

# **STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST**

**Obor č. 12: Tvorba učebních pomůcek, didaktická technologie**



M A T K A P

**Jiří Fryjauf, Jan Jindrák, Adam Lisner**  
**Praha**

**Praha, 2022**

# **STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST**

**Obor č. 12: Tvorba učebních pomůcek, didaktická technologie**

**MatKap – mobilní aplikace pro výuky literatury**

**MatKap – literature learning mobile application**

**Autoři:** Jiří Fryjauf, Jan Jindrák, Adam Lisner

**Škola:** Gymnázium, Praha 6, Arabská 14

**Kraj:** Praha

**Konzultant:** Ing. Daniel Kahoun, Mgr. Ivana Vondráčková

## **Prohlášení**

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Praze dne 2. března 2022 .....

## **Poděkování**

Rádi bychom chtěli na tomto místě poděkovali našemu vedoucímu projektu, panu Ing. Danielu Kahounovi, se kterým jsme naši práci konzultovali a který nám dal mnohé věcné rady a tipy. Dále bychom také rádi poděkovali paní profesorce Mgr. Ivaně Vondráčkové za konzultace ohledně literárních děl.

## **Anotace**

Cíl této práce byl vytvoření mobilní aplikace pro výuku literatury. Uživatel si může v aplikaci procvičit své znalosti literatury a také získat mnoho nových znalostí z této oblasti. Aplikace je určena pro mobilní zařízení s operačním systémem Android. Naprogramována je v jazyce Java s použitím mnohých moderních technologií.

## **Klíčová slova**

Android, literatura, mobilní aplikace, Java, databáze

## **Abstract**

The aim of this thesis was to create a mobile application which would help with the education of literature. The user can revise their knowledge of literature or acquire new knowledge from this area. The application is designed for the Android operating system. It is programmed in Java while using many modern technologies.

## **Keywords**

Android, literature, mobile application, Java, database

## **Annotation**

Der Ziel unserer SOČ-Arbeit war eine Applikation für Literaturlernen zuherstellen. Der Benutzer kann seine Kenntnisse der Literatur in der Applikation praktizieren und auch viele neue Erkenntnisse aus diesem Bereich erhalten. Die Applikation ist für mobile Geräte entwickelt, die das Android-Betriebssystem haben. Es wurde in Java mit vielen modernen Technologien programmiert.

## **Schlüsselwörter**

Android, die Literatur, das Mobileapp, Java, die Datenbank

## OBSAH

1. Úvod.....	8
2. Použité technologie .....	9
3. Frontend .....	10
3.1. Activity Main a Register .....	10
3.2. MenuActivity .....	11
3.3. ProfileActivity .....	12
3.4. QuestActivity .....	12
3.5. CheatSheet.....	12
3.6. Kvíz.....	13
3.6.1. QuizSettings .....	13
3.6.2. QuestionActivity .....	13
3.6.3. QuizSummary.....	13
3.7. Grafické možnosti aplikace .....	14
4. Backend.....	15
4.1. SQLite databáze .....	15
4.1.1. Movement.....	15
4.1.2. Genre .....	15
4.1.3. Druh.....	15
4.1.4. Author .....	16
4.1.5. Book .....	16
4.2. Příprava kvízu .....	16
4.2.1. Využití třídy Cursor .....	16
4.2.2. QuestionType .....	16
4.2.3. QuestionList .....	17
4.2.4. Question .....	18
4.2.5. copyDatabaseToStorage.....	19
4.2.6. isDatabaseCorrect.....	19
4.2.7. openOrCreateGeneralDatabase .....	19
4.3. Třída Firestore .....	19
4.3.1. QuestType .....	20
4.3.2. Quest .....	20
4.3.3. User .....	20
4.3.4. getToday's metody .....	21
4.3.5. addFirebaseUser .....	21

4.3.6.	setFirebaseUser .....	22
4.3.7.	updateUser.....	23
4.4.	Třída RegisterActivity.....	24
4.4.1.	firebaseAuthWithGoogleAccount .....	24
4.4.2.	signIn.....	24
4.4.3.	checkUser .....	24
4.5.	Třída ProfileActivity .....	25
4.5.1.	signOut .....	25
4.5.2.	deleteAccount.....	25
4.6.	Třída SQL.....	25
4.6.1.	Třída Author.....	25
4.6.2.	Třída Book.....	25
4.6.3.	Třída Movement.....	25
4.7.	Třída QuizSummary .....	26
4.8.	Storage.....	26
4.9.	Třída Utils .....	26
4.9.1.	inputStreamEquals .....	26
4.9.2.	getNNumbersFromDate .....	27
4.9.3.	getNRandomNumbers .....	27
4.9.4.	UTFComparator .....	27
5.	Závěr .....	28

# 1. ÚVOD

Tato práce se zabývá naprogramováním mobilní aplikace MatKap (zkrácenina „*Maturita v kapse*“), jejíž účelem je pomoci studentům maturitních ročníků se lépe připravit na ústní zkoušku z českého jazyka a literatury. Cílem je udělat plně funkční mobilní aplikaci pro operační systém Android, která bude uživatelsky přívětivá a intuitivní. Dalším cílem bylo vytvořit testovací systém, ve kterém si uživatel může ověřit a prohloubit své znalosti, které bude potřebovat u zkoušky. Kromě testovací fáze si může uživatel také pročíst krátké zápisky o jednotlivých autorech, knihách a literárních epochách.

Práce byla rozdělena do tří hlavních směrů: front-end, databáze a back-end.



## 2. POUŽITÉ TECHNOLOGIE

V této práci je zkombinováno mnoho moderních a užitečných technologií. Zdrojový kód aplikace je napsán v programovacím jazyce *Java*. Program byl napsán ve vývojovém prostředí *Android Studio* ve verzi *Artic Fox / 2020.3.1*. Toto prostředí je založené na jádře *IntelliJ* od české společnosti *JetBrains*. Data aplikace jsou uložena v *SQLite* databázi. *SQLite* je odnoží *SQL*, která umožňuje lokálně uložit databázi. Pro práci s databází byl použit program *DataGrip* ve verzi 2021.3.4. Tento program je velmi užitečný pro práci s více tabulkami najednou a umožňuje práci s mnoha databázovými systémy včetně právě *SQLite*. Stojí za zmínku, že i *DataGrip* je českým produktem od společnosti *JetBrains*.

Pro autentikaci, správu účtů, sdílené cloudové úložiště některých souborů byl použit open-source nástroj *Firebase* od *Googlu*. *Firebase* je výborný nástroj díky své dostupnosti a funkčnosti. Udržuje zabezpečení účtů, nabízí mnoho možností komunikace s uživateli (např. no-reply emaily). *FireBase* je využíván mnohými velkými projekty světově známých firem, což je také známkou kvality. Užitečnou vlastností *Firebase* je také skvělé propojení s *Android Studi*em a dobře popsaná dokumentace, která obsahuje názorné video tutoriály.

Během vývoje bylo použito také mnoho grafických a designových nástrojů. Velmi nápomocný byl program *Figma*. Jedná se o vektorový grafický editor a nástroj pro navrhování webových, desktopových a mobilních aplikací, který používají mnozí profesionální *UI/UX* designéři. Další grafické programy použité v práci jsou programy z dílny *Adobe*, *Adobe Photoshop* a *Adobe Illustrator*. Tyto programy jsou určeny pro práci s rastrovou a vektorovou grafikou.

Pro optimalizaci a usnadnění týmové práce byl použit *Git* a *GitHub*. *Git* je nástroj pro správu verzí zdrojového kódu. Byl vyvinut počátkem tohoto století a jedná se o open-source systém. *GitHub* je naopak služba, která nabízí zdarma webhosting pro open-source projekty a jejich verzování pomocí *Gitu*. Zdrojový kód je uložen v repozitáři, který je uložen na *GitHubu*.

### 3. FRONTEND

Front-endová část aplikace je taková část aplikace, kterou vidí a kterou používá uživatel. Proto je potřeba mu věnovat čas a úsilí, stejně jako back-endu. Front-end v Android aplikacích je psán pomocí značkovacího jazyka XML (*Extensible Markup Language*) a tzv. aktivit. O jednotlivých aktivitách a jejich struktuře se dozvíte v následujících kapitolách.

#### 3.1. Activity Main a Register

*Activity Main* je první obrazovkou, se kterou se uživatel setká. Na obrazovce se nachází *Button* s nápisem „Pokračovat“, který naslouchá uživatelským příkazům pomocí *Listeneru*. V případě, že uživatel má již provedenou registraci, tak po stisknutí tlačítka proběhne přihlášení. V druhém případě, kdy ještě nedošlo k jakémukoliv přihlášení do aplikace skrze uživatelské zařízení, je uživatel přesměrován do další aktivity, do *Activity Register*. Přesměrování mezi aktivitami probíhá pomocí *Intentů*, které ve svých argumentech přebírají informace o tom, z jaké aktivity dojde ke přesměrování a do které se má přesměrovat. Ukázkou této aktivity si můžete prohlédnout na obrázku č. 1.

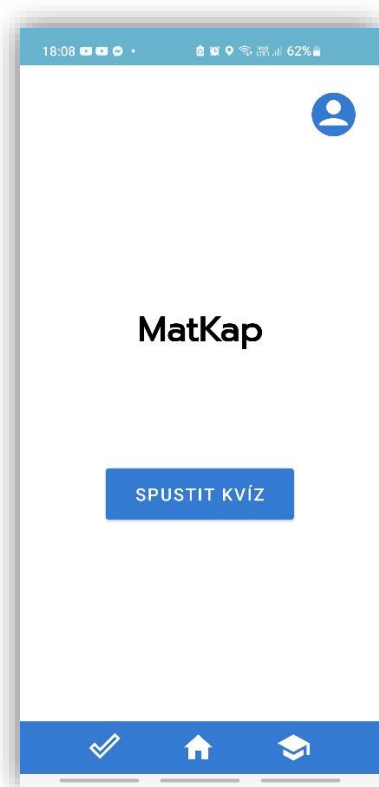


Obrázek č.1: ActivityMain

V aktivitě Register se uživatel registruje do aplikace pomocí svého *Google* účtu. Po stisknutí tlačítka s nápisem „Registrovat“ se vytvoří uživatelský účet. Více o registraci a přihlašování se dozvíte v kapitole *Třída RegisterActivity*.

## 3.2. MenuActivity

Aktivita Menu je základním kamenem orientace uživatele v aplikaci, jelikož právě menu aplikace umožňuje uživateli se dostat do všech aktivit, které aplikace obsahuje. Vpravo nahoře je *ImageButton*, jehož stisknutím se uživatel dostane na svůj profil. Uprostřed stránky je tlačítko, kterým se zapíná kvíz. Ve spodku okna se nachází ovládací panel. Tímto panelem se uživatel pohybuje mezi jednotlivými aktivitami a je stejný ve většině aktivit. Tento panel je reprezentován *FlexBoxLayoutem*, který má velkou řadu výhod. Jak již z názvu napovídá *FlexBoxLayout* je velmi podobný tomu, který většina webových inženýrů zná z CSS. A tato androidová implementace je mu velmi podobná, co se týče jeho použití. Obsahuje většinu funkcí, které jeho webový příbuzný, jako například: *alignItems*, *flexDirection*, *alignContent* nebo *justifyContent*. Poslední z jmenovaných příkladů funkcí byl použit pro tento panel. Díky *justifyContent* se všechny prvky daného layoutu rovnoměrně uspořádají, což je dobré pro správné vykreslení na různých telefonech, jelikož rozlišení není na mobilních zařízeních sjednoceno. *FlexBoxLayout* ale není v základním Android Studio balíčku, tudíž jej člověk musí implementovat v souboru *build.gradle*. Jak vypadá výsledný produkt zjistíte na obrázku č. 2.



Obrázek č.2: MenuActivity

### 3.3. ProfileActivity

V profilové aktivitě nalezne uživatel informace o svém účtu. V horní části obrazovky nalezne uživatel svůj profilový obrázek, který se mění s jeho úrovní, a jeho doposud dosaženou úroveň. Pod touto dvojicí ukazatelů se nachází *ProgressBar*, jenž slouží jako ukazatel počtu XP do nové úrovně. Dodatečnými informacemi jsou jméno a email uživatele. Všechny dosud jmenované položky jsou silně spjaté s back-endovou částí programu o níž se dozvíte v kapitole xXx. Kromě osobních informací obsahuje *ProfileActivity* tlačítka odhlásit a smazat účet. Po stisknutí druhého jmenovaného se objeví *AlertDialogBox* (vyskakovací okno), který se uživatele zeptá, jestli opravdu chce zrušit svůj účet. Ukázku této obrazovky si můžete prohlédnout na obrázku č. 4542112. Posledním prvkem je ovládací panel na spodku obrazovky, kterým se přepíná mezi jednotlivými aktivitami aplikace.

### 3.4. QuestActivity

V této aktivitě nalezne uživatel úkoly na daný den. Na každý den jsou vždy 3 úkoly. U každého úkolu je vždy jeho zadání, ukazatel pokroku v jeho dokončení, počet XP za dokončení a ukazatel, který indikuje dokončení úkolu. Tyto ukazatele jsou zabaleny v *LinearLayout*, který dále obsahuje *ConstraintLayout* se zadáním a ukazatelem dokončení, *ProgressBar* a množství XP, které získá. Na obrázku č. si můžete prohlédnout, jak tato aktivita vypadá. O back-endovém principu úkolů se dozvíte v kapitole *Quest*.

### 3.5. CheatSheet

Aktivita CheatSheet slouží k učení jednotlivých pojmů. Po otevření této aktivity objeví uživatel seznam jednotlivých autorů, knih a směrů, kterými může scrollovat a rozkliknout je pro další informace. Seznam je reprezentován *RecyclerView*, které je nutné implementovat v souboru *build.gradle*. *RecyclerView* umožňuje vývojářům dynamicky přidávat jednotlivé položky na obrazovku z *ArrayList*. Jedna z jeho velkých výhod je také dobrá optimalizace, jelikož pracuje pouze s daty potřebnými pro vykreslení, nikoliv s celým seznamem, což je velmi nápomocné zejména při práci s velkými daty. Pro nastavení jednotlivých záznamů v *RecyclerView* je nutné vytvořit třídu dědící z *RecyclerView.Adapter<ViewHolder>*. Po implementování všech potřebných metod a vytvoření třídy dědící z *RecyclerView.ViewHolder* se může nastavit vše potřebné. Ve funkci *onCreateViewHolder* je vytvořen nový View, který bere mnoho vlastností z XML souboru *book\_card.xml*. V metodě *onBindViewHolder* jsou kartám nastaveny jednotlivé texty a informace. Informace, které jednotlivé info-karty obsahují jsou brány z databáze, o jejímž fungování se dozvíte více v kapitole SQL.

Pro každou variantu seznamu (autoři, knihy, směry) je vytvořen vlastní Adapter i třída, která obsahuje přístupové metody k jednotlivým proměnným. Tyto soubory najdete v package *cheat\_sheet*.

## 3.6. Kvíz

Jednou z hlavních částí této práce je literární kvíz. V následujících kapitolách se dozvíte více informací o jednotlivých částech tohoto kvízu.

### 3.6.1. QuizSettings

Po stisknutí tlačítka “Spustit kvíz” v menu je uživatel přesměrován do aktivity nastavení kvízu. V této aktivitě si nastaví, jaké otázky bude kvíz obsahovat a jak se bude kvíz chovat. Pro výběr pouze některých okruhů otázek (okruhem je myšleno dané umělecké období / směr) je nutné stisknout čtyřúhelník, který je vytvořen pomocí *TextView*. Po této události se uživateli objeví *AlertDialogBox*. Toto vyskakovací okno obsahuje všechny směry z databáze a kliknutím na ně si může uživatel zvolit jednotlivá témata kvízu. Je také možné si nastavit variantu tohoto kvízu. První varianta je kvíz, při kterém je hned po zodpovězení zobrazena správná / špatná odpověď. Druhá varianta je test, při této variantě se uživatel dozví svůj výkon až po testu. Tyto varianty se nastavují pomocí *CheckBoxu*. Po nastavení lze kvíz zapnout stisknutím tlačítka.

### 3.6.2. QuestionActivity

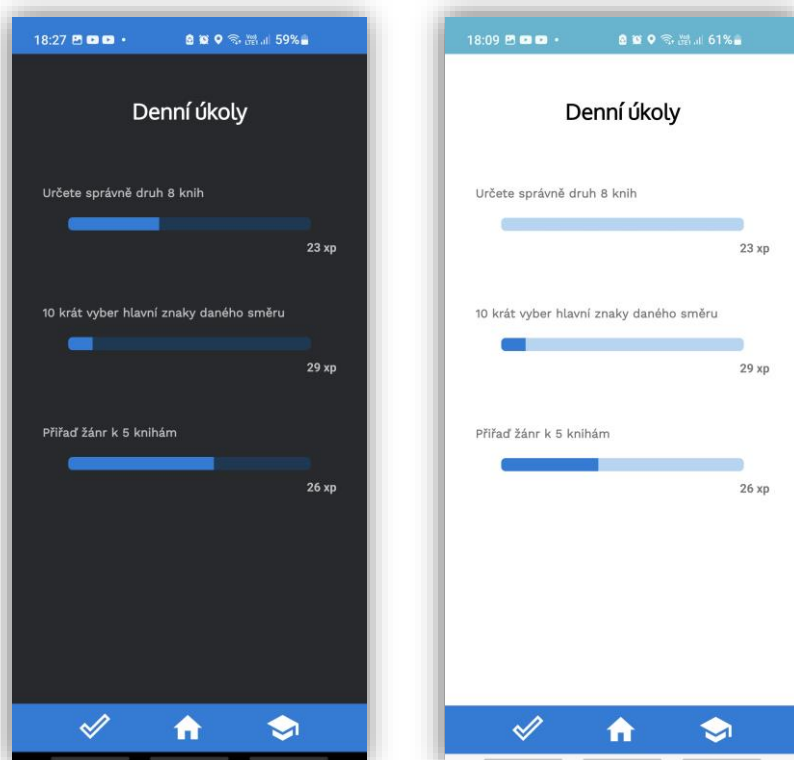
Tato aktivita je už samotný kvíz. Kvíz má defaultně deset otázek, pouze v případě, že se v daných okruzích nenalézá dostatek otázek, je tento počet snížen. Toto je však velmi okrajová situace, ke které by nemělo docházet. Ovládání kvízu je velmi jednoduché. V horní části obrazovky se nachází text otázky. Pod zadáním se nachází čtyři možné odpovědi, kde vždy právě jedna je správná. Políčka na odpověď jsou jednotlivé *RadioButtony*, které dohromady tvoří jednu skupinu neboli *RadioGroup*. Po zvolení odpovědi ji může uživatel potvrdit pomocí tlačítka v pravém dolním rohu. Když není zvolena odpověď, není možné odpovědět. U varianty kvízu (viz předchozí kapitola) je také po odpovězení na 1,5 vteřiny zobrazena správná a špatná odpověď. Zároveň je v kvízu uživateli zobrazeno aktuální skóre. V průběhu testu se ukládají jednotlivé odpovědi (reprezentované objektem *AnsweredQuestion* dědicí z *Sql.Question*) do *ArrayListu<AnsweredQuestion>*. Po dokončení kvízu je uživatel přesměrován do souhrnu kvízu (*QuizSummary*).

### 3.6.3. QuizSummary

Po dokončení kvízu se zobrazí vyhodnocení kvízu. Uživateli se zobrazí skóre v procentech, zpráva o tom, jak si vedl. Dále se v případě více špatných odpovědí na jedno téma zobrazí zpráva: “*Měl by sis procvičit směr:*” + směry k procvičení. Také si může uživatel zpětně přečíst otázky, na které odpovídal, společně s jejich správnou odpovědí, případně jeho špatnou odpověď. Zpětné čtení otázek je vytvořeno pomocí *RecyclerView*, stejně jako *CheatSheet*. Více o jeho fungování se dozvíte v kapitole .

### 3.7. Grafické možnosti aplikace

V dnešní době je absolutní nutností, aby aplikace nabízela možnost světlého i tmavého režimu. Tmavý režim je velmi užitečný pro práci na telefonu ve večerních hodinách, jelikož ze světlého režimu mnozí dostávají bolest očí a pociťují až bolesti hlavy. Proto i aplikace MatKap disponuje možností tmavého režimu, který se nastavuje podle uživatelského výchozího nastavení ze systému telefonu. Tmavý režim si ponechává typickou modrou barvu, která provází uživatele celou aplikací. Pro porovnání najdete na obrázku č. 3 snímky obrazovky z QuestActivity ve dvou rozdílných režimech



Obrázek č.3: Porovnání režimů

## 4. BACKEND

Kromě Frontendu přichází část obsahující veškerou logiku aplikace, což je backend společně s databází. Tato kapitola je rozsáhlá, takže je rozdělena do několika podkapitol, které obsahují informace a popis jednotlivých tříd a jejich důležitých funkcích.

### 4.1. SQLite databáze

Jako dotazovací jazyk byl zvolen *SQL* s nářečím *SQLite*, jelikož toto nářečí nevyžaduje online propojení z databází. Databázi lze totiž uložit jako lokální soubor. Databáze obsahuje pět tabulek, které jsou navzájem propojené pomocí cizích klíčů. U některých záznamů chybí nějaké hodnoty, jelikož buď nejsou dohledatelné nebo jednoznačné, což způsobí to, že pouze nemohou být obsaženy v určitých otázkách. Každá z těchto tabulek používá primární klíč *id*.

#### 4.1.1. Movement

Tato tabulka má čtyři sloupce; *id* (primární klíč), *name* (název – unikátní klíč), *sign* (hlavní znaky) a *century* (časové zařazení daného období).

Některé záznamy neobsahují hlavní znaky, protože znaky nejsou jednoznačné.

#### 4.1.2. Genre

Tato tabulka obsahuje dva sloupce; *id* (primární klíč) a *name* (název – unikátní klíč).

Tato tabulka vyžadovala zvláštní opatrnost, jelikož na některých zdrojích jsou žánry udávány často špatně nebo jsou jako žánry označovány pojmy, které žánry nejsou, např. často jsou díla označována jako báseň, která se ale dělí na několik dalších žánrů. Při přidání tohoto falešného žánru do databáze může dojít na případ, kdy jsou v kvízu dvě správné odpovědi, ale jako správná je vyhodnocena pouze jedna.

#### 4.1.3. Druh

Tato tabulka má dva sloupce; *id* (primární klíč) a *name* (název – unikátní klíč).

Jako jediná tabulka je jako jediná pojmenována česky, jelikož v angličtině je tento pojem příliš podobný slovu *genre* a mohlo by dojít k záměně.

#### 4.1.4. Author

Tato tabulka obsahuje pět sloupců; id (primární klíč), name (jméno – unikátní klíč), movement\_id (id literárního směru – cizí klíč z tabulky movement), sex (pohlaví), country (země původu).

V této tabulce je i autor jménem „Neznámý autor“, který nemá literární směr ani zemi původu a slouží zde pouze pro díla, která mají neznámého autora.

#### 4.1.5. Book

Tato tabulka obsahuje sedm sloupců; id (primární klíč), name (název – unikátní klíč), author\_id (id autora – cizí klíč z tabulky author), genre\_id (id žánru – cizí klíč z tabulky genre), druh\_id (id literárního druhu – cizí klíč z tabulky druh), movement\_id (id literárního směru – cizí klíč z tabulky movement) a year (rok vydání díla).

### 4.2. Příprava kvízu

Stěžejní část aplikace je automaticky vytvářený kvíz, který využívá lokální SQLite databázi obsahující informace, ze kterých se vytváří osm typů otázek. Jelikož je učivo českého jazyka velice rozsáhlé, bylo potřeba mnoho optimalizace a efektivnosti, jak bude popsáno v následujících odstavcích.

Tato třída je nejrozsáhlejší z celého projektu, jelikož obsahuje nejvíce funkcí, které pomocí databáze zároveň vytváří otázky do kvízu, kontrolují databázi apod.

#### 4.2.1. Využití třídy Cursor

Pro operaci s SQLite databází se v aplikaci využívá objekt třídy *Cursor*, která reprezentuje výslednou tabulku po provedení dotazu pomocí klíčového slova SELECT v jazyce SQL.

#### 4.2.2. QuestionType

Aby byl jednoduchý a z vnějšku neměnný počet typů otázek, je ve třídě *Sql* výčet *QuestionType*, který obsahuje vzory pro jednotlivé otázky. Za vzor se počítá to, že každý typ uchovává text otázky, který se později pouze doplní o potřebné informace, které jsou získány z databáze, k čemuž slouží další informace, kterou výčet uchovává; název sloupců, kde se hledá text odpovědi a text, který je třeba doplnit do textu otázek.

Dále třída obsahuje metody pro snadnější doplnění textů do otázek. Jelikož jsou texty otázek vždy stejné, bylo dostatečným řešením do vzorového textu vložit unikátní označení úseků řetězce, která jsou později vyměněna za potřebná slova pomocí metody *String.replaceAll*, která je obsažena v samotném jádru Javy.



Jak je možno vidět na obrázku č.4, každý typ otázky má svou vlastní metodu pro doplnění do textu, je tomu tak, protože různé typy otázek mají různé počty argumentů.

```
AUTHOR_BOOK("Kterou knihu napsal<sex> <author>?", "author_name", "book_name"),
BOOK_AUTHOR("Kdo napsal \<book>\"?", "book_name", "author_name");

static String completeBAText(String book) {
    return BOOK_AUTHOR.questionText.replaceAll("<book>", book);
}

static String completeABText(sex, author) {
    res = AUTHOR_BOOK.questionText;
    res = res.replace("<author>", author);
    return res.replace("<sex>", sex);
}

//completeBAText("Máj") returns "Kdo napsal "Máj"?"
```

Obrázek č.4: QuestionType

Názvy těchto typů mohou být na první pohled nesrozumitelné, ale mají jednoduchý vzorec; např. *AUTHOR\_BOOK* znamená, že v textu otázky se objeví některý autor a jako odpovědi budou jednotlivé knihy.

### 4.2.3. QuestionList

Třída, jak název napovídá, obsahuje list otázek, které bylo za daného nastavení možno vytvořit. Obsahuje pouze tři použitelné metody; *getPossibleQuestionsCount*, která pouze vrátí délku listu, *add(Question)*, která přidá otázku do listu, a *getNQuestions*, která vrátí daný počet náhodných otázek.

Samotná třída nemá tolik funkcí, důležitější je vnější metoda *getQuestionList*, která z databáze vytvoří tři *Cursor* objekty; *author*, *book* a *movement*, které obsahují všechny potřebné informace. Následně projde všechny řádky těchto *Cursorů* a pomocí metody *Question.createQuestion* vytvoří jednotlivé otázky.

Každý kurzor slouží k vytvoření nějakých typů otázek, kterých je dohromady osm.

Aby si uživatel mohl kvíz přizpůsobit, vedle této třídy je ještě výčet *FilterType*, který měl původně obsahovat více filtrů, ale v zachování snadné ovladatelnosti a efektivity je tu pouze filtr, který třídí podle jednotlivých literárních směrů.

Dále je zde třída *Filter*, která při předání listu objektů *String* metodě *formatFilters* vytvoří doplněk do SQL dotazů, které jsou provedeny v již zmíněné metodě *getQuestionList*. Jelikož uživatel nikde nemůže narušit text těchto dotazů, databáze je chráněna vůči tzv. SQL injection.

#### 4.2.4. Question

Tato třída reprezentuje otázku, která se může objevit ve kvízu. Každá otázka má nějaký text a odpovědi, z nichž je právě jedna správnou, takže toto jsou informace, které má každý objekt této třídy.

Třída má privátní konstruktor, takže nelze vytvořit vlastní otázku a musí se použít metoda *createQuestion*, která přístup ke konstruktoru má. Tato metoda je použita pouze v již zmíněné metodě *getQuestionList*, kde se vyvolá právě jednou pro každý záznam v databázi.

Algoritmus pro vytvoření otázky, který lze vidět na obrázku č.5, je až na dva typy (*BOOK\_DRUH* a *AUTHOR\_BOOK*) stejný; jelikož jsou potřeba čtyři odpovědi, první (správná) odpověď se vezme ze stejného řádku jako údaj do otázky, zbytek odpovědí se pokud možno náhodně vybere ze zbytku řádků. Samozřejmě lze narazit na případ, kdy by algoritmus narazil na druhou správnou odpověď, proto se před přidáním každé odpovědi odpověď zkontroluje pomocí metody *isValidAnswer*, která otestuje všechny případy, které by mohli znemožnit správnou selekci odpovědi.

```
List<Answer> answers = new ArrayList<>();
List<String> questionStrings = new ArrayList<>();
answers.add(new Answer(cursor.getString(column)));
questionStrings.add(cursor.getString(cursor.getColumnIndex(qCol)));
int[] indexes = Utils.getNRandomNumbers(cursor.getCount(),
cursor.getCount());
loop:
    for (int i = 0; i < 3; i++) {
        for (int j : indexes) {
            cursor.moveToPosition(j);
            if ((isValidAnswer())){
                answers.add(new Answer(cursor.getString(column)));
                continue loop;
            }
        }
        cursor.moveToPosition(position);
    }
    return null;
}
```

Obrázek č.5: Hledání možných odpovědí

Nakonec se po vybrání odpovědi odpovědi zamíchají pomocí metody *Collections.shuffle*.

U typu *BOOK\_DRUH* je tento algoritmus zbytečně zdouhavý, jelikož existují pouze 4 literární druhy. Díky tomuto faktu stačí vždy vytvořit stejné odpovědi a u té správné akorát zaznamenat, že je správná, čehož se dosáhne jednoduchou sérií podmínek. Odpovědi se nakonec opět stejným způsobem zamíchají.

Pro typ *AUTHOR\_BOOK* vzniká relace 1:N z pohledutabulky *author*, na což by stačil první algoritmus, ale autor může mít dvě různá pohlaví, což se objeví na textu otázky, takže se musí pro získání odpovědi využít tabulka *book*, ale do otázky je potřeba doplnit údaj o

pohlaví autora. Nakonec se tedy využije stejný algoritmus jak ov prvním případě, ale pro text otázky je třeba využít metodu *getAuthorPositionFromBook*, která získá index řádku se správným autorem v tabulce *author*, takže lze získat pohlaví autora a pokračovat stejně jako v prvním případě.

Když nelze vytvořit danou otázku (např. není dostatečný počet odpovědí), metoda vrátí *null* a otázka se nepřidá do výsledného listu.

#### 4.2.5. **copyDatabaseToStorage**

Soubor s databází je uložený v adresáři samotné aplikace, která je zkompileovaná, takže s těmito soubory nelze pracovat jako s *File* soubory, ale pouze jako s *OutputStream* objekty. Ale pro práci s SQLite databází je třeba *File* objekt, takže se pomocí této metody vytvoří soubor v úložišti telefonu, do kterého se vloží data z *OutputStream* objektu.

Tímto se tedy databáze zkopíruje do úložiště telefonu, což sice působí nebezpečně, ale opak je pravdou, jelikož původní *OutputStream* působí jako jakási záloha databáze, která se případně znovu zkopíruje při jakémkoliv rozdílu dat v těchto dvou souborech.

#### 4.2.6. **isDatabaseCorrect**

Tato metoda slouží ke kontrole databáze. Pokud soubor z databází v telefonu není nebo se kopie jakkoliv liší od původního souboru. Hned se znovu vyvolá metoda *copyDatabaseToStorage*.

Ke kontrole dat se využije metoda *Utils.inputStreamEquals*.

Tato metoda slouží jako jakási pojistka při jakémkoliv narušení databáze, což může být například tzv. *SQL injection*, ruční změna dat uživatelem nebo třeba aktualizace aplikace, kvůli které je třeba znovu zkopírovat databázi.

#### 4.2.7. **openOrCreateGeneralDatabase**

Tato metoda zajistí, že databáze bude otevřena a přístupná pouze ve třídě *Sql* dosazením objektu *SQLiteDatabase* do privátní statické proměnné *generalDatabase*. Vyvolá předchozí metody *AppCompatActivity.openOrCreateDatabase*, *isDatabaseCorrect* a popřípadě i *copyDatabaseToStorage*.

### 4.3. **Třída Firestore**

Tato třída zprostředkovává ukládání uživatelských dat do *NoSQL Google Firestore* databáze, která k ukládání využívá kolekce JSON dokumentů.

Kromě samotného propojení tu dochází i k uložení veškeré práce s uživatelskými daty do RAM.

### 4.3.1. QuestType

Třída Firestore obsahuje výčet *QuestType*, který obsahuje typy úkolů stejnojmenné objektům výčtu *QuestionType*. *QuestType* obsahuje statickou metodu, která z daného *QuestType* objektu získá analogický *QuestionType* objekt. Dále každý typ úkolu obsahuje vzor textu zadání úkolu, do kterého se pomocí metody *getText* vždy jen doplní číslo, které udává maximální počet akcí potřebných pro splnění úkolů.

Znamená to tedy, že např. zadání pro typ *AUTHOR\_BOOK* („Přiřaď správně 7 knih k autorům“) vyžaduje daný počet správně zodpovězených *AUTHOR\_BOOK* otázek.

Ná obrázku č.6 lze vidět, že výčet má narozdíl od výčtu *QuestionType* společnou *getText* metodu, jelikož počet argumentů a požadovaná hodnota by byly pro všechny metody stejné

```
AUTHOR_BOOK("Přiřaď správně <amount> knih k autorům");

private String getText(int max) {
    return text.replaceAll("<amount>", ""+max);
}

//AUTHOR_BOOK.getText(5) returns "Přiřaď správně 5
knih k autorům"
```

Obrázek č.6: QuestType

### 4.3.2. Quest

Třída *Quest* reprezentuje jednotlivé úkoly, které obsahují jednak text zadání úkolu, ale také procento dokončení úkolu, logickou hodnotu, zda je úkol dokončen, a číselnou hodnotu, udávající zkušenostní odměnu za dokončení úkolu.

Dále jsou zde metody *stepForward*, která progres úkolu posune o jedna, a *collect*, která za případu, že úkol ještě není vyzvednutý, úkol vyzvedne, tak, že logickou proměnnou *isComplete* nastaví na *true* a vyvolá metodu *User.addXp*, které předá správný počet zkušeností.

### 4.3.3. User

Tato třída obsahuje veškeré informace o přihlášeném uživateli, což je pole s úkoly na daný den, *level* a zkušenosti.

Kromě proměnných obsahuje třída *User* i dvě důležité metody; *addXp* a *questEventHandler*.

Metoda `addXp` přičte daný počet zkušeností a v případě, že je výsledný počet zkušeností vyšší než požadovaný počet pro další level, level se zvýší o jeden a zkušenosti se nastaví na počet, který zbyde po odečtení potřebných zkušeností pro tento level.

Metoda `questEventHandler` je vyvolána při každém správném zodpovězení otázky s parametrem objektu třídy `Sql.Question`, podle kterého zjistí, zda je typ této otázky potřebný pro některý z úkolů, pokud ano, tak vyvolá metodu `Quest.stepForward` pro daný úkol

#### 4.3.4. `getTodays` metody

Pod tuto kategorii spadají metody `getTodaysQuestMaxes`, která vrátí maximální počty vyžadovaných akcí pro jednotlivé úkoly, `getTodaysQuestRewards`, která vrátí zkušenostní odměny pro jednotlivé úkoly, a `getTodaysQuests`, která vrací `QuestType` objekty pro jednotlivé úkoly.

Všechny tyto metody využívají metodu `Utils.getNNumbersFromDate`, která vrací pole čísel, které se liší podle data daného dne, velikostí pole a maximální hodnoty.

#### 4.3.5. `addFirebaseUser`

Tato metoda uživateli vytvoří JSON dokument ve Firestore databázi, ve kterém jsou uloženy jednotlivé názvy `QuestType` objektů, procentuální stavy jednotlivých úkolů, level a počet zkušeností. Docíleno je toho tak, že stejně, jako je vidět na obrázku č.7, vytvoří `Map<String, Object>` objekt, do kterého se vloží začátečnická data, jelikož se vytváří nový dokument. Mapa se poté nahraje do Firebase databáze v podobě JSON dokumentu, jehož název je hash kód uživatelského emailu.

Jako název dokumentu se nastaví hash kód emailu uživatele.

Metoda je vyvolána ve dvou případech; uživatel je nový, uživateli byl smazán dokument v databázi.

```
public static void addFirebaseUser() {  
    QuestType[] numbers = getTodaysQuests();  
    Map<String, Object> user = new HashMap<>();  
    user.put("u1_type", numbers[0]);  
    user.put("u1_stat", 0.0);  
    user.put("u2_type", numbers[1]);  
    user.put("u2_stat", 0.0);  
    user.put("u3_type", numbers[2]);  
    user.put("u3_stat", 0.0);  
    user.put("lv1", 1);  
    user.put("xp", 0);  
  
    db.collection("users")  
        .document(document_name)  
        .set(user);  
    setFirebaseUser(firebaseUser, false);  
}
```

Obrázek č.7: addFirebaseUser()

#### 4.3.6. setFirebaseUser

Tato metoda z databáze přečte všechny údaje o uživateli a pomocí nich vytvoří objekt třídy User, čímž se informace zpřístupní samotné aplikaci.

Všichni uživatelé mají pro daný den stejné úkoly, což znamená, že když úkoly uživatele jsou jiné než získané pomocí metody getTodaysQuests, úkoly uživatele jsou z jiného dne a jak lze vidět na obrázku č.8, nastaví se nové úkoly, kterým se restartuje progres nastavením proměnné *Quest.percentage* na hodnotu 0.0 u všech tří úkolů.

Pokud metoda v databázi nenajde příslušný dokument uživatele, vyvolá metodu *addFirebaseUser*.

Tato metoda je vyvolána ve dvou případech; po dokončení metody *addFirebaseUser*, po přihlášení uživatele.

```

public void onComplete(@NonNull Task<DocumentSnapshot> task) {
    Map<String, Object> user = task.getResult().getData();
    if (user == null) {
        addFirebaseUser();
        return;
    }
    int i = 0;
    Quest[] quests = new Quest[]{new Quest(), new Quest(), new Quest()};
    int lvl = 0;
    int xp = 0;
    QuestType[] types = getTodaysQuests();
    boolean shouldBeRestarted = false;
    int[] xps = getTodaysQuestRewards();
    double[] maxes = getTodaysQuestMaxes();
    for (Map.Entry<String, Object> entry : user.entrySet()) {
        String row = entry.getKey();
        switch(row) {
            case "lvl":
                lvl = ((Long)entry.getValue()).intValue();
                break;
            case "u1_type":
                quests[0].setQuestionType(types[0]);
                break;
            ...
            case "u3_stat":
                quests[2].setPercentage((Double)entry.getValue());
                break;
            case "xp":
                xp = ((Long)entry.getValue()).intValue();
                break;
        }
    }
    Firestore.user = new User(quests, lvl, xp);
}

```

Obrázek č.8: setFirebaseUser()

#### 4.3.7. updateUser

Podle obrázku č.9 lze říct, že tato metoda má podobnou funkci jako metoda addFireBaseUser s tím rozdílem, že nevytvoří nový dokument se začátečnickými údaji, ale aktualizuje již existující dokument, do kterého zapíše údaje ze třídy User.

Tato metoda je vyvolána po jakékoli manipulaci s daty ve třídě User.

```

public static void updateUser() {
    Map<String, Object> user = new HashMap<>();
    user.put("u1_type", Firestore.user.quests[0].questType);
    user.put("u1_stat", Firestore.user.quests[0].percentage);
    user.put("u2_type", Firestore.user.quests[1].questType);
    user.put("u2_stat", Firestore.user.quests[1].percentage);
    user.put("u3_type", Firestore.user.quests[2].questType);
    user.put("u3_stat", Firestore.user.quests[2].percentage);
    user.put("lvl", Firestore.user.lvl);
    user.put("xp", Firestore.user.xp);


    db.collection("users")
        .document(document_name)
        .set(user);
}

```

Obrázek č.9: updateUser()

## 4.4. Třída RegisterActivity

Tato třída se stará o registraci a přihlášení uživatele ke Google účtu, pomocí služby Google Firebase. V nástroji Firebase lze v sekci Authentication vidět přehled účtů uživatelů. Je zde vidět jejich e-mail, služba, přes kterou se přihlásili (v našem případě Google), datum vytvoření účtu, datum posledního přihlášení a identifikátor uživatele.

Identifier	Providers	Created ↓	Signed In	User UID
test@gmail.com		Jan 1, 2022	Feb 20, 2022	1XyzXYz2XyZXyZ3XyzXYZ4

Obrázek č.10: Seznam uživatelů ve Firebase

### 4.4.1. firebaseAuthWithGoogleAccount

Tato metoda se stará o přihlášení uživatele přes Google účet. Po úspěšné autentifikaci pomocí Firebase metoda uživatele přihlásí. Pokud ještě uživatel neexistuje, vytvoří se mu účet na Firebase.

### 4.4.2. signIn

Metoda signIn spustí Google přihlašovací intent, pomocí které si uživatel může vybrat, z jakého účtu se chce přihlásit.

### 4.4.3. checkUser

Pokud je uživatel již přihlášen (tj. neodhlášen z Google účtu), přeskočí aplikace registraci nebo autentifikaci účtu a přejde rovnou k další aktivitě.



## 4.5. Třída ProfileActivity

### 4.5.1. signOut

Tato Metoda zajistí odhlášení uživatele z Google účtu, pomocí Firebase API. Je vyvolána po stisknutí tlačítka odhlásit.

### 4.5.2. deleteAccount

Tato metoda po vyvolání získá instanci momentálního uživatele a pomocí API cloudové služby Firestore ze služby Google Firebase smaže jeho uživatelský profil.

## 4.6. Třída SQL

Třída SQL se stará o zpracovávání SQL příkazů a propojením s databází.

### 4.6.1. Třída Author

Třída Author se stará o získávání údajů o autorovi z SQL databáze. Obsahuje metody *getAuthorList*, *getBooks*, *getName*, *getMovement* a *getCountry*, které pomocí SQL dotazů získají informace.

Metoda *getAuthorList* vrátí list autorů, ze kterého v třídě CheatSheet poté dostaneme pomocí ostatních getter metod důležité údaje o autorovi. Mezi údaje patří jeho jméno, knihy, které napsal, směr, ke kterému se hlásí a stát, ze kterého pochází.

### 4.6.2. Třída Book

Třída Book zpracovává údaje o knihách z SQL databáze. Obsahuje metody *getBookList*, *getAuthor*, *getName*, *getGenre*, *getDruh*, *getMovement* a *getYear*, které pomocí SQL dotazů obstarají získání informací.

Metoda *getBookList* vrátí list knih, ze kterého v třídě CheatSheet získáváme pomocí getter metod podstatné informace o knihách. Z listu se získává jméno knihy a autora, směr, ke kterému se autor hlásí, žánr knihy, druh knihy a rok, kdy bylo dílo vydáno.

### 4.6.3. Třída Movement

Třída Movement slouží ke zpracovávání údajů o knihách z SQL databáze. Obsahuje metody *getMovementList*, *getAuthors*, *getName*, *getSign* a *getCentury*, které pomocí SQL dotazů obstarají informace.

Metoda `getMovementList` vrátí list směrů, ze kterého se v třídě `CheatSheet` získávají použitím getter metod podstatné informace o směrech. Z listu se získává jméno hnutí, autorů, které se k němu hlásí, jeho znaky a období jeho popularity.

## 4.7. Třída `QuizSummary`

Ze třídy `QuizSummary` se po dokončení kvízu získá procentuální úspěšnost testu, a směry, ve kterých se nejvíce chybovalo. Třída se stará i o hlášky, které uživatele podle úspěšnosti testu informují o tom, jestli by si měl látku procvičit a nabídne mu látku k zopakování.

## 4.8. Storage

Tato třída doplňuje `ProfileActivity` o profilový obrázek, který je vybrán podle levelu uživatele. Je zde využit nástroj `Firebase cloud storage`, kde jsou uloženy jednotlivé předvytvořené profilové obrázky.

Jednotlivé obrázky jsou reprezentovány ve výčtu `ProfilePicture`, který obsahuje názvy osmi obrázků, které jsou uloženy na sdíleném cloudovém úložišti.

Metody `getProfilePicturePath` a `getPicturePath` dohromady stáhnout danou fotku do úložiště telefonu při každém přihlášení do aplikace pro případ, že by v cloudu fotky byly změněny.

Třída obsahuje metodu `downloadPics`, která stáhne všechny obrázky. Dále je ve třídě logická proměnná `downloaded`, které se nastaví hodnota `true` ve chvíli, kdy se dokončí stahování obrázku, který je právě potřebný, avšak i dále se stahují i ostatní.

## 4.9. Třída `Utils`

Toto je doplňková třída, která obsahuje jednoduché funkce, jež jsou opakovaně užity v různých třídách a nejsou přímo spojené se žádnou z tříd.

### 4.9.1. `InputStreamEquals`

Metoda `InputStreamEquals` porovnává data, které lze získat ze dvou `InputStream` objektů pomocí cyklu, který postupně čte data a porovnává je. Jakmile najde neshodu, ihned vrátí hodnotu `false` a zavře oba `InputStream` objekty. Pokud nenajde rozdíl, neboli oba objekty vrátí hodnotu `-1`, metoda vrátí hodnotu `true` a opět zavře oba `InputStream` objekty.

### 4.9.2. getNNumbersFromDate

Metoda *getNNumbersFromDate* vrací pro stejné argumenty a datum vždy stejné pole čísel. Jak lze vidět na obrázku č.11, metoda vytvoří hash kód z datumu daného dne a předá ho metodě *fillNumbers* společně s počátečním indexem výsledného pole.

Metoda *fillNumbers* je rekurzivní a vyvolá se sama znovu pro další index nebo pro stejný v případě, že vygenerované číslo není unikátní. Nový seed vytvoří pomocí vygenerovaného čísla.

```
public static int[] getNNumbersFromDate(int bound, int count){
    int[] numbers = new int[count];
    int seed = DateTimeFormatter
        .ofPattern("yyyy/MM/dd")
        .format(LocalDate.now()).hashCode()*1234;
    Arrays.fill(numbers, Integer.MIN_VALUE);
    fillNumbers(seed, numbers, 0, bound);
    return numbers;
}

private static void fillNumbers(int seed, int[] numbers, int index, int bound) {
    if (index >= numbers.length) return;
    Random r = new Random(seed);
    int n = r.nextInt(bound);
    if (!contains(numbers, n)) {
        numbers[index] = n;
        fillNumbers(r.nextInt(), numbers, index+1, bound);
        return;
    }
    fillNumbers(r.nextInt(), numbers, index, bound);
}
```

Obrázek č.11: getNNumbersFromDate()

### 4.9.3. getNRandomNumbers

Metoda *getNRandomNumbers* vrací pole o zadané délce plné unikátních hodnot v zadaném rozsahu. Metoda tohoto docílí cyklem, který pro každý index generuje náhodná čísla dokud nevygeneruje hodnotu, která v poli ještě není. Metoda vrací null v případě, že takové pole nelze vygenerovat, což nastane, když je zadaný rozsah menší než požadovaná délka pole.

### 4.9.4. UTFComparator

Tato třída společně s *interfacem NameAcquire* tvoří nástroj pro abecední řazení, které zahrnuje i číslice a českou abecedu. Třída implementuje z *interfacu Comparator*, které se používá na definování vlastního způsobu řazení. Rozdíl je v tom, že při volání např. metody *Collections.sort* se už nemusí definovat nový *Comparator*, ale stačí vytvořit nový *UTFComparator*, jehož konstruktor vyžaduje objekt *interfacu NameAcquire* jako argument.

*NameAcquire* obsahuje jednu metodu *getName*, kterou je třeba definovat tak, aby vracela řetězec, podle kterého se bude řadit.

## 5. ZÁVĚR

Cílem tohoto projektu bylo naprogramovat intuitivní aplikaci pro pomoc výuky literatury v českém jazyce, která využije různých online i offline technologií pro možnost ukládání uživatelských dat a přihlašování a doufáme, že jsme tento cíl splnili.

Bylo použito vývojářské prostředí Android Studio a DataGrip, které jsou obě od šeskové společnosti JetBrains, jazyky Java pro backend, XML pro frontend, SQLite pro offline databázi a JSON pro online databázi. Projekt jsme ukládali pomocí služby GitHub využívající technologie Git.

Dokumentace byla rozdělena na dvě hlavní části; frontend a backend. V první části je vysvětleno vytvoření front-endu a struktura jednotlivých oken aplikace. V jednotlivých kapitolách se dozví čtenář o tom, jak dané části vizuálu aplikaci vznikly a z čeho se skládají. Toto vše je doplněno ilustrativními obrázky pro lepší představu čtenáře.

Druhá část byla samotná rozdělena do několika dalších kapitol.

První z těchto kapitol byla SQLite databáze, kde byla vysvětlena struktura všech tabulek. Další kapitola byla Příprava kvízu, kde je popsána hlavní funkce aplikace, což je vytvoření kvízu podle filtrů zadanými uživatelem. Poté byla kapitola popisující třídu Firestore, která operuje s uživatelskými daty a ukládá je do Firebase Firestore databáze v podobě kolekcí JSON dokumentů. Předposlední kapitola je Storage, která popisuje, jak se stahují a ukládají profilové obrázky do úložiště telefonu uživatele. Nakonec byla kapitola Utils, která osvětluje vedlejší funkce, které nebylo vhodné zařadit do jiné třídy.

Během vytváření této aplikace jsme se naučili několik nových dovedností z oblasti programování. Mezi ně patří práce s databází, designování, návrh dobrého UI/UX a propojení více jazyků.

Ačkoliv projekt prošel četnými změnami, nakonec vývoj došel do konce díky skvělé spolupráci, kterou jsme si v týmu sjednali.

Dále by se aplikace dala doplnit o mnoho funkcí, např. herní měna, vlastní profilový obrázek nebo více typů otázek a úkolů. Kromě těchto jednoduchých návrhů by se obor této aplikace dal rozšířit z na více maturitních předmětů.

## OBRÁZKY

Obrázek č.1: ActivityMain.....	10
Obrázek č.2: MenuActivity .....	11
Obrázek č.3: Porovnání režimů .....	14
Obrázek č.4: QuestionType.....	17
Obrázek č.5: Hledání možných odpovědí .....	18
Obrázek č.6: QuestType.....	20
Obrázek č.7: addFirebaseUser() .....	22
Obrázek č.8: setFirebaseUser() .....	23
Obrázek č.9: updateUser().....	24
Obrázek č.10: Seznam uživatelů ve Firebase .....	24
Obrázek č.11: getNNumbersFromDate() .....	27

## POUŽITÉ ZDROJE

Android Developers. (14. únor 2022). *Developer Android - Create dynamic lists with RecyclerView*.

Načteno z Developers Android Guide:

<https://developer.android.com/guide/topics/ui/layout/recyclerview>

Android WorldCup. (10. květen 2020). *Expandable RecyclerView in AndroidStudio*. Načteno z

YouTube:

[https://www.youtube.com/watch?v=pGi02uJre4M&t=941s&ab\\_channel=AndroidWorldCup](https://www.youtube.com/watch?v=pGi02uJre4M&t=941s&ab_channel=AndroidWorldCup)

Google. (20. květen 2021). *Google GitHub Repository*. Načteno z GitHub:

<https://github.com/google/flexbox-layout>

Google Android Developers. (24. leden 2022). *Android Developers*. Načteno z Android Developers

Guide - Dialog: <https://developer.android.com/guide/topics/ui/dialogs>

Google Android Developers. (2022). *Developers Android*. Načteno z Android Developers

Documentation: <https://developer.android.com/docs>

Google Firebase. (11. 1. 2022). *Authenticate Using Google Sign-In on Android*. Načteno z Firebase:

<https://firebase.google.com/docs/auth/android/google-signin>

Google Firebase. (13. 1. 2022). *Cloud Firestore*. Načteno z Firebase:

<https://firebase.google.com/docs/firestore/>

Google Firebase. (5. 2. 2022). *Get started with Cloud Storage on Android*. Načteno z Firebase:

<https://firebase.google.com/docs/storage/android>

Google Material. (2022). *Material.io*. Načteno z material.io: material.io

Podhajská, M. V. (19. červenec 2017). *Seznam autorů literárních děl, literárních žánrů, směrů a*

*hnutí k DT*. Načteno z Bean.cz: [http://www.bean.cz/wp-](http://www.bean.cz/wp-content/uploads/2019/09/Seznam-autor%C5%AF-liter%C3%A1rn%C3%ADch-d%C4%9Bl-liter%C3%A1rn%C3%ADch-%C5%BE%C3%A1nr%C5%AF-sm%C4%9Br%C5%AF-a-hnut%C3%AD-k-DT.pdf)

[content/uploads/2019/09/Seznam-autor%C5%AF-liter%C3%A1rn%C3%ADch-d%C4%9Bl-liter%C3%A1rn%C3%ADch-%C5%BE%C3%A1nr%C5%AF-sm%C4%9Br%C5%AF-a-hnut%C3%AD-k-DT.pdf](http://www.bean.cz/wp-content/uploads/2019/09/Seznam-autor%C5%AF-liter%C3%A1rn%C3%ADch-d%C4%9Bl-liter%C3%A1rn%C3%ADch-%C5%BE%C3%A1nr%C5%AF-sm%C4%9Br%C5%AF-a-hnut%C3%AD-k-DT.pdf)

Uživatelé Dribbble.com. (2022). *Dribbble*. Načteno z Dribbble: dribbble.com

Uživatelé Rozbor-díla.cz. (nedatováno). *Rozbor-díla.cz*. Načteno z Rozbor díla: [https://rozbor-](https://rozbor-dila.cz/)

[dila.cz/](https://rozbor-dila.cz/)