. pomocou triedy `LiveData`). Tieto požiadavky, exekuované v korutinách spustených v komponente `ViewModel`, sú závislé od životného cyklu tohto komponentu a v prípade jeho deštrukcii sa automaticky ukončia aj závislé korutiny spracovávajúce dané požiadavky.

Okamžité garantované požiadavky

Medzi ďalšie typy požiadaviek patria požiadavky, ktorých úlohou je aktualizovať dáta na serveri, ktorých zmenu je žiadané odoslať čo najskôr (napríklad

[49] angl. Subject Alternative Name, SAN
[50] angl. Observer

požiadavky typu zmena nastavení dieťaťa). Úspešné vykonanie týchto požiadaviek musí byť garantované a v prípade neúspešného procesu musí byť exekúcia zopakovaná neskôr. Podľa kapitoly 2 je najvhodnejším riešením využiť knižnicu WorkManager. Keďže sa aktualizačné akcie iniciujú za aktívneho užívania aplikácie užívateľom, spustenie práce sa vykoná väčšinou ihneď.

Každý zdroj URI má vlastnú triedu, dediacu z triedy `CoroutineWorker`, v rámci ktorej je definovaná invokácia HTTP požiadavky a následná aktualizácia údajov v databáze podľa odpovede. V prípade neúspechu sa daná úloha opäť vráti do fronty úloh, z ktorej bude znova čo najskôr exekuovaná. Ako bolo zmienené v oddiele 4.4.3, každá aktualizačná požiadavka musí obsahovať časovú známku, kedy bola vytvorená. Táto známka sa vytvára pri definícii novej úlohy. Každá práca je vytvorená ako tzv. unikátna práca, čo zabezpečuje, že každá požiadavka rovnakého názvu môže byť umiestnená vo fronte prác maximálne jedenkrát súčasne. Pri opätovnej aktualizácii údajov v Android aplikácii sa stará práca vo fronte nahradí novou, čím sa prepíše časová známka.

Odložené garantované požiadavky

Medzi posledný typ požiadaviek patria požiadavky, ktoré je možné odložiť bez možnosti vzniku aktualizačných anomálií, pri krorých by si užívateľ mohol prepísať dáta (t. j. dáta, ktoré sa po odoslaní na server nikdy neaktualizujú). Medzi takéto požiadavky patria záznamy o navštívených aplikáciách, slúžiace pre zobrazovanie štatistík, alebo požiadavky odosielajúce informácie o aplikáciách. Tieto požiadavky je žiaduce agregovať a odosielať v dávkach.

Požiadavky sa ukladajú do internej Android databázy Room vo forme fronty, kde sa telá správ uložia priamo vo formáte JSON. Tieto požiadavky sa následne odosielajú v rámci inštancie WorkManager, ktorá sa v súčasnosti spúšťa periodicky každé 4 hodiny, a to v prípade, že užívateľ má dostatočne silnú batériu na zariadení a zároveň má sieťové pripojenie, ktoré nie je sprostredkované pomocou mobilnej siete. Periodicita spúšťania bola zvolená heuristicky a po dostatočnom množstve spracovania dát a informácií o vyťaženosti servera na produkčnom prostredí bude možné určiť vhodný interval presnejšie.

Požiadavky sa taktiež synchronizujú pri prechode z detského módu aplikácie do rodičovského a administratívneho módu a opačne, pretože pri prechode je očakávané, že sa pozastaví vytváranie istého typu požiadaviek, a teda je žiaduce tieto požiadavky odoslať (napríklad po prechode z detského módu sa nebudú pridávať záznamy o aplikačnej histórii). Užívateľ je taktiež v tomto prípade aktívny v aplikácii, čím urýchli spúšťanie práce. V počiatočných fázach sa do tohto typu zaraďovali aj požiadavky, ktoré mohli spôsobovať problémy s neaktuálnosťou dát na webe, a ktoré mohli byť prepísané na webe, ako napríklad požiadavka na aktualizáciu dát dieťaťa. Pre tento typ požiadaviek je vnodné odoslanie čo naskôr po zmene.

4.6.3 Synchronizácia dát

Okrem odosielania dát na server je taktiež nutné zaručiť pravidelné prijímanie dát, čím sa zaručí ich aktuálnosť v Android aplikácií. Okrem sťahovania konkrétnych dát pri prehliadaní aplikácie je potrebné pre správne fungovanie aplikácie zaručiť aktuálnosť všetkých údajov. Pre synchronizáciu sa používa rovnaká inštancia práce, ktorá odosiela odložené požiadavky, z dôvodu šetrenia zdrojov zariadenia.

Exekúcia práce zabezpečí aktualizáciu všetkých entít okrem tých, ktoré sú zaradené vo fronte požiadaviek na odoslanie – ich stiahnutím by sa prepísali zmeny, ktoré vykonal užívateľ v aplikácii. Aktualizácia týchto údajov prebehne podľa tela odpovede metódy POST. Okrem aktualizácii existujúcich údajov sa na server nahrajú novo nainštalované aplikácie zariadenia podľa údajov z internej databáze. Jediný rozdiel v aktualizačnom procese je pri aktualizácii monitorovaných aplikácií, kde sa ignoruje status o tom, či je nainštalovaná aplikácia, respektíve v prípade nekonzistencie sa vytvorí aktualizačná požiadavka, keďže tento údaj je vždy aktuálny jedine v Android aplikácii.

Okrem pravidelných aktualizácii je potrebné zaistiť okamžitú aktualizáciu údajov kritických pre chod aplikácie. Takéto údaje sú všetky údaje, ktoré sú potrené pre prácu v detskom móde, v ktorom je žiadané, aby rodič mohol zmeniť nastavenia dieťaťa cez webové rozhranie s tým, že zmeny sa musia prejaviť čo najskôr (jedná sa o entity dieťaťa, rozvrhu hrania a monitorovaných aplikácií). Okrem týchto údajov je potrebné odhlásenie užívateľa z aplikácie v prípade zmazania účtu cez web. Okamžitá aktualizácia týchto údajov je zabezpečená pomocou notifikácií zo servera pomocou služby Firebase Cloud Messaging. Zo servera sa odošle textová správa o tom, ktoré údaje je potrebné notifikovať. Reakcia na správy je v aplikácii zabezpečená v službe (komponent je popísaný v podkapitole 2.3) s potomkom `FirebaseMessagingService`. Pri implemetácii notifikácii v aplikácii sa ukázala nutnosť prerobiť proces synchronizácie, aby bolo taktiež možné aktualizovať len jednotlivé entity podľa potreby, nakoľko boli synchronizačné procesy vysoko previazané.

4.7 Zhrnutie

V tejto kapitole bol popísaný návrh a realizácia servera a komunikácia Android aplikácie so serverom. Bola ukázaná základná architektúra servera a jej jednotlivé komponenty až po úplný popis serverovej aplikácie s aplikačným rámcom Spring Boot. Okrem toho bol v tejto kapitole popísaný spôsob vývoja a prevádzky projektu v časti DevOps. V kapitole boli popísané chyby z prvotných iterácií návrhu a implementácie projektu až po popis aktuálneho riešenia. Vyhodnoteniu a budúcim vylepšeniam súčasných nedostatkov je venovaná kapitola 6.

Kapitola 5

Testovanie

Táto kapitola je venovaná testovaniu serverovej časti aplikácie Educhild a testovaniu časti Android aplikácie súvisiacej s komunikáciou a synchronizáciou so serverom. Testovanie servera prebiehalo so spoluprácou Petra Šímu, ktorého bakalárska práca sa venuje vzdelávacej časti aplikácie. Dôkladnému otestovaniu rodičovského módu Android aplikácie sa venuje bakalárska práca Daniela Matouška.

5.1 Testovanie servera

Testovanie servera zahŕňa otestovanie všetkých vrstiev aplikácie vrátane kontroly zabezpečenia aplikácie. Pre testovanie servera sa používa aplikačný rámec Spring Boot Starter Test, pomocou ktorého je možné zabezpečiť potrebné závislosti k testovaniu a ich automatickú konfiguráciu. Aplikačný rámec taktiež v sebe zahŕňa testovacie knižnice ako JUnit pre jednotkové testovanie, Mockito pre vytvorenie falošných simulovaných objektov alebo JsonPath pre testovanie obsahu správ vo formáte JSON.

Pre testovanie sa používa samostatný profil, v rámci ktorého je možné nakonfigurovať aplikáciu pre účely testovania. V profile sa najmä konfiguruje databázový systém H2, ktorý uchováva dáta počas behu programu v pamäti RAM. Vďaka zvláštnemu profilu pre testovanie je taktiež možné zamedziť vytváraniu komponentov aplikácie, ktoré nie je vhodné spúšťať v testovacom prostredí, ako je napríklad knižnica FirebaseAuth slúžiaca na autentizáciu užívateľov. Túto knižnicu v každom teste simuluje falošný objekt Mock. Triedy knižnice sú ale finálne, čo implicitne znemožňuje nástroju Mockito vytvoriť falošný objekt, a preto bol nainštalovaný doplnok Mockito Inline umožňujúci simulovať finálne triedy.

Pri vytváraní objektu Mock pre knižnicu zabezpečujúcu autentizáciu pomocou platformy Firebase bol zistený nedostatok návrhu systému. Knižnicu je možné použiť priamo v programe zavolaním metódy `getInstance`. V prog-

rame sa teda nachádzala silná väzba na knižnicu, čo znemožňovalo jednoduché nahradenie inou závislosťou. Knižnica bola nakoniec do programu vložená ako tzv. *bean* a je ju možné spravovať pomocou kontajnera Spring IoC (viac v oddiele 4.4.1).

5.1.1 Jednotkové testy

Jednotkové testy slúžia na zaistenie predpokladaného správania istej časti aplikácie a majú za úlohu otestovať istú metódu alebo triedu. Pre testovanie bola použitá typická knižnica pre jazyk Java, JUnit, ktorá je súčasťou testovacieho aplikačného rámca. V aplikácii sa jednotkové testy využívajú na otestovanie biznis logiky aplikácie v balíku `Service` a pre komunikáciu s databázou v balíku `Repository`.

Keďže komunikácia s databázou je zabezpečená pomocou aplikačného rámca Spring Data Repository, je predpokladané, že automaticky generované a zdedené metódy sú vytvorené správne a ich funkčnosť je otestovaná vývojármi aplikačného rámca. Preto boli otestované len metódy, v ktorých je manuálne zadaný JPA alebo natívny SQL dotaz.

V servisných triedach je požadované otestovať čo najväčšiu časť zdrojového kódu, keďže v nich je umiestnená celá logika aplikácie. Keďže sa jedná o tzv. white box testy (t. j. testy vytvorené na základe znalostí implementácie), boli testy zamerané na miesta, kde sa rozhoduje o vykonanej akcii (typicky sa jedná o príkaz if) a následne sa overujú všetky možnosti priechodu kódu. Typicky bol v prípadoch užitia testovaný hladký priechod, následne boli validované hodnoty parametrov a prípadné správne vyhodenie výnimky, vrátenie správnych dát z perzistentnej vrstvy a prípadne odoslanie notifikácie na Android zariadenie. Pre komunikáciu s perzistentnou vrstvou aplikácie, na ktorej je aplikačná vrstva závislá, alebo iných komponent z aplikačnej vrstvy mimo aktuálne testovaného komponentu, bolo nutné vytvoriť objekty Mock, aby výsledky testovaného komponentu neboli ovplyvňované inými komponentami.

Kód podľa nástroja zabudovaného vo vývojovom prostredí IntelliJ IDEA pokrýva 100% tried, 81% metód a 74% riadkov kódu. Oproti pokrytiu kódu bol však pri testovaní kladený dôraz najmä na otestovanie kritických úsekov aplikácie (ako napríklad vetvenie kódu pomocou príkazu if). Posledné testovanie servera už nezahŕňa testovanie štatistík, pretože – ako bude uvedené v kapitole 6 – je táto časť aplikácie nepotrebná a je určená na odstránenie.

5.1.2 Integračné testy

Integračné testy slúžia na otestovanie aplikácie ako celku, kde sú prevolané všetky vrstvy aplikácie. Testy sú exekuované volaním REST API až po uloženie do databáze a vrátenie odpovede. Pre prevolanie požiadavky a následnú kontrolu odpovede je využitý nástroj aplikačného rámca MockMvc.

Testy boli zamerané najmä na správnosť dát vrátených v odpovedi a prípadne kontrolu databáze po prevolaní požiadavky. V rámci odpovede musel byť vrátený správny stavový kód a správny formát tela odpovede. V integračných testoch je taktiež otestovaná správna funkčnosť autentizačného a autorizačného procesu na základe autentizačného tokenu vloženého do požiadavky.

Pri integračnom testovaní bol zistený problém s nekonzistentnými výsledkami testov na lokálnom prostredí vo vývojovom prostedí IntelliJ IDEA a následnom testovaní v rámci nepretržitej integrácie na platforme Gitlab CI. Pri automatickom testovaní boli totiž testy spúšťané v rámci jednej inštancie dát, zatiaľ čo v lokálnom prostredí boli dáta vždy obnovené do pôvodného stavu po exekúcii každého testu. Táto nekonzistencia bola spôsobená tým, že v rámci testovania boli vytvorené preddefinované objekty, ktoré boli využívané vo viacerých testoch, čím sa znížilo množstvo redundantného kódu. V niektorých testoch teda bolo možné pracovať s už upravenými dátami, kde následne nemusel byť správne vyhodnotený predpoklad[51].

Integračné testy pokrývajú 100% tried, 78% metód a 85% riadkov kódu. Tento výsledok však skresľujú metódy potrebné pre kooperáciu so vzdelávacou časťou aplikácie, ktorej testy sú zahrnuté v integračných testoch Petra Šímu.

5.2 Testovanie Android aplikácie

V rámci Android aplikácie Educhild bolo potrebné otestovať triedy, ktoré komunikujú so serverom pomocou API a triedy, ktoré zabezpečujú synchronizačný proces pri prenose dát so serverom. Pre jednotkové testovanie tried bola využitá štandardná knižnica JUnit.

Keďže všetky funkcie zabezpečujúce logiku aplikácie musia byť exekuované v korutinách, musí byť telo každej testovacej metódy taktiež spustené v korutine. To je zabezpečené zabudovanou funkciou `runBlocking`, ktorá spustí novú korutinu s telom testovacej funkcie a zároveň zablokuje vlákno. Pre vytváranie simulovaných objektov Mock bola využitá knižnica MockK, ktorá zabezpečuje simulovanie funkcií, ktoré môžu byť spúšťané jedine v korutinách (t. j. sú označené kľúčovým slovom `suspend`).

Pri testovaní volania serverovéo API bolo potrebné preskúšať všetky možnosti, ktoré mohli nastať pri volaní a overiť predpokladané správanie triedy. Jedná sa o tzv. black-box testovanie, pri ktorom v rámci testovania Android aplikácie nie je známa vnútorná implementácia serverového API. Pre preverenie všetkých možností boli využívané rozhodovacie tabuľky (viď. tabuľku 5.1). Následne bol kladený dôraz na otestovanie volania tried zodpovedných za správne spracovanie a uloženie dát zo servera do zariadenia. V rámci testovaných tried bolo pokrytých 88% metód a 84% riadkov kódu.

[51] angl. assert

Existuje rodičovský účet?	A	A	A	A	N	N	N	N	Vstupy
Existuje autorský účet?	A	N	N	A	A	A	N	N	
Je možné odstrániť autorský účet?	N	A	N	A	A	N	A	N	
Odstránený rodičovský účet?	A	A	A	A	N	N	N	N	Výstupy
Odstránený autorský účet?	N	N	N	A	A	N	A	N	
Odstránený Firebase účet?	N	A	A	A	A	N	A	N	

Tabuľka 5.1: Ukážka rozhodovacej tabuľky pri testovaní odstránenia účtu v Android aplikácii. Pre všetky kombinácie vstupov sú zobrazené očakávané výstupy, ktoré je nutné otestovať. Červenou farbou sú zobrazené kombinácie vstupov, ktoré v Android aplikácii nikdy nemôžu nastať, a teda ich nie je potrebné otestovať.

Kapitola 6

Vyhodnotenie

Táto kapitola je venovaná vyhodnoteniu súčasného stavu a navrhuje riešenia pre budúce úpravy súčasného stavu.

6.1 Súčasný stav

V súčastnosti je serverová časť aplikácie Educhild plne funkčná a vyhovuje všetkým funkčným požiadavkám projektu. Aplikácia umožňuje v rámci administratívnej časti registráciu a odstránenie účtu, správu rodiča, autora a dieťaťa. Všetky koncové body sú zabezpečené a vyžadujú autentizáciu.

Aplikácia Educhild je momentálne dostupná v testovacej prevádzke na adrese `https://dev.edu4child.com` po zadaní autentizačných údajov. Na rovnakej doméne pod cestou `/api` je dostupné serverové REST API. Dokumentácia REST API je prístupná pod cestou `/api/swagger-ui.html`. Nasadzovanie servera sa v súčasnosti vykonáva automaticky pomocou nástroja Gitlab CI/CD po tom, čo sú zmeny zrevidované a zlúčené do patričnej vetvy. Nástroj automaticky vytvorí Docker obraz a vytvorí SSH pripojenie so serverom, na ktorý zmeny aplikuje. Všetky chyby na serveri sú monitorované nástrojom Sentry, kde na základe hlásení sú chyby priebežne opravované.

V Android aplikácii Educhild sa podarilo úspešne zintegrovať komunikáciu so serverom a následnú synchronizáciu dát. Aplikácia tak funguje okrem plne funkčného módu off-line taktiež v móde on-line, v ktorom sa môže užívateľ prihlásiť alebo zaregistrovať na server, čím mu je umožnené využívať webovú aplikáciu, kde môže vzdialene spravovať zariadenie dieťaťa alebo vytvárať vlastné kvízy.

Webová aplikácia, pre ktorej vývoj je fungovanie servera kritické, má úspešne naimplementovanú komunikáciu so serverom. Vývoj webovej aplikácie prebiehal s pomocou dokumentácie REST API v nástroji Swagger, pomocou ktorého je možné zobraziť všetky zdroje vrátane formy požiadavky a odpovede. Pre webovú aplikáciu bol taktiež úspešne zriadený doménový

dôveryhodný SSL/TLS certifikát, ktorý sa automaticky obnovuje každé 3 mesiace a je nutný pre správne funkovanie stránky vo webovom prehliadači.

Súčasná verzia servera obsahuje 9 996 riadkov kódu od autora práce (z toho celkovo pridaných riadkov bolo 22 305 a odstránených riadkov 12 309). Android aplikácia obsahuje od autora práce 7 319 riadkov kódu (z toho 11 121 pridaných riadkov a 3 802 odstránených riadkov) v súvislosti s integráciou servera do aplikácie. Aktívnemu vývoju aplikácií bolo venované približne 6 mesiacov s časovou náročnosťou 1 MD za týždeň.

Zdrojový kód serverovej aplikácie je dostupný na adrese `https://gitlab.fit.cvut.cz/popovmil/educhild-server`, zdrojový kód Android aplikácie je dostupný na adrese `https://gitlab.fit.cvut.cz/simapet4/educhild-android`.

6.2 Budúce kroky

Napriek tomu, že aktuálny stav je funkčný a spĺňa všetky požiadavky, existujú riešenia, ktoré vedú k vyššej efektvite servera, menšiemu prenosu dát a efektívnejšie narábajú s úložiskom.

Medzi najvýznamnejšie vylepšenie servera je úprava spôsobu práce so štatistikami o navštívených aplikáciách. V súšasnosti sa okrem samotných záznamov o aplikačnej histórii vypočítavajú na serveri konkrétne štatistické údaje. Tie je následne nutné aktualizovať a premazávať v načasovaných úlohách, čo môže viezť k neaktuálnosti údajov. Štatistiky boli realizované na základe požiadaviek webovej aplikácie. V Android aplikácii sa tieto štatiky vypočítavajú priamo v zariadení na základe údajov z aplikačnej histórie, ktoré poskytuje server. To vedie k väčšej flexibilite výberu intervalov a zo servera je vždy nutné stiahnuť jedine tie záznamy o aplikačnej histórii, ktoré sa nenachádzajú na klienskom zariadení.

Keďže sa ku dňu písania práce nezačala implementácia aplikačných štatistík na webovom klientovi, dané záznamy a API nie sú využívané a API je možné na serverovej strane spolu s dátami bezpečne odstrániť. Webový klient bude využívať API pre dotazovanie záznamov o aplikačnej histórii, ktoré súčasne využíva Android aplikácia. Dané riešenie predstavuje architektúru tučného klienta, ale jedná sa o spracovanie desiatok až stoviek záznamov na jedného klienta, čo v klientskom zariadení predstavuje minimálnu záťaž, ako sa ukázalo aj pri riešení tohto problému v Android aplikácii.

V súčasnosti je Android aplikácia pripravená pre používanie práve jedným dieťaťom a rodičom. V budúcnosti je plánované rozširenie pre viacero detí, na čo je v súčasnosti serverové riešenie už navrhnuté. Jedinou nutnou úpravou bude notifikovanie Android zariadení. Pri prechode Android aplikácie na možnosť využitia viacero deťmi bude nutné prispôsobiť odosielanie notifikácií viacero zariadeniam a bude nutné na serveri spravovať všetky notifikačné tokeny zariadení.

Aktuálne riešenie REST API poskytuje vysokú granularitu systému. Je možné dotazovať jednotlivé entity zo servera, čo je v špecifických prípadoch výhodné, ale pri procesoch ako je synchronizácia dochádza k vyššiemu počtu prevolávania servera. Preto je žiaduce v zdrojoch, ktoré poskytujú menšie množstvo údajov – čo je v tomto prípade väčšina zdrojov – zlúčiť v REST API tak, aby bolo pomocou jedného prevolania získané čo najviac informácií. Príkladom je požadovanie entity dieťaťa, v ktorom je možné v odpovedi vrátiť ihneď taktiež všetky rozvrhy a monitorované aplikácie dieťaťa. Pre prípadné zavedenie zmien je pre zamedzenie chýb žiaduce začať verzovať REST API, ktoré doteraz nebolo nutné zaviezť.

Pre správne fungovanie webovej aplikácie sa z Android zariadení získavajú informácie o aplikáciách, ako sú názvy v rôznych jazykoch, kategórie a ikony. Posledným návrhom na budúcu úpravu je zavedenie získavania týchto informácií z externých zdrojov, čím sa údaje na serveri zjednotia a znemožní sa prípadná šanca na sfalšovanie údajov klientom. Tieto externé zdroje sú však spoplatnené mesačným poplatkom, a preto je žiaduce zaviezť tieto zmeny v prípade úspešnosti projektu. Server si však naďalej bude ukladať tieto dáta vo vlastnom úložisku a poskytovať ich užívateľom, a teda jediná zásadná zmena je zmena poskytovateľa údajov na serveri.

Záver

Cieľom tejto práce bol návrh a realizácia serverovej časti aplikácie pre vzdelávanie a rodičovskú kontrolu detí Educhild a následná implementácia komunikácie Android aplikácie so serverom. Základ Android aplikácie vznikol počas trvania predmetu BI-SP1, kde vznikla potreba realizície servera. Serverová časť aplikácie slúži ako podpora Android aplikácie, kde sa uchovávajú užívateľské dáta a následne je možné spravovať účet pomocou webového rozhrania alebo na inom Android zariadení.

V analytickej časti bol podrobne popísaný aktuálny stav Android aplikácie, na základe ktorej boli vytvorené požiadavky na server. Na základe analýzy bol vytvorený návrh a architektúra servera a následne prebehla samotná implementácia.

V rámci návrhu bol navrhnutý databázový model, logická a fyzická architektúra servera a rozhranie pre komunikáciu s klientskými aplikáciami. Návrh a realizácia boli v práci popísané spôsobom, ktorý okrem výsledného stavu popisuje priebežné chyby, ktorým by sa mal čitateľ pri budovaní systému vyvarovať.

Pre realizáciu boli zanalyzované konkrétne aplikačné rámce a následne bol vybraný najvhodnejší aplikačný rámec, pomocou ktorého bol zrealizovaný návrh aplikácie. Okrem samotnej aplikácie bola na serveri zavedená databáza a reverzný proxy server, pomocou ktorého bolo možné roztriediť HTTP požiadavky na aplikačný server a webovú klientskú aplikáciu. V rámci implementácie bola v Android aplikácii úspešne vytvorená komunikácia so serverom, a to na základe rôznych typov požiadaviek na server.

Okrem návrhu a realizácie servera bola v práci riešená forma vývoja softvéru, ktorej cieľom je zautomatizovať čo najviac krokov pri prevádzke servera. Bol zrealizovaný systém, pomocou ktorého sa serverová aplikácia pri každej vývojovej zmene automaticky zostaví, otestuje a nasadí na server. Serverová aplikácia bola následne otestovaná pomocou jednotkových a integračných testov, a to so zreteľom na funkčné aj nefunkčné požiadavky.

Výsledkom tejto práce je funkčná aplikácia, ktorá uskutočňuje všetky funkčné požiadavky, je nasadená na testovacom serveri a je pripravená pre použitie klientskými aplikáciami. V rámci práce bola taktiež úspešne zrealizovaná komunikácia a synchronizácia Android aplikácie Educhild so serverovou časťou. Na záver boli v práci navrhnuté zmeny pre budúci rozvoj aplikácie.

Zoznam použitej literatúry

1. KALUŽA, Marin; KALANJ, Marijana; VUKELIĆ, Bernard. A comparison of back-end frameworks for web application development. *Zbornik veleučilišta u rijeci* [online]. 2019, roč. 7, č. 1, s. 317–332 [cit. 01. 03. 2021]. Dostupné z DOI: 10.31784/zvr.7.1.10.
2. GRÜNWALDT, Jan-Marius. *A Comparison of Modern Backend Frameworks Protections against Common Web Vulnerabilities* [online]. Tufts University, Medford MA, 2019 [cit. 01. 03. 2021]. Dostupné z: http://www.cs.tufts.edu/comp/116/archive/fall2019/jgrunwaldt.pdf.
3. JEON, Taewoong; LEE, Sungyoung; SEUNG, Hyonwoo. Increasing the testability of object-oriented frameworks with built-in tests. In: *International Workshop on Advanced Internet Services and Applications*. 2002, s. 169–182. ISBN 978-3-540-43968-4.
4. RAIBLE, Matt. *How I Calculated Ratings for My JVM Web Frameworks Comparison* [online] [cit. 13. 02. 2021]. Dostupné z: https://raibledesigns.com/rd/entry/how_i_calculated_ratings_for.
5. MITTAL, Kanika. *Backend Frameworks with the most stars on GitHub in 2020* [online] [cit. 03. 03. 2021]. Dostupné z: https://medium.com/@kanikamittal0606/backend-frameworks-with-the-most-stars-on-github-in-2020-14d319b2c85a.
6. LARAVEL LLC. *About Laravel* [online] [cit. 07. 03. 2021]. Dostupné z: https://github.com/laravel/laravel.
8. KUMAR, Pardeep. *An Overview of the Best Laravel Security Practices* [online] [cit. 06. 03. 2021]. Dostupné z: https://www.cloudways.com/blog/best-laravel-security-practices/.
9. LARAVEL LLC. *Laravel - The PHP Framework For Web Artisans* [online] [cit. 06. 03. 2021]. Dostupné z: https://laravel.com/docs/8.x.

10. KUCHLING, A.M. *PEP 206 – Python Advanced Library* [online] [cit. 07. 03. 2021]. Dostupné z: https://www.python.org/dev/peps/pep-0206/.

11. DJANGO SOFTWARE FOUNDATION. *Django* [online]. Lawrence, Kansas [cit. 07. 03. 2021]. Dostupné z: https://docs.djangoproject.com/en/3.1/.

12. COPPERWAITE, Matt; LEIFER, Charles. *Learning Flask Framework*. Packt Publishing Ltd, 2015. ISBN 9781783983360.

13. AUT. KOL. *Spring Framework Reference Documentation* [online] [cit. 13. 03. 2021]. Dostupné z: https://docs.spring.io/spring-framework/docs/3.2.19.BUILD-SNAPSHOT/spring-framework-reference/pdf/spring-framework-reference.pdf.

14. PIVOTAL SOFTWARE, INC. *Spring Trademark Guidelines* [online] [cit. 13. 03. 2021]. Dostupné z: https://spring.io/trademarks.

15. AUT. KOL. *Spring Boot Reference Documentation* [online] [cit. 13. 03. 2021]. Dostupné z: https://docs.spring.io/spring-boot/docs/2.4.3/reference/pdf/spring-boot-reference.pdf.

16. GOOGLE LCC; OPEN HANDSET ALLIANCE. *Frame rate* [online] [cit. 14. 03. 2021]. Dostupné z: https://developer.android.com/guide/topics/media/frame-rate.

17. GOOGLE LCC; OPEN HANDSET ALLIANCE. *Guide to background processing* [online]. 2020 [cit. 14. 03. 2021]. Dostupné z: https://developer.android.com/guide/background.

18. GOOGLE LCC; OPEN HANDSET ALLIANCE. *Running Android tasks in background threads* [online]. 2020 [cit. 20. 03. 2021]. Dostupné z: https://developer.android.com/guide/background/threading.

19. GOOGLE LCC; OPEN HANDSET ALLIANCE. *Kotlin coroutines on Android* [online] [cit. 14. 03. 2021]. Dostupné z: https://developer.android.com/kotlin/coroutines.

20. DANĚČEK, Jiří. *Programování v jazyku Kotlin: Korutiny* [online] [cit. 17. 03. 2021]. Dostupné z: https://docs.google.com/presentation/d/17onpHU9YSLrVq56Xh4Ym1gILrf4OeqKc2HhtZDURDuE.

21. KOTLIN FOUNDATION. *CoroutineScope* [online] [cit. 14. 03. 2021]. Dostupné z: https://kotlin.github.io/kotlinx.coroutines/kotlinx-coroutines-core/kotlinx.coroutines/-coroutine-scope/index.html.

22. NOVOTNÝ, Róbert. *Korutiny v Kotline* [online] [cit. 10. 05. 2021]. Dostupné z: https://novotnyr.github.io/scrolls/korutiny-v-kotline/.

23. GOOGLE LCC; OPEN HANDSET ALLIANCE. *AsyncTask* [online] [cit. 15. 03. 2021]. Dostupné z: https://developer.android.com/reference/android/os/AsyncTask.

24. GOOGLE LCC; OPEN HANDSET ALLIANCE. *Bound services overview* [online]. 2020 [cit. 20. 03. 2021]. Dostupné z: https://developer.android.com/guide/components/bound-services.

25. GOOGLE LCC; OPEN HANDSET ALLIANCE. *Services overview* [online]. 2021 [cit. 20. 03. 2021]. Dostupné z: https://developer.android.com/guide/components/services.

26. GOOGLE LCC; OPEN HANDSET ALLIANCE. *Schedule tasks with WorkManager* [online]. 2021 [cit. 20. 03. 2021]. Dostupné z: https://developer.android.com/topic/libraries/architecture/workmanager.

27. GOOGLE LCC; OPEN HANDSET ALLIANCE. *Optimize for Doze and App Standby* [online]. 2020 [cit. 20. 03. 2021]. Dostupné z: https://developer.android.com/training/monitoring-device-state/doze-standby.

28. GOOGLE LCC. *Firebase Products / Build* [online] [cit. 10. 04. 2021]. Dostupné z: https://firebase.google.com/products-build.

29. GOOGLE LCC. *FirebaseToken* [online]. 2020 [cit. 13. 02. 2021]. Dostupné z: https://firebase.google.com/docs/reference/admin/java/reference/com/google/firebase/auth/FirebaseToken.

30. GOOGLE LCC; OPEN HANDSET ALLIANCE. *Set the application ID* [online]. 2020 [cit. 13. 02. 2021]. Dostupné z: https://developer.android.com/studio/build/application-id.

31. GOOGLE LCC; OPEN HANDSET ALLIANCE. *ApplicationInfo* [online]. 2020 [cit. 13. 02. 2021]. Dostupné z: https://developer.android.com/reference/android/content/pm/ApplicationInfo.

32. GOOGLE LCC. *AbstractFirebaseAuth* [online]. 2021 [cit. 10. 04. 2021]. Dostupné z: https://firebase.google.com/docs/reference/admin/java/reference/com/google/firebase/auth/AbstractFirebaseAuth.

33. GOOGLE LCC. *Manage Users in Firebase* [online]. 2021 [cit. 10. 04. 2021]. Dostupné z: https://firebase.google.com/docs/auth/android/manage-users.

34. *What is a Container?* [online] [cit. 10. 05. 2021]. Dostupné z: https://www.docker.com/resources/what-container.

35. RAD, Babak Bashari; BHATTI, Harrison John; AHMADI, Mohammad. An introduction to docker and analysis of its performance. *International Journal of Computer Science and Network Security (IJCSNS)*. 2017, roč. 17, č. 3, s. 228. ISSN 1738-7906.

36. *Overview of Docker Compose* [online] [cit. 10. 05. 2021]. Dostupné z: https://docs.docker.com/compose/.

37. *Why use PostgreSQL?* [online] [cit. 10. 05. 2021]. Dostupné z: https://www.postgresql.org/about/.

38. MAYILYAN, Aren K.; HOVSEPYAN, Levon M. Research and Comparative Analysis of Build Technology Programming in Java [online]. 2018, s. 107–110 [cit. 10. 05. 2021]. Dostupné z: http://compsci.asj-oa.am/925/1/107%2D110.pdf.

39. F5, INC. *What is NGINX?* [online] [cit. 10. 05. 2021]. Dostupné z: https://www.nginx.com/resources/glossary/nginx/.

40. BAUER, Christian; KING, Gavin; GREGORY, Gary. *Java persistence with Hibernate*. Manning, 2016. ISBN 1617290459, 9781617290459.

41. MUMBAIKAR, Snehal; PADIYA, Puja. Web Services Based On SOAP and REST Principles. 2013, roč. 3, č. 5. ISSN 2250-3153.

42. GUTH, Ondřej. *RESTful web services* [online] [cit. 10. 04. 2021]. Dostupné z: https://courses.fit.cvut.cz/BI-TJV/lectures/files/bi-tjv-p-rest.pdf.

43. NARUMOTO, Masashi. *Návrh webových rozhraní API* [online] [cit. 11. 04. 2021]. Dostupné z: https://docs.microsoft.com/cs-cz/azure/architecture/best-practices/api-design.

44. KOTLIN FOUNDATION. *Data classes* [online] [cit. 11. 05. 2021]. Dostupné z: https://kotlinlang.org/docs/data-classes.html.

45. *Working with Spring Data Repositories* [online] [cit. 11. 05. 2021]. Dostupné z: https://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html.

46. EBERT, Christof; GALLARDO, Gorka; HERNANTES, Josune; SERRANO, Nicolas. DevOps. 2016, roč. 33, č. 3. ISSN 1937-4194.

47. DATANYZE. *Error and Exception Monitoring Market Share* [online] [cit. 18. 04. 2021]. Dostupné z: https://www.datanyze.com/market-share/error-and-exception-monitoring--422.

Dodatok **A**

Zoznam použitých skratiek

API	Application Programming Interface
BI-SP1	Bakalársky predmet Softwatový týmový projekt 1 vyučovaný na FIT ČVUT v Prahe
BI-SP2	Bakalársky predmet Softwatový týmový projekt 2 vyučovaný na FIT ČVUT v Prahe
CRUD	Create, Read, Update, Delete
ČVUT	České vysoké učení technické
DAO	Data Access Object
DDL	Data Definition Language
DTO	Data Transfer Object
FIT	Fakulta informačních technologií
GUI	Graphical User Interface
HATEOAS	Hypermedia as the Engine of Application State
HTTPS	Hypertext Transfer Protocol Secure
JDBC	Java Database Connectivity
JPA	Jakarta Persistence
JSON	JavaScript Object Notation
MIME	Multipurpose Internet Mail Extensions
MVC	Model-view-controller

MVT	Model-view-template
MVVM	Model-view-viewmodel
OS	Operačný systém
OWASP	The Open Web Application Security Project
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SpEL	The Spring Expression Language
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Sockets Layer
TSL	Transport Layer Security
URI	Uniform Resource Identifier
UUID	Universally unique identifier
XSS	Cross-site scripting

Dodatok B

Slovník pojmov

Android	operačný systém pre mobilné zariadenia založený na Linuxovom jadre
Aplikačný rámec	angl. framework, softvér slúžiaci ako podpora pri programovaní, vývoji a organizácii iných softvérových produktov. Môže obsahovať podporné programy, knižnice API, podporu pre návrhové vzory alebo odporúčané postupy pri vývoji
Autentizačný token	unikátny kód využívaný na komunikáciu so serverom
Certifikačná autorita	vydávateľ digitálnych certifikátov potvrdzujúci autenticitu dát
Docker	nástroj, ktorého cieľom je poskytnúť jednotné rozhranie pre izoláciu aplikácií do kontajnerov v rôznych prostrediach
Firebase	platforma vyvinutá spoločnosťou Google pre vytváranie mobilných a webových aplikácií
Maven	nástroj pre správu, riadenie a automatizáciu zostavenia aplikácií
Mock	falošný simulovaný objekt, ktorý riadeným spôsobom napodobňuje chovanie reálneho objektu
Model-view-controller	softvérová architektúra, ktorá rozdeľuje dátový model aplikácie, užívatelské rozhranie a riadiacu logiku do troch nezávislých komponentov tak, že modifikácia niektorého z nich má minimálny vplyv na ostatné

Objektovo-relačné mapovanie	programovacia technika zaisťujúca automatickú konverziu dát mezdi relačnou databázou a objektovo orientovaným jazykom
Relácia	angl. session, dočasná a interaktívna výmena informácií medzi komunikujúcimi zariadeniami. Relácia sa vytvorí v určitom okamihu a v neskoršom okamihu sa ukončí
WorkManager	knižnica pre OS Android, ktorá spúšťa odloženú prácu na pozadí, keď sú splnené podmienky pre spustenie práce

Dodatok C

Konfiguračný súbor pre nástroj Docker Compose

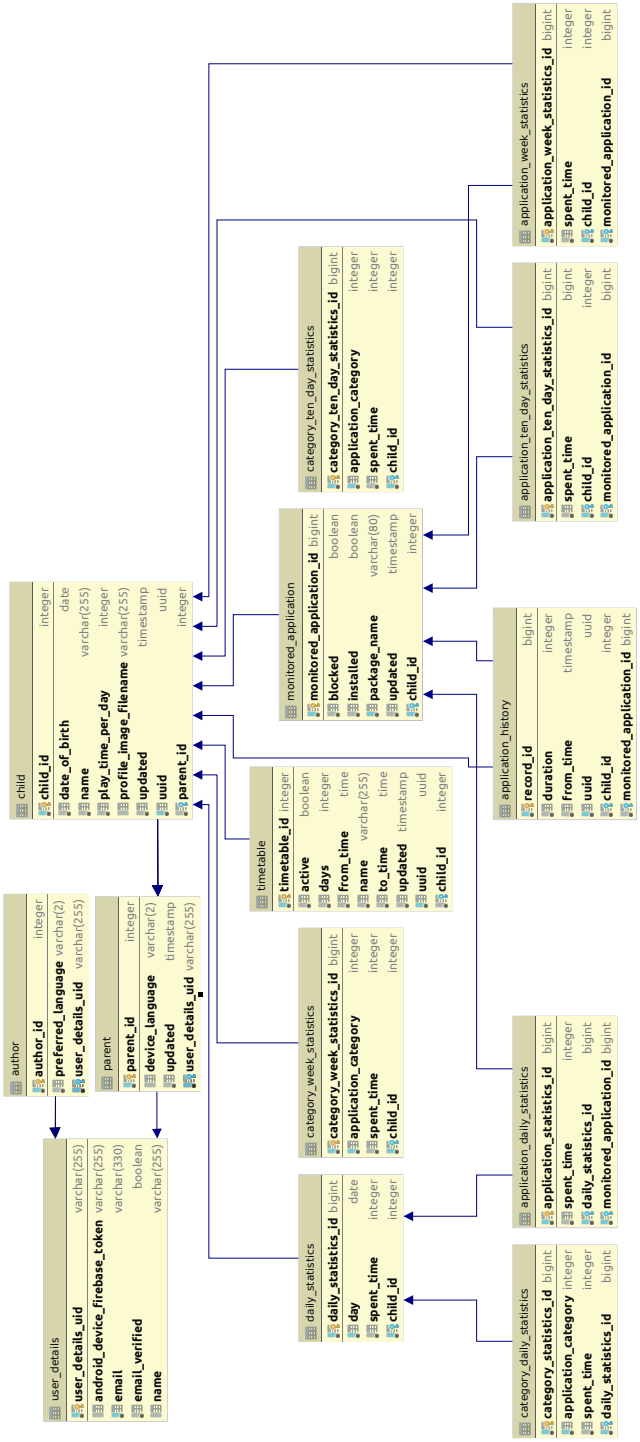
```
version: '3.8'

services:
  db:
    image: postgres:12.3
    container_name: db
    restart: always
    volumes:
      - my-db:/var/lib/postgresql/data
    env_file:
      - ./files/prod.env
    networks:
      - backend

  app:
    image: gitlab.fit.cvut.cz:5000/popovmil/educhild-server:test
    container_name: app
    environment:
      - SPRING_PROFILES_ACTIVE=server
      - GOOGLE_APPLICATION_CREDENTIALS=/files/firebase.json
    volumes:
      - /home/educhild/server/files:/files
      - my-app:/app/media
    depends_on:
      - db
    networks:
```

```
27        - backend
28        - web
29
30    web:
31      image: gitlab.fit.cvut.cz:5000/stejsj26/educhild-web:development
32      container_name: web
33      networks:
34        - web
35
36    nginx:
37      image: nginx:latest
38      container_name: nginx
39      volumes:
40        - ./nginx.conf:/etc/nginx/nginx.conf
41        - ./.htpasswd:/etc/nginx/.htpasswd
42        - /etc/letsencrypt/:/etc/letsencrypt/
43        - my-nginx:/etc/nginx/logs
44      ports:
45        - 443:443
46        - 80:80
47      depends_on:
48        - app
49        - web
50      networks:
51        - web
52
53  volumes:
54    my-db:
55    my-app:
56    my-nginx:
57
58  networks:
59    backend:
60    web:
61
```

Zdrojový kód C.4: Ukážka konfigurácie nástroja Docker Compose pre testovacie prostredie.

Dodatok D

Databázový model



Obrázok D.1: Databázový model s najdôležitejšími entitami v rámci administratívnej a rodičovskej časti aplikácie

Dodatok E

Obsah priloženého média

- `readme.md` ..... stručný popis obsahu média
- `doc`
  - `android` ..... adresár s dokumentáciou Android aplikácie
  - `server` ..... adresár s dokumentáciou serverovej aplikácie
- `src`
  - `docker-compose.yml` ..... konfiguračný súbor docker compose
  - `Dockerfile` ..... súbor pre zostavenie serverovej aplikácie
  - `gitlab-ci.yml` ..... konfiguračný súbor pre Gitlab CI/CD
  - `nginx.conf` ..... konfiguračný súbor pre nástroj Nginx
- `test`
  - `android_test_report.html` ..... výsledky testov Android aplikácie
  - `server_test_report.html` ..... výsledky testov serverovej aplikácie
- `img`
  - `gitlab_pipeline_jobs.png` .. ukážka výsledku práce v Gitlab CI/CD
- `BP_Popovic_Milos_2021.pdf` ..... text práce vo formáte PDF