

Gymnázium Lipany
Komenského 13

ROČNÍKOVÁ PRÁCA

Elektronický obchod pre smartfóny Keksobox

Lipany
2019

Riešitelia
Sebastián Petrík

Ročník štúdia: **tretí**

Gymnázium Lipany
Komenského 13

ROČNÍKOVÁ PRÁCA

Elektronický obchod pre smartfóny Keksobox

Lipany
2019

Riešitelia
Sebastián Petrík

Ročník štúdia: **tretí**

Konzultant
Ing. Štefan Palušek

Čestné vyhlásenie

Vyhlasujem, že celú prácu na tému Elektronický obchod pre smartfóny Keksobox som vypracoval samostatne, s použitím uvedenej literatúry. Som si vedomý zákonných dôsledkov, ak v nej uvedené údaje nie sú pravdivé.

Lipany, 21. február 2019

.....

vlastnoručný podpis

Obsah

Úvod.....	5
1 Ciel' a metodika práce.....	6
2 Analýza, pojmy a nástroje	7
2.1 Kotlin	7
2.2 GIT a GitLab	7
2.3 Gradle	7
2.4 Jenkins	8
2.5 Javalin.....	8
2.6 MongoDB	8
2.7 API a Web API	8
2.8 VPS	9
3 Server.....	10
3.1 Umiestnenie a automatizácia	10
3.2 Konfigurácia servera.....	11
3.3 Jadro servera a štruktúra	11
3.4 Model.....	12
3.5 Systém používateľov	13
4 API.....	14
4.1 Používatelia	15
4.2 Položky	15
4.3 Kategórie	15
4.4 Objednávanie	16
4.5 Záloha databázy	16
4.6 Obrázky	17
4.7 API správ o chybe.....	17
5 Aplikácia pre Android	18
5.1 Nástroje a knižnice	18
5.2 Aktivity	18
5.3 Hlavná aktivita a fragmenty.....	20
5.4 Model.....	20
5.5 Súbežnosť a Kotlin Coroutines.....	21
5.6 Pomocné funkcie	21
5.7 Komunikácia so serverom	22
6 Záver práce	23
Zoznam použitej literatúry	24

Úvod

V našej škole je už dlhší čas ustálená myšlienka študentského bufetu, ktorý zvyknú prevádzkovať žiaci predmetu Aplikovaná ekonómia. Tento bufet fungoval na princípe buď stacionárnom (predaj bagiet pri vstupe do školy), alebo mobilnom (zvolení „donášači“ cez veľkú prestávku navštevovali rôzne učebne a predávali žiakom za drobné peniaze rôzne potraviny, väčšinou bagety a cukrovinky, ktoré nosili v taškách). Tento systém bufetu bol vždy dobrým nápadom, keďže naša škola pozostáva z dvoch budov a žiaci z hornej budovy by inak museli zbytočne chodiť do dolnej budovy, v ktorej sa dané produkty predávali.

Tento systém však samozrejme nie je dokonalý. V prvom rade majú žiaci málo času na výber produktov. Mobilné bufety majú tiež výrazne limitovanú ponuku a ľahko sa vypredajú. Žiak ktorý by mal napríklad chuť na bagetu by si ju nemohol kúpiť, pretože ostatní žiaci už všetky bagety vykúpili. Najväčšia nevýhoda spočíva v spôsobe platby. Na škole je mnoho ľudí, ktorí pri sebe práve vtedy nemajú peniaze. Niektorí ich zabudnú, iní ich potrebujú na niečo iné, ďalší ich zasa nikdy nenosia pri sebe. Platenie v hotovosti je tiež časovo a logisticky náročné a trvanie veľkej prestávky (15 minút) môže byť pre donášačov bufetovej firmy zjavne obmedzujúci čas, čo môže odraziť v ich vyššom potrebnom počte.

Tento bufet sme sa rozhodli vylepšiť vývojom softvérového systému na princípe interaktívneho elektronického obchodu – Keksoboxu. Na jeho vývoj sme dostali povolenie od študentskej spoločnosti gyMind, ktorá v školskom roku 2018/19 podniká na našej škole. Systém zahŕňa internetový server a klientske aplikácie na rozličných platformách (Android/iOS/Web). Študenti si môžu u pracovníkov firmy vytvoriť účty a zároveň nabiť virtuálnu menu obchodu – mince.

Po stiahnutí aplikácie Keksobox sa do nej môžu prihlásiť a pomocou mincí si s ľahkosťou objednať tovar z ponuky firmy. Po objednaní tovaru si objednávku môžu prezrieť pracovníci firmy, ktorí majú na starosti bufet. Cez čas donášania (napr. veľká prestávka) tovar rezervujú a donášajú na miesta, ktoré boli špecifikované v objednávkach. Pri vyzdvihnutí tovaru donášač naskenuje z telefónu zákazníka QR kód, a tým sa objednávka potvrdí. Systém zároveň umožňuje vedieť presný počet kusov tovaru, ktorý potrebuje firma vyrobiť.

V teoretickej časti budeme rozoberať pojmy, ktoré súvisia s vývojom systému. V praktickej časti popíšeme a rozoberieme konkrétnu implementáciu servera a aplikácie pre systém Android.

1 Cieľ a metodika práce

Cieľom našej práce bolo vytvoriť a uviesť do prevádzky server a Androidovú aplikáciu elektronického obchodu KeksoBox. Ich vývoj bol sprevádzaný konzultáciou s pracovníkmi a riaditeľmi firmy gyMind, ktorí nám poskytli aktívnu odozvu a preferencie pri používaní systému.

Hlavnými cieľmi boli:

- Využiť prevažne moderné technológie a širokú škálu nástrojov, ktoré pomáhajú pri softvérovom vývoji.
- Naprogramovať rýchly a zabezpečený server, ktorý je schopný komunikovať s klientami fungujúcich na rozličných platformách.
- Nastaviť zabezpečenú databázu, ku ktorej má server prístup.
- Umiestniť server na VPS (virtuálny privátny server) s operačným systémom Linux a nastaviť ho ako samostatnú službu v systéme.
- Naprogramovať klientsku aplikáciu pre Android, ktorá by zahŕňala ako rozhranie pre používateľa, tak aj pre donášača a administrátora.
- Sprístupniť aplikáciu pre zákazníkov v obchode Google Play.

Pri plnení týchto cieľov sme postupovali takto :

- Návrh sme prezentovali riaditeľom firmy, od ktorých sme potom dostali súhlas na vývoj.
- Vytvorili sme funkčný server.
- Naprogramovali sme aplikáciu a prezentovali sme ju firme a žiakom školy.
- Systém sme niekoľkokrát vylepšili vďaka postrehom žiakov.
- Prešli sme obdobím testovania pomocou zvolených žiakov školy a následne opravili všetky chyby, ktoré sme našli.
- Aplikáciu sme sprístupnili na Google Play, vyškolili sme pracovníkov firmy a nasadili systém do prevádzky.

2 Analýza, pojmy a nástroje

Pri vytváraní ako aplikácie, tak aj servera sme sa snažili využívať prevažne moderné technológie. Naším prvým problémom bola potreba navrhnuť model systému. Rozhodli sme sa pre samostatný server, ktorý by sa dal využiť s klientami na rôznych platformách. Zároveň bolo potrebné vybrať správne nástroje pre vývoj.

2.1 Kotlin

Kotlin je multiplatformový, objektovo orientovaný a staticky typovaný programovací jazyk pre všeobecné využitie s automatickou detekciou typov. Bol dizajnovaný na úplnú spoluprácu s kódom jazyka Java a jeho JVM (Java Virtual Machine) verzia závisí na knižniciach jazyka Java, pričom automatická detekcia typov a jeho iné vlastnosti umožňujú stručnejšiu syntax. Aj keď je (podobne ako Java) objektovo orientovaný, podporuje aj funkcionálne programovanie. Kotlin je zároveň oficiálny jazyk pre vývoj aplikácií pre operačný systém Android. (*Wikipedia, 2019*)

Jazyk Kotlin sme si vybrali, pretože je to moderný a aktívne vyvíjaný jazyk a umožnil nám vývoj ako servera, tak aj aplikácie pre Android.

2.2 GIT a GitLab

GIT je distribuovaný systém riadenia verzií pre sledovanie zmien v zdrojovom kóde počas vývoja softvéru. Bol dizajnovaný pre koordinovanie práce medzi programátormi, ale môže byť použitý na sledovanie zmien ľubovoľných typov súborov. (*Wikipedia, 2019*)

Služba GitLab umožňuje bezplatné ukladanie a správu GIT repozitárov (verejných aj súkromných) na jej internetovom úložisku.

2.3 Gradle

Gradle je open-source nástroj pre automatizáciu zostavovania programu využívaný najmä pre projekty napísané v jazykoch bežiacich na JVM (Java, Groovy, Kotlin a Scala). Je možné ho konfigurovať pomocou úloh, ktoré kompilujú kód, testujú ho, vytvárajú dokumentáciu a oveľa viac. (*MacMurray, 2018*)

2.4 Jenkins

Jenkins je open-source automatizačný server napísaný v jazyku Java. Jenkins pomáha automatizovať časť procesu vývoja softvéru, ku ktorej nie je potrebný človek, pomocou priebežnej integrácie (neustálemu zostavovaniu pri zmene kódu, testovaníu a aplikovaníu do produkcie). Je schopný zostavovať projekty typu Apache Ant, Apache Maven a Gradle. Zároveň môže spúšťať systémové príkazy. (*The Mightywomble, 2018*)

2.5 Javalin

Javalin je moderný nenáročný open-source framework pre vývoj serverov v jazykoch Java a Kotlin. Jeho hlavné ciele sú jednoduchosť, príjemný a jednoduchý vývoj a spolupráca jazykov Kotlin a Java. Je založený na Java serveri Jetty a podporuje rôzne technológie, ako WebSockets, HTTP2 a asynchrónne HTTP požiadavky. (*David Åse, 2019*)

Tento framework sme si zvolili pre jeho flexibilitu a taktiež preto, lebo sme sa sami na jeho vývoji podieľali.

2.6 MongoDB

MongoDB je open-source systém databázy, ktorý využíva objektovo orientovaný model dát a neštruktúrovaný jazyk pre vyhľadávanie. Je založený na princípe NoSQL – na rozdiel od SQL nevyužíva tabuľky a vzťahový systém databázy, ale tzv. kolekcie a dokumenty. Umožňuje, aby dokumenty mali rozdielne atribúty a štruktúry. Ich štruktúra je podobná formátu JSON. Umožňuje väčšiu flexibilitu a pohodlnejšiu správu databázy. Jeho klientske knižnice sú dostupné pre rôzne platformy a jazyky. (*Intellipaat, 2019*)

2.7 API a Web API

Rozhranie pre programovanie aplikácie (API, Application programming interface) je zbierka definovaných metód komunikácie medzi rozličnými komponentmi. Umožňuje jednoduchší vývoj programu, pretože poskytuje všetky jeho potrebné jednotlivé súčasti, ktoré sú potom spojené do celku programátorom. (*Wikipédia, 2019*)

Pojmom Web API sa označuje API, ktoré využíva HTTP protokol. Je založené na systéme posielania požiadaviek (request) a odpovedí (response) medzi serverom a klientom, pričom na prenos dát môže využívať formáty ako JSON a XML.

2.8 VPS

Virtuálny privátny server (VPS, Virtual private server) je virtuálny server, ktorý z hľadiska používateľa pôsobí ako dedikovaný server, ale v skutočnosti je nainštalovaný na jednom počítači spolu s ostatnými VPS. Každý VPS môže mať vlastný operačný systém, ktorý beží vo virtuálnom stroji hostiteľského počítača.

3 Server

Server je centrálnym bodom celého systému. Plní úlohu komunikácie s klientami, spravuje databázu, používateľov, zabezpečenie. Na vývoj serverov sa často používajú technológie ako PHP, Node.js a ASP.NET. My sme však zvolili platformu JVM, pretože nám záležalo na výkonnosti a bezpečnosti. Keďže však využívame jazyk Kotlin a framework Javalin, tento server je v určitých bodoch dokonca jednoduchší na vývoj, než s použitím iných technológií. Každá komunikácia medzi serverom a klientmi je zabezpečená pomocou šifrovania TLS (verzia 1.2).

Pre jeho vývoj sme využili vývojové prostredie IntelliJ IDEA, GIT klient GitHub Desktop, program na testovanie HTTP požiadaviek Insomnia REST a správcu MongoDB databázy Robo 3T. Na VPS sme sa napájali pomocou SSH klienta MobaXterm.

3.1 Umiestnenie a automatizácia

Server spolupracuje s jednotlivými súčasťami systému:

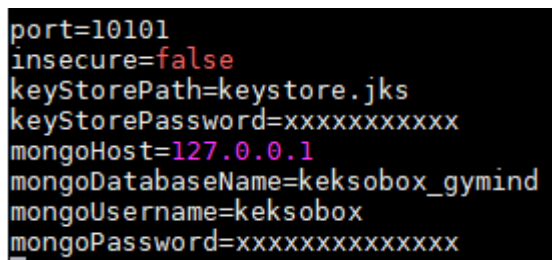
- MongoDB server – zabezpečený databázový MongoDB server umiestnený na VPS, ku ktorému sa server pripája pomocou vlastného dedikovaného MongoDB používateľa.
- Automatizačný server Jenkins – umožňuje automatické načítanie kódu z GIT repozitára, jeho následné zostavenie a vytvorenie JAR súboru (spustiteľná aplikácia pre JVM) priamo na VPS.
- Služba systemd – server je uložený vo svojom vlastnom priečinku a je prepojený s vlastnou linuxovou systemd službou, ktorá umožňuje jeho bežanie v pozadí systému a zároveň správu procesu a záznamov (logs).

Celý kód servera je umiestnený v jeho Gradle projekte, čo umožňuje jeho zostavenie na prakticky ľubovoľnom systéme s nainštalovaným Gradle a JVM. Jeho GIT repozitár využíva 2 vetvy – dev a master. Pri zlúčení vetvy dev do vetvy master notifikuje GitLab náš Jenkins server, ktorý stiahne najnovší kód servera a spustí zostavu. Po úspešnej zostave zároveň nakopíruje JAR súbor do priečinka servera a reštartuje ho (pomocou systemd).

Na VPS sme umiestnili dva rôzne servery – vývojový a produkčný. Vývojový využívame na testovanie a vývoj a využíva automatizačný server. Produkčný automatizáciu nevyužíva, uvedenie nového vydania do produkcie musí byť spravované manuálne. Zároveň je možné využívať viacero produkčných serverov naraz (pre rozdielne firmy).

3.2 Konfigurácia servera

Pre spustenie servera sú okrem JAR súboru potrebné aj konfiguračné súbory umiestnené v rovnakom adresári. Hlavným konfiguračným súborom je *config.properties*, v ktorom je nastavený port, cesta ku TLS certifikátu, heslo ku certifikátu, adresa MongoDB servera, názov databázy, prihlasovacie údaje pre MongoDB a iné nastavenia. Potrebný je taktiež súbor TLS/SSL certifikátu vo formáte Java Keystore (.jks), ktorý umožní šifrovanie. Pre vývojové účely na lokálnom počítači však nie je potrebný (nastavením nezabezpečeného módu *insecure=true*).



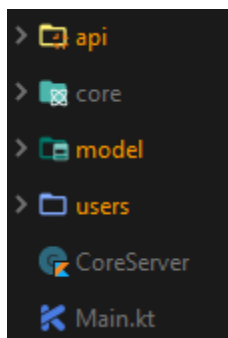
```
port=10101
insecure=false
keyStorePath=keystore.jks
keyStorePassword=xxxxxxxxxxx
mongoHost=127.0.0.1
mongoDatabaseName=keksobox_gymind
mongoUsername=keksobox
mongoPassword=xxxxxxxxxxxxxxxx
```

Obr. 1 Súbor *config.properties*

3.3 Jadro servera a štruktúra

Kód servera je roztriedený do balíčkov. V koreňovom balíčku sa nachádzajú dva súbory:

- Main - načíta konfiguračný súbor a vytvorí CoreServer objekt.
- CoreServer – hlavná trieda servera, ktorá využíva konfiguračný objekt a zabezpečuje inicializáciu Javalin inštancie, MongoDB databázy a routing (napojenie ciest) API.



Obr. 2 Štruktúra kódu servera

V balíčku *core* sa nachádzajú jadrové súčasti servera:

- Config – model konfigurácie.
- ConfigLoader – zabezpečí načítanie konfiguračného súboru.
- CoreDatabase – vytvorí Mongo klient, otestuje pripojenie, nastaví databázu (indexy) a umožňuje jednoduchý prístup ku kolekciám databázy.
- CoreLogger – jednoduchý konzolový SLF4J logger.
- OrderPasswordGenerator – generátor hesiel pre objednávky.
- ServerMetadata – obsahuje pomocné dáta servera, ako číslo jeho verzie a názvy priečinkov, ktoré má využívať.
- ServerState – objekt s aktuálnym stavom servera, zabezpečuje jeho načítanie a ukladanie do súboru *state.json* a zachováva stav servera aj po jeho vypnutí.

3.4 Model

V balíčku *model* sa nachádzajú triedy rôznych objektov, ktoré sa využívajú v serveri. Objekty tried ako *ShopOrder* a *ShopItem* sú ukladané v databáze, ale sú využívané aj ako protokol pre prenos dát v API pomocou formátu JSON (Javalin zabezpečuje parsovanie). V podbalíčku *apimodel* sú triedy iba na účel prenosu dát.

3.5 Systém používateľov

Balíček *users* obsahuje kód na správu používateľov. Pre skoro každú požiadavku na API servera je potrebná autentifikácia. Server preto poskytuje nástroje na registráciu, prihlásenie, modifikáciu a autentifikáciu používateľov, ktoré realizuje trieda *UserManager*. Objekty používateľov sú uložené v databáze a definuje ich trieda *User* v balíčku *model*. Každý používateľ má meno, haš hesla, rolu, zostatok na účte (*balance*) a relácie prihlásenia (*sessions*).

Každý používateľ má rolu, ktorá obmedzuje jeho prístup k jednotlivým častiam API. Každá vyššia rola zahŕňa povolenia nižšej roly. Čísla rol sú definované v súbore *UserRoles* a využívajú sa hlavne tieto:

- Normal (obyčajný používateľ) – môže sa prihlásiť, načítať položky a kategórie v obchode, vytvoriť a zrušiť vlastnú objednávku, atď.
- Delivery (donáška) – môže dokončiť objednávky, vypnúť/zapnúť objednávanie, vytvoriť normálnych používateľov.
- Operator (operátor, správca obchodu) – môže meniť položky v obchode, registrovať nových používateľov donášky, dobíjať kredit.
- Admin (administrátor) – najvyššia rola, umožňuje registráciu operátorov, vymazanie používateľa a podobne.

Pri registrácii sa heslá používateľov nikdy neukladajú do databázy, ukladá sa iba ich haš. Na hašovanie používame algoritmus *Bcrypt*. Hašovanie umožní overenie hesla pomocou hašu, avšak pôvodné heslo sa z hašu prakticky nedá zistiť.

Pri prihlásení sa overí haš v databáze, vytvorí sa relácia prihlásenia (*session*) a používateľovi sa vráti tzv. token (kľúč), pomocou ktorého sa môže autentifikovať. Relácia definovaná v triede *UserSession* a umožňuje rýchle autentifikovanie pomocou tokenu. Pri behu servera sa relácie priebežne načítavajú a umožňujú tak autentifikáciu bez zbytočného využívania databázy.

Pri vytváraní požiadavky na API sa využíva autentifikácia typu *Basic access authentication*. Tento spôsob autentifikácie využíva HTTP header *Authorization*, v ktorom je uvedené meno a heslo (v prípade API je heslo token relácie prihlásenia). V prípade nesprávneho hesla, nesprávnej roly alebo inej chyby zabezpečenia server vracia stavový kód 401 (*Unauthorized*).

4 API

Najdôležitejšou časťou servera je jeho Web API založené na protokole HTTP. Umožňuje komunikáciu s jeho klientmi, definuje adresy a spracovanie HTTP požiadaviek, vykonáva rôzne operácie. Jeho hlavnou výhodou je univerzálnosť. Využívať ho môže prakticky ľubovoľný počítač so schopnosťou odosielať HTTP požiadavky, čo nám umožňuje využívať klientov rôznych platforiem (napr. Windows, Android, iOS, MacOS, ale aj webový prehliadač). Keďže server využíva šifrovanie TLS, všetky HTTP požiadavky na API sú zabezpečené (využíva sa protokol HTTPS).

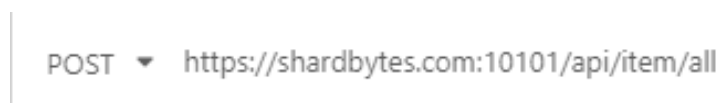
Triedy API sú uložené v balíčku *api*. Každá trieda reprezentuje určitú funkciu servera, definuje adresy a spracovanie požiadavky. API je v kóde definované pomocou funkcií vyššieho rádu patriacich do tzv. zostavovača API frameworku Javalin (Javalin API Builder), ktoré umožňujú jednoduché definovanie ciest a implementácií požiadaviek. Jednotlivé triedy definujú metódu *route*, ktorá tieto funkcie volá.

V balíčku je taktiež definovaná funkcia vyššieho rádu *apiHandler* vracajúca Javalin Handler (obsluhovač, dá sa využiť v zostavovači API), v ktorom sa najprv overuje, či je databáza správne inicializovaná, autentifikuje sa používateľ a potom sa spúšťa lambdový blok s implementáciou. Umožňuje tak jednoduché implementovanie častí API.

```
// list all images
get(apiHandler(db, mgr, UserRoles.OPERATOR) { it: Context
    it.json(File(DIRECTORY_IMAGES).list())
})
```

Obr. 4 GET požiadavka pre operátora, ktorá vráti zoznam obrázkov

Každá API trieda má určenú koreňovú adresu, od ktorej sa odvádzajú adresy jednotlivých požiadaviek.



POST ▼ https://shardbytes.com:10101/api/item/all

Obr. 5 Ukážka POST požiadavky, ktorá využíva API položiek obchodu

4.1 Používatelia

API používateľov je definované v triede UserApi poskytuje rôzne funkcie ohľadom systému používateľov – implementuje funkcionality triedy UserManager. Poskytuje tieto funkcie:

- Testovanie autentifikácie používateľa.
- Prihlásenie pomocou mena a hesla.
- Získanie informácií o používateľovi.
- Získanie informácií o inom používateľovi.
- Získanie zoznamu informácií o všetkých používateľoch.
- Nabitie kreditu.
- Registrácia nového používateľa.
- Zmena hesla.

4.2 Položky

Obchod je založený na princípe záznamov položiek a kategórií, ktoré sú uložené v databáze. Každá položka má určený identifikátor, názov, popis, kategóriu a aj identifikátor obrázku. API položiek je uložené v triede ItemApi a poskytuje tieto funkcie:

- Získanie zoznamu všetkých položiek v databáze.
- Získanie určitého zoznamu položiek podľa ich identifikátorov.
- Získanie jednej položky podľa identifikátora.
- Modifikácia alebo vytvorenie položky.
- Vymazanie položky.

4.3 Kategórie

Položky v obchode sú kategorizované. Kategórie sú umiestnené v samostatnej kolekcii a každá z nich má určený, podobne ako položka, identifikátor, názov, popis a obrázok. Jednotlivé položky na kategórie poukazujú pomocou ich identifikátora. API kategórií definuje trieda CategoryAPI a poskytuje funkcie:

- Získanie zoznamu všetkých kategórií.
- Získanie jednej kategórie podľa identifikátora.
- Modifikácia alebo vytvorenie kategórie.
- Vymazanie kategórie.
- Získanie zoznamu všetkých položiek s daným identifikátorom kategórie.
- Získanie zoznamu všetkých položiek v skrytej kategórii.

4.4 Objednávanie

Používatelia si môžu zakúpiť položky z obchodu vytvorením objednávky. Pre jej úspešné vytvorenie musia mať dostatočný kredit (mince), musí byť zapnuté objednávanie a položky musia byť na sklade. API objednávok definuje trieda OrderApi a poskytuje funkcie:

- Zistenie stavu objednávania (zapnuté/vypnuté).
- Nastavenie stavu objednávania.
- Vytvorenie objednávky.
- Vyhľadávanie všetkých objednávok (query).
- Vyhľadávanie objednávok používateľa (query).
- Vytvorenie externej objednávky – objednávky, ktorá nesúvisí priamo so systémom a jej úlohou je iba odčítať položky zo skladu (napr. priamy fyzický predaj položky).
- Získanie zoznamu všetkých externých objednávok.
- Získanie objednávky podľa jej čísla.
- Zrušenie objednávky.
- Dokončenie objednávky.

4.5 Záloha databázy

Server je schopný zálohovať celú databázu (používatelia, objednávky, položky, kategórie, záznamy o nabití kreditu, externé objednávky a správy o chybách) do súborov formátu JSON. Zálohovanie rieši trieda BackupApi.

4.6 Obrázky

Server podporuje ukladanie a sťahovanie obrázkov, ktoré využívajú kategórie a položky. Pri odoslaní obrázka ho server prekonvertuje na formát JPG, z jeho obsahu sa vytvorí heš typu SHA-256, uloží sa pod názvom rovnakým ako je heš a server tento názov vráti ako odpoveď. Týmto spôsobom sa zabezpečí jedinečnosť obrázkov a taktiež zabráni zbytočnému ukladaniu duplikátov (server obrázkov neuloží, ak obrázok s rovnakým hešom už existuje).

Obrázky sa ukladajú v priečinku *static/images* a server ich poskytuje ako statické súbory (napr. <https://shardbytes.com:10101/images/abcd.jpg>).

4.7 API správ o chybe

V prípade, že klientska aplikácia vedome zaznamenala nejakú chybu, môže serveru poslať správu o chybe, ktorá obsahuje popis chyby (napr. Java stack trace). Správu o chybe môžu odoslať iba registrovaní používatelia. API správ o chybe rieši trieda `ErrorReportApi`.

5 Aplikácia pre Android

Keďže je e-shop určený prevažne pre študentov škôl, ako hlavnú platformu sme vybrali mobilné zariadenia. My sme sa konkrétne venovali vývoju aplikácie pre operačný systém Android. Oproti webovému e-shopu umožní aplikácia omnoho pohodlnejší a rýchlejší nákup, pretože je prispôbená pre dotykový displej.

5.1 Nástroje a knižnice

Aplikáciu sme vytvorili pomocou integrovaného vývojového prostredia Android Studio (založené na prostredí IntelliJ IDEA). Pre jej vývoj sme vybrali Kotlin, pretože je oficiálnym programovacím jazykom pre systém Android, ale aj preto, lebo je v ňom napísaný server. Využitie multiplatformového jazyka je výhodné, pretože programátor nemusí zbytočne premýšľať nad odlišnými jazykovými štruktúrami a sústreď sa hlavne na riešenie problému.

Pre vývoj sme využili mnoho open-source knižníc, ktoré buď poskytujú nástroje na vývoj určitej súčasti aplikácie (napr. Jackson, Picasso, AhBottomNavigation, Kotlin Coroutines, QRGen), alebo adaptujú už existujúce nástroje jazyku Kotlin a tak umožňujú rýchlejší vývoj (napr. Jackson Kotlin Module, Kotlin Anko).

5.2 Aktivita

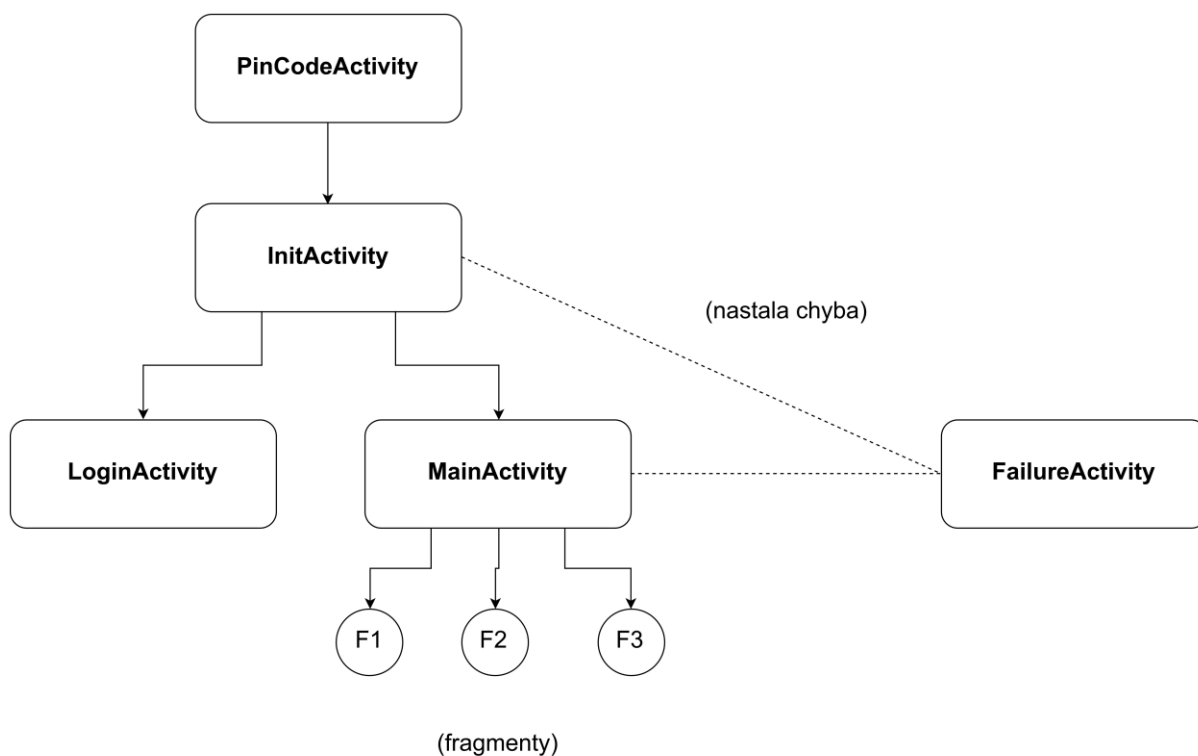
Aplikácia skladá z niekoľkých aktivít. Aktivita je základný prvok aplikácie, jej úlohou je prezentovať používateľovi používateľské prostredie - GUI (vytvorené z XML dizajnu - layoutu) a interakcia s ním. Každá aktivita má svoj životný cyklus (vytvorenie, pozastavenie, pokračovanie, zastavenie, zničenie) a je si ho vedomá. Zároveň je schopná spúšťať iné aktivity a tak umožňuje plynulú navigáciu.

Niekedy však navigáciu nemôžeme riešiť iba cez aktivity. Príkladom je spodné navigačné menu (BottomNavigationView), ktoré by muselo byť znova vytvorené v každej aktivite. Tento problém riešia fragmenty. Fragment je prvok aplikácie veľmi podobný aktivite (má aj svoj vlastný životný cyklus, vlastný XML dizajn), avšak je určený na fungovanie vnútri

už existujúcej aktivity. Aktivita fragmenty spravuje, môže s nimi komunikovať, vymieňať ich a prezentovať používateľovi taký fragment, aký momentálne potrebuje.

V našej aplikácii sa aktivity nachádzajú v balíčku activities a fragmenty v balíčku fragments. Využívame tieto aktivity:

- PinCodeActivity – táto aktivita sa spustí pri spustení aplikácie a zabezpečuje nastavenie/overenie pin kódu, pod ktorým je aplikácia uzamknutá (zabráni sa tak zneužitiu) a spúšťa InitActivity.
- InitActivity – overuje pripojenie na internet, verziu aplikácie, uložený token prihlásenia, nastavuje stav aplikácie AppState a spúšťa LoginActivity alebo MainActivity.
- LoginActivity – spustí sa v prípade potreby prihlásenia (token je nefunkčný alebo neexistuje – prvé prihlásenie), spúšťa späťne InitActivity.
- FailureActivity – jej úlohou je zobrazíť používateľovi popis chyby, ktorá v aplikácii nastala a v prípade neznámej chyby odoslať správu o chybe späť na server.
- MainActivity – hlavná aktivita.



Obr. 5 Vývojový diagram aktivít

5.3 Hlavná aktivita a fragmenty

Aktivita MainActivity je centrálnym bodom celej aplikácie. Obsahuje panel navigácie, panel nástrojov a kontajner pre fragmenty. Jej úlohou je spravovať fragmenty a vymieňať ich (pomocou kontajneru), keď používateľ zvolí položku v navigačnom menu.

Najpočetnejším komponentom aplikácie sú práve fragmenty. Každý z nich si svoju úlohu plní osobitne a je na ostatných fragmentoch nezávislý. Najdôležitejšie sú fragmenty, ktoré sú spojené s navigačným menu (ktoré vedú potom k ďalším fragmentom) :

- a) CategoriesFragment – koreňový fragment obchodu, načíta a zobrazí kategórie.
 - ItemsFragment – načíta a zobrazí položky vnútri kategórie.
 - ItemShowcaseFragment – zobrazí položku.
 - EditItemFragment – úprava položky (operátor).
- b) CartFragment – zobrazí obsah košíka.
 - CreateOrderFragment – vytvorenie a odoslanie objednávky.
- c) OrdersFragment – načíta a zobrazí objednávky používateľa.
 - OrderShowcaseFragment – zobrazí objednávku.
- d) ProfileFragment – fragment s profilom používateľa, obsahuje napr. tlačidlo pre resetovanie nastavení a v prípade vyššej roly používateľa aj administrátorské nástroje pre túto rolu.
 - DeliveryFragment – panel pre donášku, obsahuje rôzne nástroje napr. čítačku QR kódov objednávok, zoznam objednávok, zoznam položiek a registráciu.
 - DeliveryShowcaseFragment – zobrazí objednávku z pohľadu donášky, donáška tak môže potvrdiť objednávku.
 - AdminFragment – panel s nástrojmi pre operátora.
 - ChargeFragment – zabezpečuje nabitie mincí používateľa.

5.4 Model

V balíčku model sa nachádzajú dátové triedy takmer identické, ako triedy na serveri. Fakt, že jazyk Kotlin je využitý aj na Androide, nám umožňuje jednoducho využiť súbory tried zo servera. Jedinou zmenou je pridanie anotácií, ktoré umožnia rýchlejšie parsovanie.

5.5 Súbežnosť a Kotlin Coroutines

V systéme Android beží väčšina kódu aplikácie v tzv. UI vlákne. Toto vlákno má na starosti vytvárať a aktualizovať GUI, spúšťajú sa na ňom aktivity a podobne. Ak by sme chceli spustiť sieťovú komunikáciu na tomto vlákne, spôsobilo by to zasekávanie sa GUI, preto ju Android na UI vlákne nepodporuje. Je potrebné teda kód spustiť na inom vlákne súbežnom s UI vláknom. Paralelný beh viacerých kódov rieši konkurencia (angl. concurrency, súbežnosť). Na tento problém sa často využíva Androidová trieda `AsyncTask`, ale je pre ňu potrebné písať veľa zbytočného kódu.

Konkurenciu v aplikácii riešime pomocou knižnice Kotlin Coroutines. Umožňuje nám vytvárať tzv. podprogramy – korutiny, ktoré bežia súbežne, ale nepotrebujú pre seba samostatné vlákna. Androidová verzia knižnice nám poskytuje dva dôležité dispatchery korutín, ktoré ich umožňujú spúšťať na daných vláknach – Main (pre UI vlákno) a IO (pre asynchrónne operácie).

Jednotlivé korutiny o sebe vo vlákne vedia a striedajú sa v jeho využívaní. V prípade, že jedna korutina vlákno práve nevyužíva (napr. čaká na odpoveď servera po vykonaní požiadavky), vlákno sa nezablokuje, ale je naďalej využívané ostatnými korutinami.

Kotlin zároveň umožňuje funkciu označiť modifikátorom *suspend* a vytvoriť tak tzv. suspendovaciu funkciu, ktorá musí byť spustená v bloku korutiny a zároveň môže volať funkcie korutín (teda napr. prepínať medzi UI a IO vláknom).

5.6 Pomocné funkcie

Súbor `HelperExtensions` obsahuje rozličné pomocné funkcie, ktoré sú využívané v kóde celej aplikácie. Funkcie ako *ui* a *io* umožňujú jednoduchšie vykonávanie kódu v dispatcheroch korutín. Funkcia *guard* je veľmi podobná štruktúre `try/catch`, avšak pri vyhodení výnimky vnútri jej bloku sa miesto zrútenia celej aplikácie spustí funkcia *handleError*, ktorá výnimku preverí a spustí aktivitu `FailureActivity`, ktorá oznámi používateľovi informácie o chybe (napr. problém so sieťou a podobne). Pre využitie v korutine existuje jej suspendovacia alternatíva – *guardSuspend*.

Funkcie *toJson* a *fromJson* umožňujú jednoduché priame parsovanie medzi formátom JSON a objektami. Funkcia *injectImageWithPicasso* zabezpečuje jednoriadkové načítanie

a vloženie obrázka do `ImageView` komponentu a funkcia `quickPermissionRequest` požiada používateľa o udelenie Android povolení pre aplikáciu.

```
// simple json<->objects parsing as one-liner
fun Any?.toJson() = jsonMapper.writeValueAsString(value: this)
inline fun <reified T> String.fromJson() = jsonMapper.readValue<T>(content: this)

// simpler coroutine logic
suspend fun <T> ui(block: suspend CoroutineScope.() -> T) = withContext(Dispatchers.Main, block)
suspend fun <T> io(block: suspend CoroutineScope.() -> T) = withContext(Dispatchers.IO, block)
```

Obr. 6 Ukážka pomocných funkcií

5.7 Komunikácia so serverom

Na komunikáciu so serverom sme použili knižnicu `Fuel`, ktorá umožňuje jednoduché vytváranie HTTP požiadaviek. Po prihlásení používateľa na daný server aplikácia uloží token relácie do internej pamäte aplikácie (`SharedPreferences`) a do objektu stavu aplikácie (`AppState`). V balíčku `api` sa nachádzajú objekty so suspendovacími metódami, ktoré vykonávajú požiadavky na API servera (využívajú `IO` dispatcher) a sú volané v rôznych častiach aplikácie.

Na vykonávanie požiadaviek využívame vlastnú suspendovaciu funkciu `statusFuel`, ktorá vykoná požiadavku a na základe stavového kódu odpovede servera vráti buď textový reťazec s dátami, alebo vyhodí vhodnú výnimku. Táto funkcia je spolu s definíciami výnimiek a pomocnou funkciou `authWithToken` (jednoduché pridanie autentifikácie k požiadavke pomocou tokenu uloženého v `AppState` objekte) uložená v súbore `NetworkExtensions`.

6 Záver práce

Touto prácou sme chceli oboznámiť so štruktúrou a fungovaním elektronického obchodu Keksobox. Vytvorili sme funkčný a flexibilný server s aplikáciou pre Android. Tento systém momentálne využíva iba firma gyMind ale môžu ho využiť aj iné firmy – je možné ho predávať ako službu. Štandardizované API zároveň umožnilo vytvorenie klientskej aplikácie pre operačný systém iOS.

Naším najväčším problémom bol určite návrh systému a vývoj servera. Občasná neznalosť používaných nástrojov viedla k nečakaným bugom, ktoré sme však úspešne opravili.

Zároveň sme nadobudli mnoho nových skúseností v oblasti vývoja a správy serverových systémov, kryptografie, bezpečnosti a vývoja aplikácií pre operačný systém Android. Zlepšili sme sa vo využívaní nástrojov pre vývoj a vyskúšali sme si vytvorenie a spravovanie kompletného softvérového produktu v praxi.

Zoznam použitej literatúry

ÅSE, D.: *Javalin (README.md)*. 24. január 2019 [cit. 21. február 2019]. Dostupné na internete: <<https://github.com/tipsy/javalin>>.

Intellipaat. *What is MongoDB*. 11. február 2019 [cit. 21. február 2019]. Dostupné na internete: <<https://intellipaat.com/blog/what-is-mongodb/>>.

MACMURRAY, A.: *A beginners guide to Gradle*. 7. jún 2018 [cit. 21. február 2019]. Dostupné na internete: <<https://medium.com/@andrewMacmurray/a-beginners-guide-to-gradle-26212ddcfa8>>.

The Mightywomble. *Jenkins – Pipeline (Beginners guide)*. 30. január 2018 [cit. 21. február 2019]. Dostupné na internete: <<https://medium.com/@mightywomble/jenkins-pipeline-beginners-guide-f3868f715ed9>>.

Wikipédia. *Application programming interface*. 5. november 2016 [cit. 21. február 2019]. Dostupné na internete: <https://sk.wikipedia.org/wiki/Application_programming_interface>.

Wikipedia. *Kotlin (programming language)*. 21. február 2019. Dostupné na internete: <[https://en.wikipedia.org/wiki/Kotlin_\(programming_language\)](https://en.wikipedia.org/wiki/Kotlin_(programming_language))>.

Wikipedia. *Git*. 21. február 2019. Dostupné na internete: <<https://en.wikipedia.org/wiki/Git>>.

Prílohy

Zoznam príloh

Príloha A: Obrázky jednotlivých častí aplikácie a servera

Príloha B: CD médium so zdrojovým kódom

Príloha A: Obrázky jednotlivých častí aplikácie a servera

Key	Value	Type
▼ (1) Objectld("5c488e7bf7eed0729f7f6a41")	{ 7 fields }	Object
_id	Objectld("5c488e7bf7eed0729f7f6a41")	Objectld
name	keksy	String
color		String
description	Sladkosti	String
imageld	be84f930a259b4aa636c32e05735cb3c2958368f143fd347c7802a8f21ef880e.jpg	String
sorter	0	Int32
title	Sladkosti	String
▼ (2) Objectld("5c489421f7eed0729f7f6c55")	{ 7 fields }	Object
_id	Objectld("5c489421f7eed0729f7f6c55")	Objectld
name	vody	String
color		String
description	Vody a nápoje	String
imageld	1a20089626a2d9f1ccf214ffe865cb6a5d2f2db8d549315535d17952e8873b82.jpg	String
sorter	0	Int32
title	Vody a nápoje	String

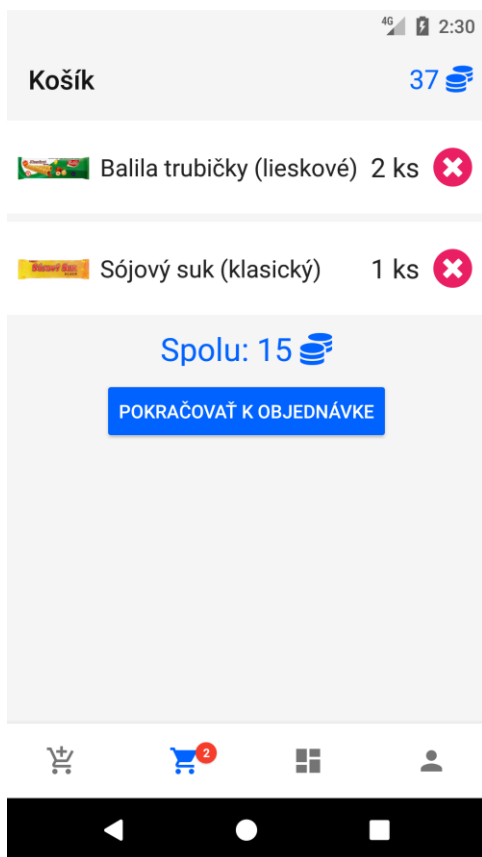
Obr. 7 Ukážka objektov kategórií v databáze



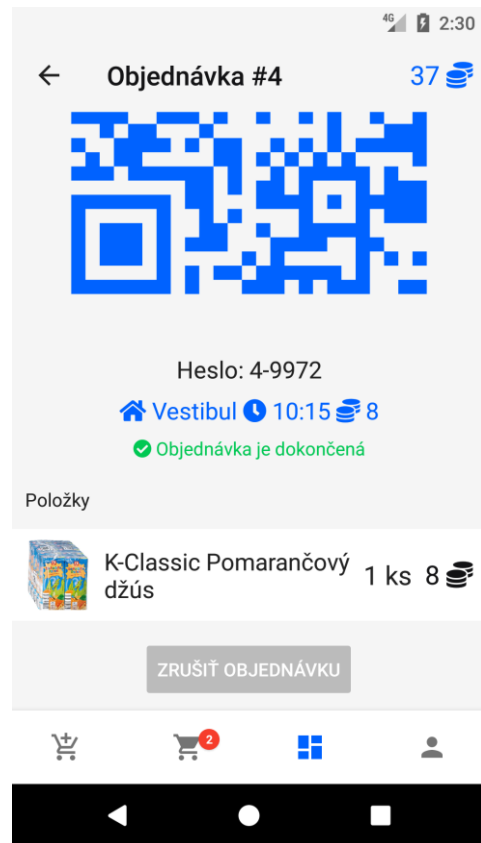
Obr.8 CategoryFragment



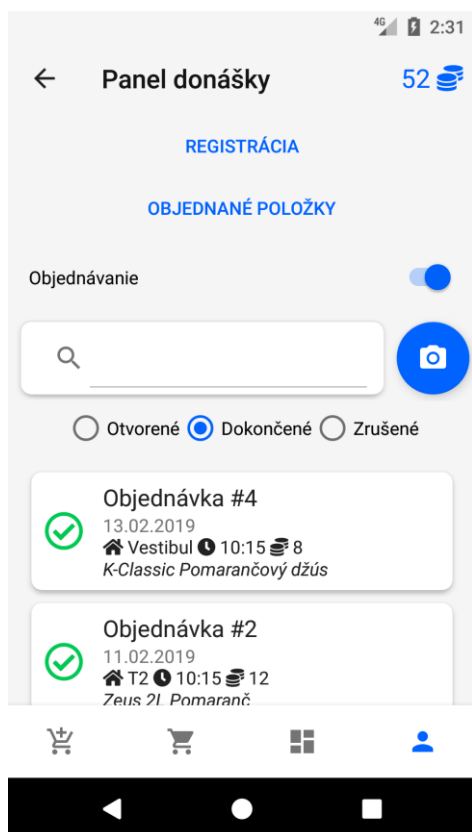
Obr.9 ItemShowcaseFragment



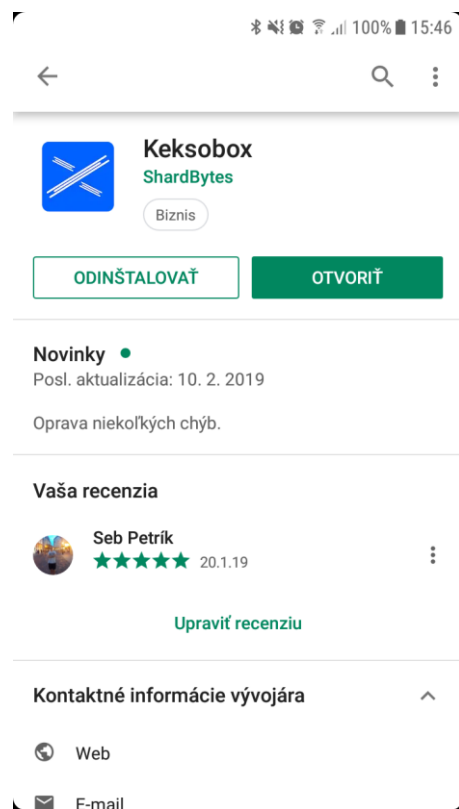
Obr. 10 CartFragment



Obr. 11 OrderShowcaseFragment



Obr. 12 DeliveryFragment



Obr. 13 Záznam v Google Play