VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÁ APLIKÁCIA NA PRECHÁDZANIE A EDITÁCIU JEDNODUCHÝCH TÝŽDENNÝCH ROZVRHOV

MOBILE APP FOR EDITING AND VIEWING SIMPLE WEEKLY SCHEDULES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ROMANA DŽUBAROVÁ**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Studentka: **Džubarová Romana**
Program: Informační technologie
Název: **Mobilní aplikace pro editaci a prohlížení jednoduchých týdenních rozvrhů**
Mobile App for Editing and Viewing Simple Weekly Schedules
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se s problematikou vývoje uživatelských rozhraní a aplikací pro Android.
2. Vyhledejte a analyzujte existující aplikace pro editaci a prohlížení týdenních rozvrhů.
3. Navrhněte vlastní aplikaci pro editaci a prohlížení týdenního rozvrhu.
4. Implementujte navrženou aplikaci.
5. Testujte aplikaci na uživatelích a iterativně ji vylepšujte.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN-13: 978-0321965516
- Android Developers: https://developer.android.com/index.html
- Susan M. Weinschenk: 100 věcí, které by měl každý designér vědět o lidech, Computer Press, Brno 2012
- Bill Phillips, Chris Stewart, Brian Hardy, Kristin Marsicano: Android Programming (2nd Edition): The Big Nerd Ranch Guide, Big Nerd Ranch 2015
- Stephen Samuel, Stefan Bocutiu: Programming Kotlin, Packt Publishing 2017

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Cieľom tejto práce je vytvoriť mobilnú aplikáciu, ktorá bude zaznamenávať pravidelné týždenné aktivity používateľa. Používateľom sa však nerozumie len študent vysokej alebo strednej školy, ale aplikácia má osloviť všetkých ľudí, ktorí si potrebujú zaevidovať pravidelné aktivity v týždni. Používateľ si môže vytvoriť aktivity, ktoré sa následne vložia do tabuľky. Tieto aktivity si môže meniť alebo mazať. Aplikácia umožňuje vytváranie úloh a skúšok. Ku skúškam je možné nastaviť upozornenie, ktoré užívateľ obdrží v čase skúšky, alebo si tento čas môže v Nastaveniach zmeniť. Používateľ si tiež môže v Nastaveniach zmeniť aj to, ako bude jeho tabuľka vyzerať. Môže zmeniť, v ktoré dni si bude chcieť zaznamenávať aktivity alebo začiatočný a koncový čas. V aplikácii tiež možno nastaviť tmavý režim a ušetriť tak výdrž batérie. Bakalárska práca popisuje návrh, implementáciu, testovanie, a tiež budúce zámery s mojou mobilnou aplikáciou.

Abstract

The aim of this work is to create a mobile application that will write down the regular weekly activities of the user. Users are not only university or high school students, but the application is intended to appeal to all people who need to register for regular activities during the week. Users can create activities which are inserted into the table. These activities can be changed or deleted. The application allows you to create a test task. It is possible to set a notification for the exams, which the user will receive at the time of the exam, or he can change this time in the Settings. The user can also change what his table will look like in Settings. He can change the days on which he wants to record activities or the start and end time of activity. The application can also be set to dark mode to save battery life. The bachelor thesis describes the design, implementation, testing, and also future intentions for my mobile application.

Kľúčové slová

mobilná aplikácia, Android Studio, Android, rozvrh hodín, databáza, SQLite, Kotlin, upozornenie, tabuľka, aktivita, úloha, skúška

Keywords

mobile aplication, Android Studio, Android, timetable, database, SQLite, Kotlin, notification, table, activity, task, exam

Citácia

DŽUBAROVÁ, Romana. *Mobilná aplikácia na prechádzanie a editáciu jednoduchých týždenných rozvrhov*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Mobilná aplikácia na prechádzanie a editáciu jednoduchých týždenných rozvrhov

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracovala samostatne pod vedením pána prof. Ing. Adam Herout, Pd.D.. Uviedla som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpala.

.....
Romana Džubarová
10. mája 2021

Poďakovanie

V prvom rade by som chcela poďakovať garantovi môjho projektu pánovi prof. Adamovi Heroutovi, ktorý mi pomohol pri výbere témy bakalárskej práce, a tiež ma zahŕňal užitočnými poznámkami k nej. Ďalšie poďakovanie patrí všetkým mojim blízkym, ktorí mi boli oporou pri tvorbe bakalárskej práce a jej testovaní.

Obsah

Kapitola 1

Úvod

Mobilné aplikácie v súčasnej dobe uľahčujú život nie jednému človeku. Existuje celé množstvo takýchto aplikácií. Od jednoduchších, ako sú slovníky, po zložitejšie, ako sú aplikácie pre banky, rôzne sociálne siete a hry. Ďalšou skupinou aplikácií, ktoré sú využívané najmä študentami vysokých škôl, alebo žiakmi stredných a základných škôl, sú rozvrhové aplikácie. Dnešný trh však neponúka aplikácie pre zaznamenávanie týždenných aktivít, ktoré by boli jednoduché a svojím obsahom by zároveň nezahltili používateľa. Cieľom tejto práce bolo vytvoriť práve takúto aplikáciu.

S rozvrhovými aplikáciami sa stretáva takmer každý človek, navštevujúci nejakú vzdelávaciu inštitúciu, dennodenne. Sama si rada ukladám informácie o predmetoch do prehľadných tabuliek a mám ich vždy rada pri sebe. Najviac ma vie nahnevať, ak ma aplikácia zahĺti veľkým množstvom informácií, reklamami alebo ak sa odo mňa vyžadujú veci, ktoré ani nepotrebujem. Preto je dôležité, aby obsahovali podstatné informácie a hlavne, aby vytváranie nových udalostí bolo efektívne a nie príliš zdĺhavé.

Na začiatku vývoja aplikácie je dobré overiť si od potencionálnych budúcich používateľov, čo by od aplikácie, kde si budú zaznamenávať pravidelné týždenné aktivity, očakávali. S jasnou víziou, ako má aplikácia vyzerať, ide jej implementácia od ruky.

V súčasnej fáze aplikácie je možné zaznamenávať týždenné aktivity do tabuľky. Tieto aktivity sa dajú farebne odlíšiť, meniť, ale aj mazať. Taktiež si používateľ môže nastaviť skúšku, ktorá mu v čase udalosti zašle upozornenie na túto skúšku, alebo vytvoriť zoznam úloh, ktoré je potrebné urobiť.

Aplikácia *Marker* pre operačný systém *Android* je písaná v programovacom jazyku *Kotlin*. Postup tvorby jej vzniku je popísaný v nasledujúcich kapitolách, a tiež v tomto videu[1]. V druhej kapitole je podrobnejšie opísané, čo ma viedlo k implementácii takejto aplikácie, a tiež porovnanie môjho projektu s inými dostupnými aplikáciami. Ďalšia kapitola ukazuje technológie použité pri implementácii tejto aplikácie, ako sú napríklad programovací jazyk *Kotlin* či databáza *SQLite* a ich popis. Štvrtá kapitola ukazuje návrh aplikácie od prvotného návrhu, až po konečný. Taktiež popisuje vytvorenie loga a názvu aplikácie. V predposlednej kapitole je opísaný postup pri implementácii jednotlivých častí, z ktorých následne vzišla aplikácia v dnešnej podobe. V poslednej časti sú popísané rôzne fázy testovania, ktorým si táto aplikácia prešla. Od interného testovania, medzi niekoľkými mojimi blízkymi ľuďmi, až po ostrú verziu zverejnenú na *Google Play*[2].

[1] https://www.youtube.com/watch?v=v3iJy02gn38
[2] https://play.google.com/store/apps/details?id=com.timetableRApp

Kapitola 2

Analýza požiadaviek na moju aplikáciu a jej použitie

Táto kapitola pozostáva z troch častí. V prvej je opísaný dôvod vývoja aplikácie *Marker*, pre akú platformu bola aplikácia vyvíjaná, v akom programovacom jazyku bola písaná a od akej úrovne je dostupná. Prečo som si vybrala práve aplikáciu pre ukladanie pravidelných týždenných aktivít a čo ma viedlo k implementácii takejto aplikácie. Ďalšia časť ukazuje podobné aplikácie a môj postoj k nim. Čo mi v nich chýbalo, a naopak, čo sa mi páčilo a bolo to malou inšpiráciou aj pri vývoji môjho projektu. Posledná časť popisuje spôsob použitia tejto aplikácie. K čomu slúžia jednotlivé karty, a čo sa od používateľa očakáva pri vytváraní aktivít, úloh alebo skúšok.

2.1 Podnet k vývoju rozvrhovej aplikácie

V čase študentských čias, kde je záznam pravidelných týždenných aktivít rutinnou záležitosťou začiatku každého semestra, som sa rozhodla priniesť do sveta aplikácií jednu v mojom prevedení.

Rozvrhová aplikácia je pre študenta veľmi podstatná. Je priam neoddeliteľnou súčasťou každého študentského dňa. Ako študent som sa stretávala s viacerými, ktoré mi nie úplne vyhovovali. Buď boli príliš komplikované a preplnené rôznymi nástrojmi, alebo pri zadávaní predmetu bolo potrebné zdĺhavé vyťukávanie informácií. Ďalšou nevýhodou bolo, že väčšinou sa jednalo o aplikácie určené pre študentov, nie pre ľudí, ktorí nespadajú do tejto kategórie a chcú si uchovať týždenné aktivity, ktorými nie sú predmety v škole, ale nejaké krúžky a voľnočasové aktivity.

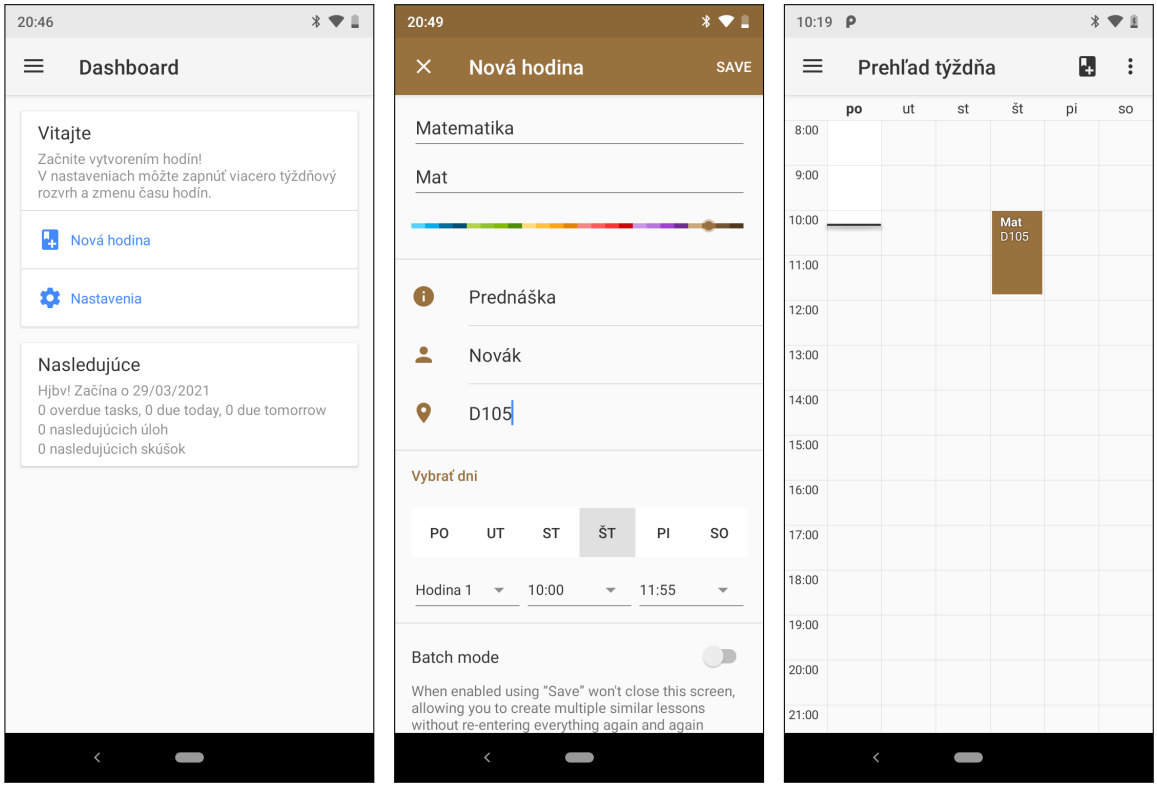
Preto som sa rozhodla vyvinúť aplikáciu *Marker*, ktorá je dostupná na platforme *Android*, implementovaná bola v programovacom jazyku *Kotlin*. Úrovne rozhrania API (z anglického „*Application Programming Interface*“) sú od 21 a vyššie. Implementácia na takýchto starých *Androidoch* bola celkom výzva, pretože som si musela všímať, od ktorej verzie sú používané prostriedky dostupné a ako popisujem v kapitole 6.2, boli chvíle, keď mi to skomplikovalo implementáciu.

2.2 Príklady podobných aplikácií

V tejto časti budú popísané tri aplikácie na zaznamenávanie týždenných aktivít, ktoré sú dostupné, buď na *Google Play*[1] alebo v *AppStore*[2].

2.2.1 Timetable

Aplikácia *Timetable*[3], vyvíjaná Gabrielom Ittnerom, patrí medzi používanejšie aplikácie. Na *Google Play* má viac ako milión stiahnutí. Aplikáciu ohodnotilo skoro štyridsaťdva tisíc používateľov a s celkovým hodnotením štyroch hviezdičiek obstála na dôstojnom mieste. Vytvorenie udalostí v tejto aplikácii je zobrazené na obrázku 2.1. Po prvom spustení, sa mi namiesto prehľadnej tabuľky, ktorá by symbolizovala to, k čomu táto aplikácia slúži, zobrazil takzvaný „*Dashboard*“, s úvodnými informáciami pre používateľa, čo ma trošku zmiatlo a myslela som si, že som urobila chybu v sťahovaní aplikácie. Viac v tabuľke 2.1.



Obr. 2.1: **Aplikácia *Timetable*.** Obrázky znázorňujú jednú z troch aplikácií, ktoré sú podobné aplikácii *Marker*. Na prvom obrázku je znázornená prvá karta aplikácie hneď po spustení. Druhý obrázok ukazuje kartu, na ktorej sa pridávajú informácie o predmete. Na poslednom je zobrazený konkrétny predmet v tabuľke.

[1] https://play.google.com/store
[2] https://www.apple.com/cz/app-store/
[3] https://play.google.com/store/apps/details?id=com.gabrielittner.timetable

Po dôslednom prejdení aplikácie môžem skonštatovať toto

+	-
Jednoduchosť používania.	Zbytočná úvodná stránka, dá sa to však zmeniť v nastaveniach.
Pri zakladaní nového predmetu, umožňuje uložiť všetky potrebné informácie.	Niektoré funkcie sa zdajú duplicitné (pri úlohe a skúške).
Pekný design.	
Založenie predmetu možné kliknutím na políčko v tabuľke.	
Rýchly prístup ku karte pre založenie nového predmetu.	
Možnosť tmavého módu.	

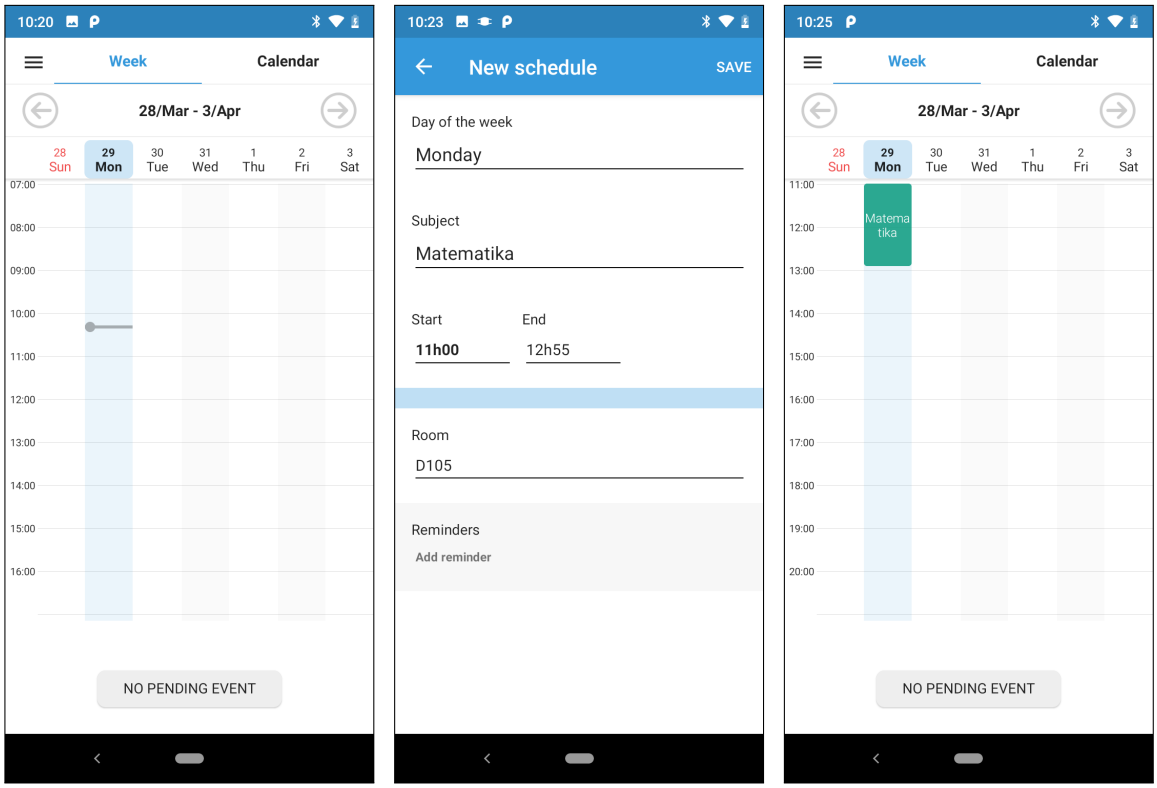
Tabuľka 2.1: **Plusy a mínusy aplikácie *Timetable*.** Tabuľka obsahuje moje zhodnotenie kladných a záporných stránok aplikácie *Timetable*.

Celkovo hodnotím aplikáciu ako veľmi dobrú. Spomedzi týchto troch aplikácií ako najlepšiu. Až na drobné chyby, ktoré je možno v nastaveniach zmeniť, nevidím nič, čo by mi pri jej používaní prekážalo. Táto aplikácia mi bola malou inšpiráciou pri vývoji môjho projektu. Obzvlášť pri implementácii karty pre založenie novej aktivity.

2.2.2 Student Calendar – Remember tasks ToDo & Timetable

Ďalšou z pozorovaných aplikácií je aplikácia *Student Calendar*[4]. Táto aplikácia je najlepšie hodnotenou rozvrhovou aplikáciou na *Google Play*. Dostala až 4,7 hviezdičiek z viac ako dvadsaťjeden tisíc hodnotení. Celkovo má, podobne ako predchádzajúca aplikácia, niečo cez milión stiahnutí. Ako vidieť na obrázku 2.2, hneď po spustení aplikácie sa zobrazí pekná a prehľadná tabuľka, do ktorej sa môžu vkladať predmety. Na rozdiel od predchádzajúcej aplikácie to hodnotím ako veľké plus. Tabuľka 2.2 však viac popisuje moje postrehy k tejto aplikácii.

[4] https://play.google.com/store/apps/details?id=com.claudivan.agendadoestudanteplus



Obr. 2.2: **Aplikácia *Student Calendar*.** Tieto obrázky ukazujú ďalšiu aplikáciu, ktorá je podobná aplikácii *Marker*. Prvý obrázok zobrazuje aplikáciu hneď po spustení. Ďalší obrázok ukazuje kartu na pridanie predmetu a posledný zobrazuje daný predmet v tabuľke.

Aplikáciu *Student Calendar* môžem zhodnotiť takto

+	-
Po spustení zobrazuje hneď tabuľku.	Nemožnosť založenia predmetu z úvodnej strany. Používateľ sa musí viac krát prekliknúť, kým sa dostane na kartu zakladania predmetu.
Možnosť týždňového prehľadu a mesačného prehľadu.	Chýba tmavý mód.
Založené úlohy sa ukladajú, a teda ich môžeme použiť ako šablóny pri zakladaní podobných úloh.	Obsahuje reklamy.
Veľmi pekný design.	Úvodná karta obsahuje zbytočné informácie.

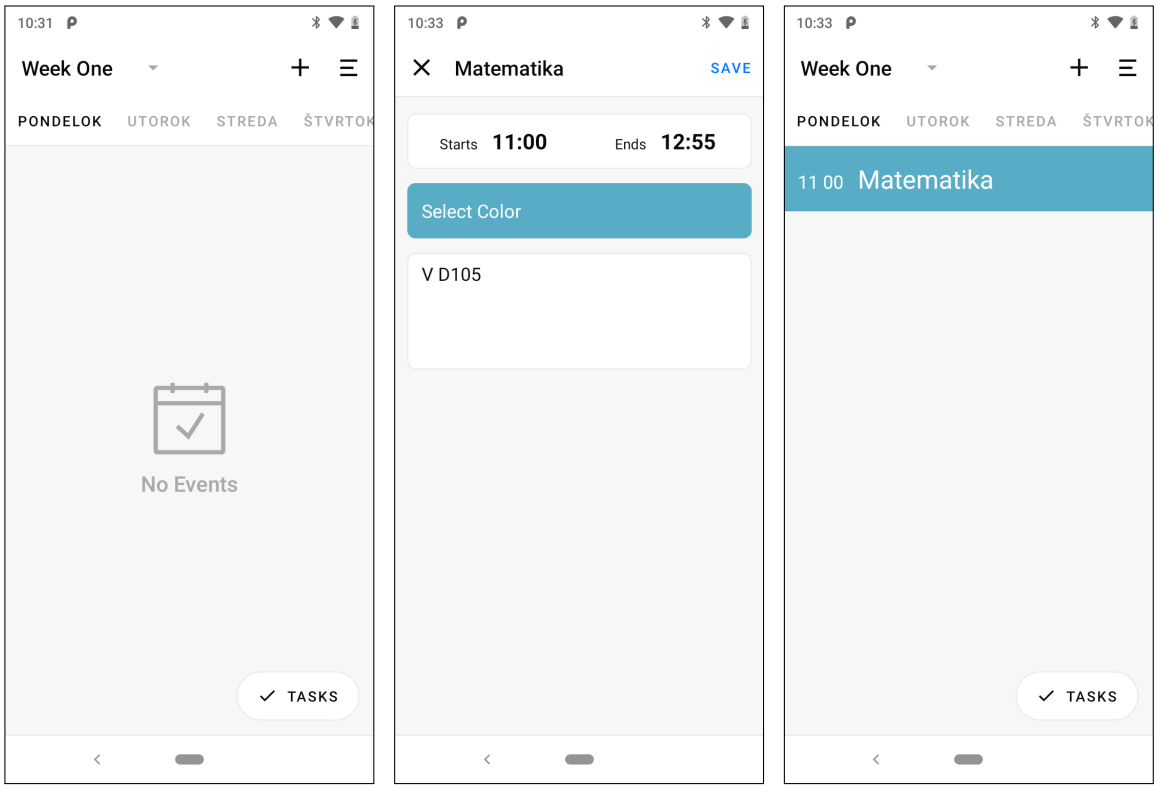
Tabuľka 2.2: **Plusy a mínusy aplikácie *Student Calendar*.** Tabuľka popisuje kladné a záporné stránky, ktoré som našla na tejto aplikácii.

Od aplikácie *Student Calendar* som čakala trošku viac, súdiac podľa hodnotenia, ktoré mala na *Obchod Play*. Najviac mi vadí komplikovanosť zakladania nových predmetov. Pri viacerých aktivitách by to mohlo znamenať stráviť, pri vytváraní nových udalostí, viac času.

Tiež negatívne na mňa vplývajú reklamy, ktoré sa objavujú pri používaní tejto aplikácie. Ďalším negatívom, alebo skôr zbytočnosťou, je popis dňa začiatku a dňa konca aktuálneho týždňa, a tiež popis, že neprichádza žiadna udalosť. Namiesto toho by som zväčšila úvodnú tabuľku na celú plochu.

2.2.3 Class Timetable – Schedule App

Poslednou aplikáciou je *Class Timetable*[5]. Toto je jediná, mne známa rozvrhová aplikácia, ktorá je dostupná ako v Obchod Play, tak aj v App Store. Jej hodnotenie je trošku nižšie, celkovo 3,9 hviezdičiek. To však môže byť spôsobené aj nižšou mierou hodnotenia aplikácie, ktoré nepresahuje ani osem tisíc. Aplikácia *Class Timetable* síce vo svojom názve má rozvrh hodín, no po prvotnom spustení (viz. obrázok 2.3) mi to takto vôbec nepripadá. Chýba mi tu nejaká tabuľka, ktorá by reprezentovala klasický rozvrh hodín, aký poznáme zo škôl. Ďalšie zhrnutie tejto aplikácie je popísané v tabuľke 2.3.



Obr. 2.3: **Aplikácia *Class Timetable*.** Obrázky popisujúce poslednú aplikáciu, ktorá je podobná aplikácii *Marker*. Na prvom obrázku je zobrazená aplikácia po spustení, kde vidieť jej prázdny obsah bez tabuľky. Druhý obrázok ukazuje zakladanie aktivity v tejto aplikácii, ktoré je dosť strohé a posledný obrázok zobrazuje danú aktivitu v aplikácii.

[5] https://play.google.com/store/apps/details?id=com.icemediacreative.timetable

Zhrnutie aplikácie *Class Timetable*

+	-
Rovnaká na platforme Android ako aj pre platformu iOS.	Chýba tabuľka, reprezentujúca rozvrh hodín.
Jednoduchosť používania.	Nemožnosť tmavého módu.
	Chýba zaznamenávanie si viac informácii o vytváranom predmete a nie len názov, čas a farbu.

Tabuľka 2.3: **Plusy a mínusy aplikácie *Class Timetable.*** Tabuľka zobrazuje, akú sú plusy a mínusy tejto aplikácie, z môjho pohľadu.

Od rozvrhovej aplikácie by som očakávala prehľadnú tabuľku, ktorá mi tu chýbala. Preto si myslím, že sa táto aplikácia skôr hodí pre ľudí, ktorí si radšej prechádzajú denné udalosti, ako by mali byť „obťažovaní“ celým týždenným rozvrhom. Celková aplikácia mi teda nekoreluje s jej názvom. Ďalším veľkým mínusom je možnosť ukladania informácií o predmete. Kde, podľa môjho skromného názoru, názov, čas a farba jednoducho nestačia a minimálne mi tu chýba typ aktivity a ďalšie podstatné údaje.

2.3 Základné použitie aplikácie

Pri navrhovaní rozvrhových aplikácií, musí byť kladený dôraz na jednoduchosť použitia. Používateľ pri vytváraní novej udalosti, úlohy či skúšky nemá čas zamýšľať sa nad tým, čo dané tlačidlo znamená a nemal by zbytočne dlho premýšľať k čomu slúži. Každá položka by mala byť pre neho jasná a zrozumiteľná.

Jednoduchosť bola aj jedným z mojich kritérií pre vytvorenie tejto aplikácie. Nič dôležité by nemalo byť viac ako dva kliky ďaleko [8]. Preto hneď po spustení sa zobrazí tabuľka s malým tlačidlom v tvare plus vpravo hore. Toto tlačidlo presunie používateľa na kartu, v ktorej zadáva informácie o aktivite, ktorú si chce uložiť do tabuľky. Je to najpodstatnejšia karta celej aplikácie.

Po úvodnom spustení si používateľ pravdepodobne bude chcieť hneď zaznamenať jednotlivé aktivity. Stlačením tlačidla plus ho aplikácia presunie na kartu, kde si môže zaznamenať jednotlivé informácie o danej aktivite. Jednotlivé položky tejto karty sú popísané nižšie.

Popis základných údajov pre založenie novej aktivity

- **Názov aktivity** – reprezentuje celý názov aktivity, ktorú používateľ zakladá. Jedná sa o celý názov tejto udalosti, ktorý sa síce neuloží do rozvrhu, ale po otvorení aktivity, sa mu tento názov zobrazí. Názov aktivity je povinný.
- **Skratka aktivity** – značí skratku danej aktivity. Pod touto skratkou sa používateľovi bude zobrazovať daná aktivita v tabuľke. Tento údaj je povinný.
- **Miestnosť** – nepovinný údaj, kde používateľ môže zadať číslo miestnosti, v ktorej sa daná aktivita, predmet alebo udalosť koná.

- **Vyučujúci** – tiež nepovinný údaj pre zadanie vyučujúceho danej aktivity. Môže sa jednať o profesora, učiteľa alebo lektora, ktorý vyučuje danú udalosť.
- **Typ aktivity** – používateľ má na výber zo štyroch možností typov aktivity. Prednáška, cvičenie, demonštračné cvičenie a iné. Tieto typy sú vyhovujúce skôr pre študentov, preto tam bol pridaný aj typ iné, pre používateľov mimo školských lavíc.
- **Farba** – zobrazí sa dialógové okno, v ktorom má používateľ na výber zo šestnástich farieb. Takto si môže jednotlivé udalosti farebne rozlíšiť podľa predmetov alebo typov.
- **Deň** – výber dňa z klasického sedemdňového týždňa. Škála pre dni v týždni sa môže zmeniť v *Nastaveniach*.
- **Začínajúci čas** – predstavuje začiatočný čas aktivity. Na základe vybraného dňa a tohto času sa aktivita uloží na patričné miesto v tabuľke.
- **Čas ukončenia** – predstavuje končiaci čas aktivity. Podľa tohto času sa mení výška obdĺžnika, reprezentujúceho danú aktivitu.

Aplikácia *Marker* tiež umožňuje založenie úlohy alebo skúšky, ktoré sú vo svojej podstate podobné. Rozdiel však je, že pri skúške si používateľ môže nastaviť upozornenie, ktoré mu príde v čase konania skúšky. To, koľko minút pred skúškou má upozornenie prísť, sa môže nastaviť v *Nastaveniach*.

Karty pre založenie skúšky a úlohy vyžadujú od používateľa ďalšie informácie. Vysvetlenie jednotlivých položiek sú popísané nižšie.

Popis základných údajov pre založenie novej úlohy/skúšky

- **Názov úlohy/skúšky** – predstavuje názov úlohy respektíve skúšky, ktorú si používateľ zakladá. Tento názov sa potom zobrazí na kartičke, ktorá bude symbolizovať jednotlivú položku. Názov je povinný údaj.
- **Dátum úlohy/skúšky** – značí dátum konania úlohy poprípade skúšky. Používateľovi sa po kliknutí na textové pole zobrazí dialógové okno s kalendárom, kde si môže vybrať ľubovoľný dátum. Vybraný dátum sa znázorní na kartičke symbolizujúcej úlohu/skúšku. Tento údaj je taktiež povinný.
- **Čas úlohy/skúšky** – reprezentuje čas konania úlohy alebo skúšky. Po kliknutí na textové pole sa používateľovi zobrazí dialógové okno s hodinami, z ktorého si môže vybrať čas. Zvolený čas sa taktiež objaví na kartičke. Výber času je povinný údaj.
- **Poznámka** – používateľ si po kliknutí na toto textové pole, môže zapísať ľubovoľnú poznámku k danej úlohe alebo skúške. Nie je to však povinné. Ak sa však rozhodne zapísať si tento údaj, poznámka sa mu zobrazí aj na kartičke, ktorá predstavuje danú úlohu alebo skúšku.
- **Upozornenie** – upozornenie je viditeľné len pri zakladaní skúšky. Používateľ má na výber z dvoch možností. Buď si nastaví upozornenie, ktoré mu príde v čase udalosti, alebo si toto upozornenie nenastaví.

Kapitola 3

Popis technológií pre tvorbu mobilných aplikácií

Táto kapitola bude bližšie rozvíjať rôzne technológie, ktoré sa používajú pri tvorbe mobilných aplikácií. Bude popisovať vývojové prostredie *Android Studio*, ktoré patrí medzi najlepšie voľby pri vývoji mobilných aplikácií pre *Android*, ako aj iné vývojové prostredia zamerané na vytváranie takýchto aplikácií. V tejto časti bude spomenutý princíp migrácie databáz, ako aj opis ich použitia.

Ďalej tu bude popísaný programovací jazyk *Kotlin*, ktorý som si vybrala na implementáciu mojej aplikácie, a tiež ďalšie programovacie jazyky zamerané na vývoj aplikácií. Taktiež oboznámim čitateľa s rôznymi technológiami, ktoré sa používajú pri ukladaní dát v mobilných aplikáciach.

V posledných dvoch častiach tejto kapitoly, bude vysvetlený rozdiel medzi fragmentom a aktivitou a výhody použitia *ViewPageru* v mobilných aplikáciach.

3.1 Android Studio

Android Studio[1] je oficiálne integrované vývojové prostredie (z anglického „*Integrated development environment*“, IDE) pre vývoj aplikácií na platforme *Android* založené na *IntelliJ IDEA*[2]. Tento typ vývojového prostredia je dostupný na všetkých typoch operačných systémov [9]. Jednou z mnohých výhod používania *Android Studia* je aj to, že je úplne zadarmo a umožňuje prepojenie projektov s *GitHubom*[3].

Ďalšie vývojové prostredia zamerané na vývoj mobilných aplikácií

1. **Xcode**[4] – je to integrované vývojové prostredie vytvorené spoločnosťou *Apple*, na vývoj softvérov pre operačný systém *iOS*. Programovacím jazykom pre toto prostredie je *Swift* [13].

2. **AppCode**[5] – ďalšie integrované vývojové prostredie zamerané na vývoj aplikácií pre operačný systém *iOS* a *macOS*. Okrem programovacích jazykov ako *Swift* alebo *C*,

[1] https://developer.android.com/studio
[2] https://www.jetbrains.com/idea/
[3] https://github.com/
[4] https://developer.apple.com/xcode/
[5] https://www.jetbrains.com/objc/

podporuje aj webové technológie ako *JavaScript*, *HTML* a *CSS*. Toto prostredie síce nie je zadarmo, ale dovoľuje tridsať dňovú skúšobnú verziu [2].

3. **Flutter**[6] – je to multiplatformová sada nástrojov používateľského rozhrania. Je navrhnutá tak, aby dovoľovala znovu použitie kódu v operačných systémoch ako *iOs* a *Android*. Sada *Flutter* je pomerne nová. Prvýkrát bola prezentovaná v roku 2016. Programovací jazyk, využívaný v tomto frameworku je *Dart* [3].

3.1.1 Migrácia databázovej schémy v SQLite

Pri navrhovaní databázy môže nastať situácia, kedy potrebujeme modifikovať starú schému databázy na novú. Potrebujeme buď pridať, alebo vymazať stĺpec tabuľky, zmeniť názov stĺpca tabuľky, alebo pridať, či vymazať tabuľku. V takýchto prípadoch sa používa migrácia databázovej schémy.

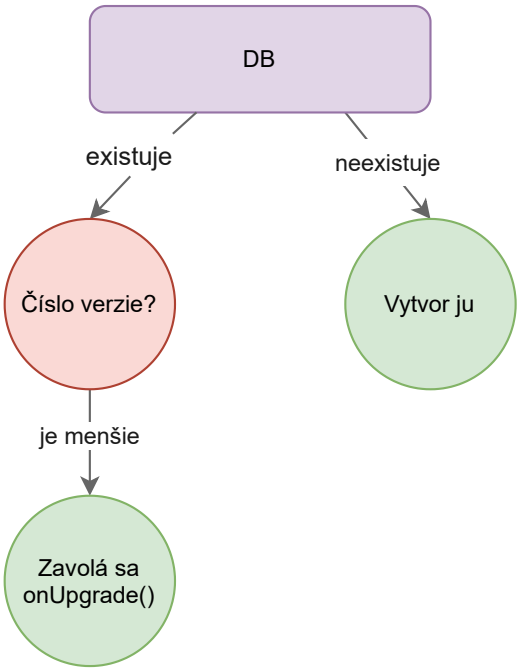
Sú to kontrolované množiny zmien, vytvorené s cieľom upraviť zloženie objektov v relačnej databáze, pričom dôležitým kritériom pri ich implementácii je zachovanie údajov pri zmene tejto databázy [12].

Cieľom softvéru na migráciu databáz je dosiahnuť, aby boli zmeny databázy opakovateľné, aby umožňovali zdieľanie údajov a testovanie dát bez ich straty. Migrácia spravidla popisuje presný súbor operácií potrebných na zmenu databázovej schémy zo súčasného stavu do nového stavu [12].

Postup pri migrácii databázovej schémy

1. Inkrementácia databázy na vyššiu verziu.
2. Zápis potrebného SQL príkazu do funkcie `onUpgrade()`.
3. Spustenie programu.

[6] https://flutter.dev/



Obr. 3.1: **Obrázok ukazujúci postup pri migrácii databázy.** Ak databáza neexistuje, vytvorí sa pomocou funkcie `onCreate()`. Ak však existuje, systém sa pozrie na jej číslo verzie. V prípade, že je toto číslo menšie ako to, čo pozná, vykonajú sa príkazy vo funkcii `onUpgrade()`.

Obrázok 3.1, znázorňuje, ako funguje funkcia `onUpgrade()`. Po spustení aplikácie sa systém pozrie na to, či daná databáza existuje alebo nie. Ak neexistuje, vytvorí ju funkciou `onCreate()`. Ak existuje, pozrie sa na číslo jej verzie, ktoré udáva, či databáza bola modifikovaná alebo nie. Ak je číslo verzie v databáze menšie ako číslo v novej databáze, spustí sa funkcia `onUpgrade()`, a jej prislúchajúce operácie.

SQLiteOpenHelper

Pre správu vytvárania databáz a databázových verzií, *Android* poskytuje pomocnú triedu *SQLitOpenHelper*. Táto pomocná trieda však sama o sebe nevytvára, či neotvára databázu. Na dosiahnutie týchto operácií potrebujeme zavolať funkcie `getWritableDatabase()` alebo `getReadableDatabase()` [17].

Ako ukazuje obrázok 3.2, na vytvorenie tejto podtriedy musíme implementovať funkcie `onCreate()` a `onUpgrade()`. Potom sa táto trieda postará o vytvorenie databázy, ak neexistuje, otvorenie databázy, ak existuje a modifikáciu databázy na novšiu verziu, ak sa zmení verzia databázy [17].

```
class DBBaseHelper(context: Context) :
    SQLiteOpenHelper(context, DATABASE_NAME, null, VERSION)  {

    companion object {
        private const val VERSION = 1
        private const val DATABASE_NAME = "SubjectDB.db"
    }

    /***
     * Create databases tables
     */
    override fun onCreate(db: SQLiteDatabase?) {
        db?.execSQL("CREATE TABLE" + DatabaseTimetable.TABLE_NAME +
            "(" + _id + " INTEGER PRIMARY KEY AUTOINCREMENT," +
                DatabaseTimetable.ID + " TEXT," +
                DatabaseTimetable.SUB_NAME + " TEXT NOT NULL," +
                DatabaseTimetable.SUBSHORT_NAME + " TEXT NOT NULL," +
                DatabaseTimetable.ROOM + " TEXT," +
                DatabaseTimetable.DAY + " TEXT NOT NULL," +
                DatabaseTimetable.BEG_TIME + " TEXT NOT NULL," +
                DatabaseTimetable.END_TIME + " TEXT NOT NULL," +
                DatabaseTimetable.TYPE + " TEXT," +
                DatabaseTimetable.COLOR + " TEXT NOT NULL," +
                DatabaseTimetable.TEACHER + " TEXT" + ")")
    }

    /***
     * Upgrade databases table
     */
    override fun onUpgrade(db: SQLiteDatabase?,
                           oldVersion: Int,
                           newVersion: Int
                           ) {
        db?.execSQL("DROP TABLE IF EXISTS" + DatabaseTimetable.TABLE_NAME)
        onCreate(db)
    }
}
```

Obr. 3.2: **Príklad vytvorenia triedy *DBBaseHelper* pomocou pomocnej triedy *SQLiteOpenHelper*.** Obrázok znázorňuje vytvorenie tabuľky s patričnými stĺpcami. V prípade prejdenia na vyššiu verziu, je vo funkcii `onUpgrade()` príkaz na odstránenie starej tabuľky a vytvorenie novej, zmenenej tabuľky.

3.2 Kotlin

Kotlin je pomerne nový programovací jazyk, ktorý sa zameriava na programovací jazyk *Java* [5]. Výborne sa hodí na vývoj „*server-side*“ aplikácií. Dovoľuje vývojárom písať stručný a výrazný kód, pri zachovaní úplnej kompatibility, s existujúcimi technickými zásobníkmi založenými na prostredí jazyka *Java* [6]. Nasledujúce riadky sú prevzaté z knihy *Learn*

Android Studio 3 with Kotlin: Efficient Android App Development [5]. Programy jazyka Kotlin bežia na virtuálnom stroji Java (z anglického „*Java Virtual Machine*“, JVM).

Jazyk *Kotlin* bol vyvinutý českou spoločnosťou *JetBrains* v roku 2011. Zaujímavosťou je to, že ako jazyk *Java* bol pomenovaný po ostrove, ležiacom v Indonézii, tak aj *Kotlin* je pomenovaný po ostrove, ležiacom neďaleko Petrohradu, kde väčšina z členov pre vývoj *Kotlinu* sídli.

Kotlin má veľa schopností a zaujímavých vlastností ako programovací jazyk. Z tých najznámejších môžeme spomenúť napríklad to, že je objektovo-orientovaný. Tiež je staticky a silne typovaný. Avšak, na rozdiel od *Javy*, sa nemusí vždy deklarovať typ premennej pred jej použitím, pretože *Kotlin* používa odvodzovanie typu premenných. Umožňuje interoperabilitu s programovacím jazykom *Java*. Používa knižnice tohto programovacieho jazyka a možno ich použiť taktiež z programov písaných v *Jave*. Ďalšou obdivuhodnou vlastnosťou programovacieho jazyka *Kotlin* je to, že funkcie môžeme používať všade tam, kde premenné. Funkcie je možné preniesť zo vstupu parametra do iných funkcií a dovoľuje vrátiť funkcie z iných funkcií. Táto posledná vlastnosť poukazuje na funkcionalitu tohto programovacieho jazyka.

Napriek všetkým týmto výhodám, má *Kotlin* slabú stránku v tom, že je to pomerne nový jazyk a neexistuje o ňom zatiaľ veľa príručiek. Pretože je tento programovací jazyk nový, nesie so sebou určitú otázku v tom, ako dobrý je. Pri zaučovaní tímu ako *Kotlin* používať, môže byť popri tom stratený čas a po určitej dobe aj peniaze.

Ďalšie programovacie jazyky na vývoj mobilných aplikácií

Informácie o programovacích jazykoch sú prebraté z knihy *Android – kompletní průvodce vývojáře* [9], a tiež z tejto webovej stránky [4], ak nie je uvedené inak.

1. **Java** – je to objektovo-orientovaný jazyk, ktorého syntax sa podobá programovaciemu jazyku *C* alebo *C++*. Pôvodne bol vyvinutý spoločnosťou *Sun Microsystem*, v súčasnosti však patrí spoločnosti *Oracle*. Tento jazyk sa neprekladá priamo do strojového kódu, ale do takzvaného „*bajtkódu*“. Ten je potom interpretovaný základným virtuálnym strojom *Java* (z anglického „*Java Virtual Machine*“, JVM). Jazyk *Java* teda nie je závislý na platforme. *Java*, podobne ako *Kotlin*, sa používa na tvorbu mobilných aplikácií na platforme *Android*.

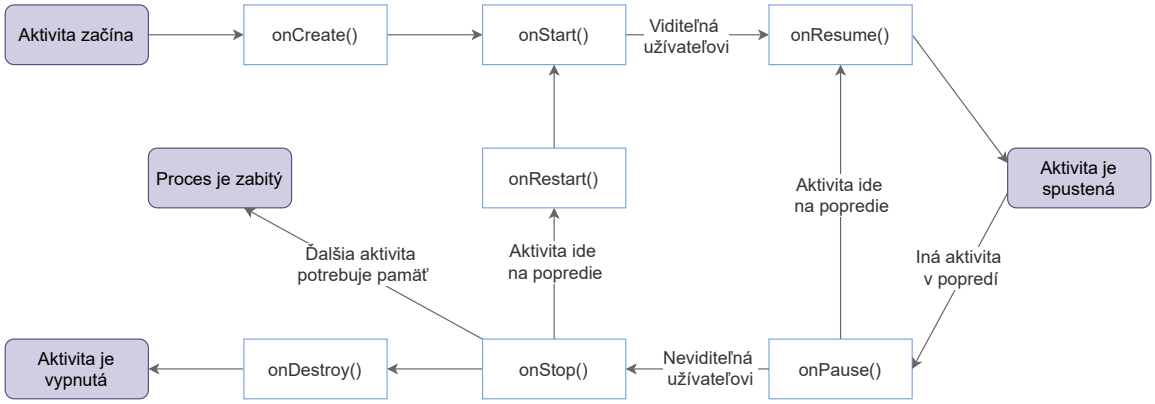
2. **Swift** – je moderný a bezpečný objektovo-orientovaný programovací jazyk navrhnutý firmou *Apple*. Tento programovací jazyk je ľahký na čítanie a jednoducho sa v ňom píše. *Swift* je celkom malý jazyk. Ponúka len niekoľko základných typov a funkcionalít. [13]. V roku 2016 prekonal *Swift* programovací jazyk *Objective-C* na písanie *iOS* aplikácií.

3. **Objective-C** – objektovo-orientovaný programovací jazyk, ktorý bol ako prvý podporovaný spoločnosťou *Apple* na písanie mobilných aplikácií. Jazyková syntax bola odvodená od programovacieho jazyka *C* a objektová-orientovanosť od programovacieho jazyka *SmallTalku*. Napriek niektorým kritikám je tento jazyk stabilný a používa sa už desiatky rokov. Od príchodu jazyka *Swift* však jeho popularita klesá.

Životný cyklus aktivity

Väčšinu informácií o životnom cykle som brala z knihy [9], a tiež [14], kde sa tejto téme do hĺbky venujú. Životný cyklus aktivity je daný funkciami, ktoré systém volá v rôznych situáciach aktivity. Na obrázku 3.3 môžeme vidieť štyri hlavné fázy tohto cyklu. Sú to:

- Aktivita začína
- Aktivita je spustená – v tejto fáze je aktivita viditeľná používateľovi a dovoľuje mu s ňou pracovať.
- Proces je zabitý – aktivita je v tejto fáze zničená kvôli nedostatku pamäti.
- Aktivita je vypnutá – táto fáza predstavuje aktivitu, ktorá je zrušená.



Obr. 3.3: **Schéma znázorňujúca životný cyklus aplikácie.** Na obrázku vidieť, akými rôznymi fázami aktivita, pri jej činnosti, prechádza a ktorými funkciami sú tieto činnosti spôsobené. Týmto životným cyklom prechádza každá jedna aktivita, s ktorou používateľ narába.

Ak potrebujeme v niektorej fáze životného cyklu uskutočniť špeciálne akcie, *Android* umožňuje tieto funkcie prepísať a docieliť tak želaného výsledku.

Obrázok 3.3 ukazuje spúšťanie funkcií, naviazaných na životný cyklus, v rôznych fázach aplikácie. Zavolaním funkcie `onCreate()` sa na pozadí vytvárajú používateľské rozhrania a rôzne ďalšie premenné a objekty, ktoré sú potrebné pre fungovanie aktivity. Aktivita je však stále zastavená a neviditeľná používateľovi. Ďalšími funkciami, ktoré umožňujú zobrazenie aktivity na popredí, sú `onStart()` a `onCreate()`. Rozdiel medzi týmito dvoma funkciami je ten, že `onStart()` uskutočňuje všetky vyžadované funkcie, ktoré sú nevyhnutné k normálnemu zobrazeniu používateľského rozhrania aktivity. Funkcie `onResume()` má za následok presunutie aktivity z pozadia na popredie. Pri príchode inej aktivity do popredia sa spustí funkcia `onPause()` a aktivita, ktorá bola predtým na popredí sa presúva do pozadia. Ak sa spustená aktivita zastaví, nie však z dôvodu nedostatku pamäti, volá sa funkcia `onStop()`. Aktivita, ale naďalej ostáva v takzvanom *Black Stacku*, odkiaľ môže byť volaná funkciou `onRestart()`. Poslednou funkciou životného cyklu aktivity je `onDestroy()`. Táto funkcia ukončuje aktivitu. Nezáleží však na tom, či ju ukončí používateľ, alebo sa takto udeje kvôli zmene usporiadania aktivity, čo môže byť spôsobené rotáciou zariadenia.

3.3 Ukladanie dát

Pri návrhu takmer každej aplikácie potrebujeme uchovávať informácie dlhodobo. Potrebujeme si zapamätať kroky používateľa, reagovať na ne, a tiež ich zobraziť pri opätovnom spustení. Aplikácie môžu pracovať s veľkým počtom dát, ktoré môžu byť uložené interne, pomocou *SQLite* databázy, alebo externe na internetových serveroch. Ďalším typom ukladania je s využitím zdieľaných vlastností, anglicky *SharedPreferences* (viz. 3.3.2), alebo ukladanie dát priamo do súborov [9].

Aplikácia *Marker* nepotrebuje na ukladanie dát, veľké internetové úložiska, ale vystačí si aj s lokálnou databázou *SQLite*, ktorá je uložená priamo v zariadení s *Androidom*. Pri zapamätaní si, aký režim má táto aplikácia nastavený aj po jej vypnutí, využíva zdieľané premenné.

3.3.1 SQLite

Stručne povedané, *SQLite* je verejný softvérový balík, ktorý poskytuje systém pre správu relačných databáz. Relačné databázové systémy slúžia na uloženie používateľom zadaných informácií do tabuliek [7].

V súčasnosti patrí medzi najviac používané databázové stroje na svete. Nemusí sa inštalovať a nedisponuje žiadnymi konfiguračnými súbormi. *SQLite* však ukladá dáta v jednoduchých súboroch, z ktorých možno čítať, alebo zapisovať pomocou *SQLite* knižnice [14]. Čítanie dát má vyššiu prioritu ako zápis dát [9].

Táto databáza, na rozdiel od iných *klient-server* databáz, ako *MySQL* alebo *Oracle*, používa len klientskú časť. Znamená to, že dáta sa ukladajú do súboru, ku ktorému má prístup len klient. Napriek tomu, že *SQLite* vybavuje vždy len jednu požiadavku, jej výkon je veľmi vysoký [9].

Celá databáza *SQLite*, ktorá zahŕňa viaceré tabuľky, indexy spúšťače a zobrazenia, je zahrnutá len v jednom súbore na disku. Má malú stopu behu programu. Potrebuje menej ako megabyte na zostavenie a niekoľko megabytov pamäte. Formát tejto databázy je multiplatformový, to znamená, že môže bežať na rôznych počítačových platformách – môžeme tento súbor dát kopírovať medzi 34-bitovými a 64-bitovými systémami, alebo medzi architektúrami „big-endian“ a „little-endian“ [7].

ContentValues

Prvou a základnou časťou využívania databázy je zapisovanie informácii do nej. Pri *SQLite* databázach sa zapisovanie, a tiež aktualizovanie dát robia pomocou triedy nazývanej `ContentValues`[7].

Podobne ako v *Jave* `Hash mapa` alebo `Bundle`, aj *ContentValues* je „*key-value*“ trieda, avšak táto trieda je špeciálne navrhnutá tak, aby ukladala také druhy dát, aké *SQLite* databáza uchováva [14].

Spôsob vytvárania tejto triedy a aj uchovávanie množiny dát je zobrazené na obrázku 3.4. Do tejto triedy sa vkladajú dáta pomocou funkcie `put(String key, String value)`, kde pre hodnotu `key` sa použije názov stĺpca tabuľky, do ktorého sa má hodnota `value` vložiť [14].

[7] https://developer.android.com/reference/android/content/ContentValues

```
/**
 * Create ContentValues for Database
 */
private fun getContentValues(tableField: TableField) : ContentValues{
    val cValues = ContentValues()

    cValues.put(DatabaseTimetable.ID, tableField.tId.toString())
    cValues.put(DatabaseTimetable.SUB_NAME, tableField.tSubject)

    return cValues
}
```

Obr. 3.4: Príklad vkladania *UUID* hodnoty a textového reťazca do databázovej tabuľky pomocou triedy `ContentValues`.

CursorWrapper

Jednou vecou je zapisovanie dát do tabuľky, druhou je výber týchto dát z tabuľky. Pre načítanie konkrétneho riadka tabuľky, sa môže použiť trieda zvanú *CursorWrapper*[8].

CursorWrapper umožní zabaliť kurzor, ktorý prišiel z iného miesta a pridať naň nové funkcie [14].

V mojom projekte je *CursorWrapper* implementovaný v triede `TableFiledCursorWrapper.kt`, ktorá obsahuje jeden primárny konštruktér. V tomto konštruktéri sa triede predáva otázka (z anglického „*query*“), ktorou sa vykonávajú operácie pre vyberanie údajov z tabuľky. Trieda `TableFiledCursorWrapper.kt` obsahuje štyri funkcie, ktoré slúžia na vrátenie údajov z konkrétneho riadka jednej zo štyroch tabuliek databázy `SubjectDB.db`.

UUID

Pri vytváraní niekoľkých objektov je potrebné ich nejakým spôsobom odlíšiť. Toto odlíšenie by malo byť jedinečné, aby sme podľa neho mohli pristupovať ku konkrétnemu objektu. *Android* umožňuje použitie triedy, ktorá predstavuje nemenný univerzálny jedinečný identifikátor (z anglického „*Universally unique identifier*“, UUID).

UUID[9] predstavuje 128-bitovú hodnotu a nevyžaduje registračný proces. Táto veľkosť je nemenná a oproti iným alternatívam predstavuje priemernú dĺžku. Používa sa na triedenie, radenie, „*hešovanie*“ a samozrejme, ukladanie objektov do databáz. Hodnota *UUID* je znázornená na obrázku 3.5. Vo svojom normatívnom stave je tento identifikátor reprezentovaný ako tridsaťdva šestnástkových číslic. Zobrazovaný je v piatich skupinách, ktoré sú oddelené spojovníkom. Každá táto skupina má vopred daný počet znakov. *UUID* je teda zložený zo znakov v tvare 8-4-4-4-12 [21].

```
f81d4fae-7dec-11d0-a765-00a0c91e6bf6
```

Obr. 3.5: Príklad hodnoty *UUID*.

[8] https://developer.android.com/reference/kotlin/android/database/CursorWrapper
[9] https://developer.android.com/reference/kotlin/java/util/UUID

3.3.2 SharedPreferences

Je to trieda, ktorá umožňuje najjednoduchšou formou uložiť malý obsah dát alebo vlastnosti aplikácie. Trieda *SharedPreferences*[10] je implementovaná ako trvalá mapa na ukladanie a načítanie hodnôt typu *kľúč-hodnota*. Údaje uložené v *SharedPreferences* sú perzistentné, to znamená, že tieto údaje budú dostupné, aj keď sa aplikácia ukončí, alebo reštartuje. V tejto triede môžeme ukladať základné dátové typy ako napríklad `string`, `int` alebo `boolean` [9].

Nevýhodou využitia tejto triedy je to, že využíva drahé operácie, čo môže viesť k spomaleniu výkonu aplikácie. Často meniace sa vlastnosti alebo vlastnosti, pri ktorých možno tolerovať straty, by mali využívať iné mechanizmy na uchovávanie dát [16].

Postup pri vytvorení takejto premennej je jednoduchý a je zobrazený na obrázku 3.6. Stačí len vytvoriť zdieľanú premennú pomocou funkcie `getSharedPreferences()`, kde ako argumenty figurujú názov, pomocou ktorého je možné modifikovať túto premennú, a prevádzkový režim. Ďalej už len ostáva vytvorenie editora na modifikovanie zmien v tejto premennej.

```
val settingsPreferences : SharedPreferences =
                              getSharedPreferences("ThemePref",0)
val settingsEditor : SharedPreferences.Editor =
                              settingsPreferences.edit()
```

Obr. 3.6: **Obrázok znázorňujúci vytvorenie zdieľanej premennej.** Najprv sa vytvorí zdieľaná premenná s názvom predstavujúcim túto premennú, a prevádzkovým režimom ktorý sa nastaví na hodnotu 0. Ďalej nasleduje vytvorenie editora, pomocou ktorého môžeme modifikovať zmeny v tejto premennej.

3.4 Fragmenty a Aktivity

Dôležitým prvkom, s ktorým sa pri vývoji mobilných aplikácií stretne každý programátor, sú aktivity a fragmenty. V tejto časti bude popísaný rozdiel medzi týmito dvoma elementami.

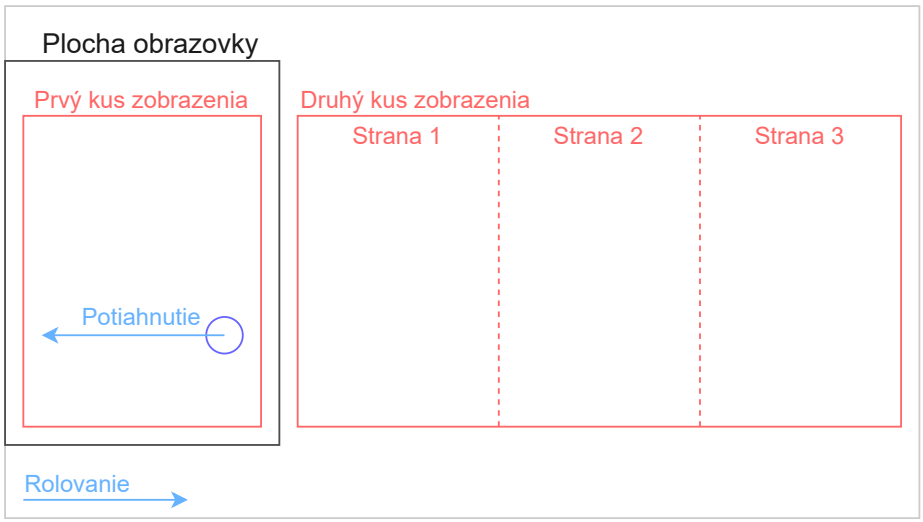
Aktivita je príklad triedy *Activity* v súbore nástrojov pre vývoj softvéru (z anglického „*Software development kit*“, SDK) pre *Android* . Je zodpovedná za vytvorenie okna informácií, s ktorým bude používateľ interagovať. Na tomto okne budú uložené všetky komponenty používateľského rozhrania, pomocou funkcie `setContentView(View)`. Aktivitu používateľ inicializuje vo funkcii `onCreate(Bundle)` so všetkými prvkami *layoutu*, s ktorými bude používateľ pracovať. Pre nájdenie konkrétneho *widgetu* sa volá `findViewById(int)` [14].

Fragment naopak predstavuje chovanie alebo časť používateľského rozhrania v aktivite. Môžeme spájať viacero fragmentov v jednej aktivite na vytvorenie používateľského prostredia s viacerými panelmi. Tiež jeden fragment možno použiť vo viacerých aktivitách. Fragment si môžeme predstaviť ako „*pod aktivitu*“, ktorá má vlastný životný cyklus, prijíma vlastné vstupné udalosti a do ktorej môžeme pridávať informácie alebo z nej tieto informácie mazať, kým aktivita beží. Fragment však musí byť vždy súčasťou aktivity, pretože jeho životný cyklus je ovplyvnený touto aktivitou [15].

[10] https://developer.android.com/reference/android/content/SharedPreferences

3.5 ViewPager

Android *ViewPager* je správca rozloženia, ktorý umožňuje používateľovi presúvať sa medzi listami v aplikácii posúvaním obrazovky mobilu vpravo, respektíve vľavo. Vo *ViewPageri* je každé nižšie zobrazenie zobrazené ako samostatná karta na rozložení [19].



Obr. 3.7: **Ukážka fungovania *widgetu ViewPager* v mobilných aplikáciach.** Obrázok znázorňuje, ako sa zobrazujú jednotlivé položky posúvaním obrazovky pomocou *ViewPager*. Tento obrázok je prevzatý z blogu o tomto komponente [19].

Na obrázku 3.7 možno vidieť, ako *ViewPager* funguje. Máme list zobrazení, ktoré chceme postupným posúvaním obrazovky sprava doľava a vice versa, zobrazovať. Prvou položkou na obrazovke je prvé zobrazenie v liste. Posunutím doprava sa zobrazí druhá položka v liste zobrazení a tak ďalej. Keď sa však dostaneme na koniec listu zobrazení, posunutím obrazovky doprava sa nezobrazí prvá položka listu, ale tu *ViewPager* akoby končí a jediným smerom, kam sa dá posunúť je doľava.

Pri vytváraní *ViewPageru* v mojom projekte, som postupovala podľa návodu v jedenástej kapitole z knihy *The Big Nerd Ranch Guide* [14]:

1. Prvým krokom je vytvorenie rozloženia, ktoré bude obsahovať *ViewPager*. V mojom programe sa jedná o súbor s názvom `activity_table_pager.xml`.

2. Ďalším krokom je vytvorenie triedy, ktorá vytvorí *ViewPager* a zároveň ho bude spravovať. Takouto triedou je v mojom projekte `TablePagerActivity.kt`, v ktorej sa inicializuje adaptér a vyhľadá sa, ktoré zobrazenie je aktuálne.

3. Posledným krokom je nastavenie adaptéra. Na vytvorenie možno použiť *FragmentStatePagerAdapter*, ktorý je podtriedou *PagerAdapteru*. Tento adaptér zredukuje konverzáciu na dve jednoduché funkcie a to `getCount()`, ktorá vracia počet stránok, ktoré adaptér bude vytvárať a `getItem(int)`, ktorá vracia pozíciu v poli fragmentov [23].

Vyššie spomínaný adaptér – *PagerAdapter*, ponúka ešte ďalší typ podtriedy, a to *FragmentPagerAdapter*. V mojom projekte bol použitý adaptér *FragmentStatePagerAdapter*, ktorý nepotrebný fragment zničí. Ak sa presunieme na druhú položku, vo *FragmentManagery* budú „*živé*“ položky jedna, dva a tri. Ak sa však presunieme na tretiu položku, prvá

položka bude z *FragmentManagera* odstránená, pretože nie je možné sa presunúť z tretej položky priamo na prvú. Pri prechode používateľa späť na druhú položku bude prvá položka obnovená [14].

FragmentPagerAdapter nikdy neničí nepotrebné fragmenty. Tento adaptér síce ukončí videnie fragmentu, no neodstráni ho z *FragmentManageru* a ostane tu, aj po prejdení na ďalšiu položku, živí. To, ktorý typ adaptéra používateľ použije, je len na ňom. *FragmentStatePagerAdapter* je však skromnejší pri narábaní s pamäťou [14].

Kapitola 4

Návrh používateľského rozhrania

Používatelia mobilných aplikácií, alebo iných elektronických zariadení, komunikujú so systémom pomocou používateľského rozhrania (z anglického „*User Interface*“, UI). Je to tá časť programu, ktorú používateľ najviac vníma, a preto by táto časť mala byť používateľovi zrozumiteľná. Samostatnou časťou návrhu používateľského rozhrania je návrh zameraný na človeka (z anglického „*User-centered design*“, UCR)). Jedná sa o prístup k návrhu, ktorý sa sústreďuje na to, čo ľudia potrebujú a robí z nich hlavným kritériom pri vzniku tohto návrhu [18].

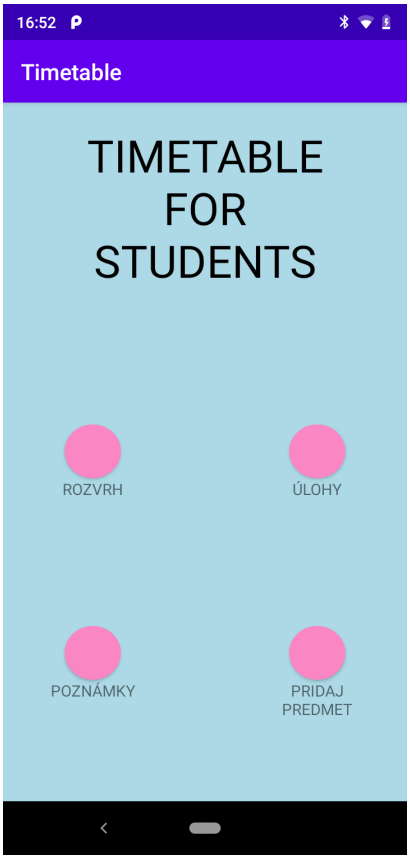
Táto kapitola bude bližšie popisovať prechod používateľského rozhrania aplikácie Marker z karty so štyrmi tlačidlami, po návrh, ktorý sa svojim vzhľadom podobá sofistikovanejším rozvrhovým aplikáciam. Bude opisovať používateľské rozhranie, ktoré dbá na to, aby čo najviac uľahčilo prácu používateľa s týmto rozhraním.

4.1 Prvotný návrh používateľského rozhrania

S implementáciou mobilných aplikácií som pred týmto projektom nemala žiadne skúsenosti, a tiež nevlastním mobil s platformou *Android*. Chvíľu mi preto trvalo, kým som si osvojila implementačné techniky a preskúmala, ako sú aplikácie, v mobiloch s *Androidom*, navrhované.

Pri počiatočnom návrhu aplikácie, by nemala byť len otázka ako bude aplikácia fungovať, ale aj ako bude vyzerať. Touto otázkou som sa zaoberala aj ja. Nechcela som svoj návrh odvíjať od prezerania si podobných aplikácií a inšpirovať sa nimi, ale chcela som si predstaviť, ako by mala vyzerať aplikácia pre rozvrh hodín v mojom mobile.

V tejto fáze som neriešila len samotný vzhľad aplikácie, ktorý som mimochodom neustále upravovala, ale aj rozloženie jednotlivých tlačidiel, s ktorými bude používateľ pracovať.



Obr. 4.1: **Prvotný návrh hlavnej karty aplikácie.** Obrázok zobrazuje prvý návrh hlavnej karty programu so štyrmi tlačidlami, ktoré odkazujú na príslušné karty.

Môj prvotný návrh spočíval v tom, že sa po spustení aplikácie zobrazí hlavná karta, ktorá bude obsahovať štyri tlačidlá ako vidieť na obrázku 4.1. Po kliknutí na tieto tlačidlá môže byť používateľ presmerovaný na kartu zobrazujúcu rozvrh hodín, alebo kartu pre pridanie nového predmetu, pre pridanie úlohy alebo poznámky. Po konzultácii s mojím garantom mi bolo odporučené, aby sa moja aplikácia začínala priamo tabuľkou, v ktorej sa používateľovi hneď zobrazia dané aktivity, namiesto zbytočného preklikovania sa.

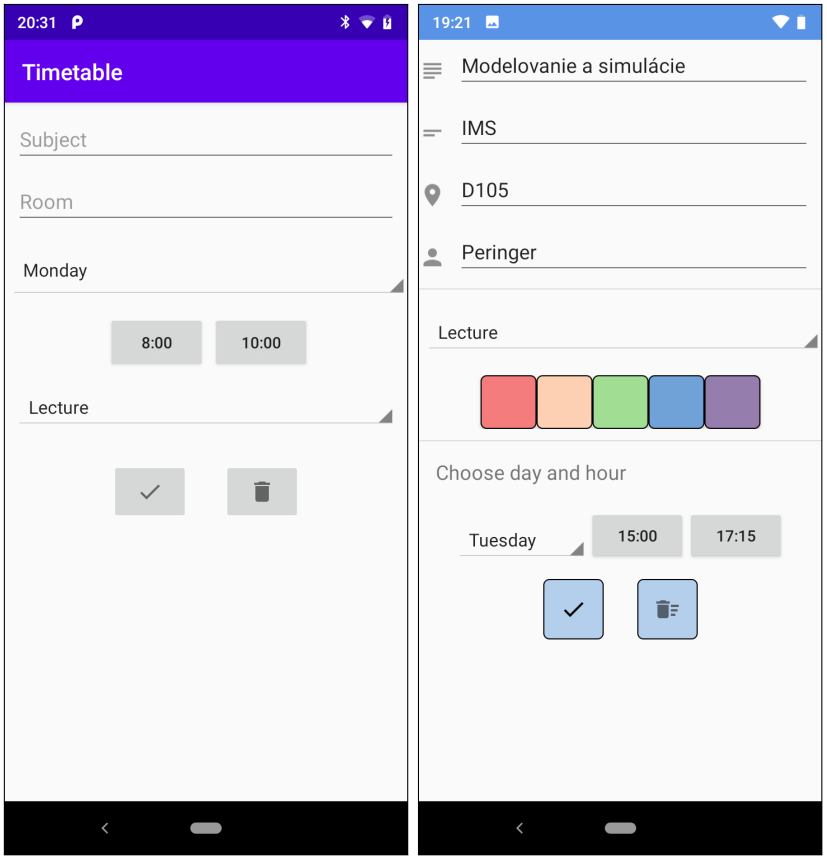
Od počiatku návrhu aplikácie som vedela, že si v nej chcem zaznamenávať úlohy, ktoré si používateľ môže separátne zapísať na nejakú kartu. Avšak, namiesto tlačidla v úvodnej karte aplikácie som sa rozhodla, že sa k danej úlohe používateľ dostane cez *Navigation-Drawer* v ľavom hornom rohu aplikácie. Neskôr som sa rozhodla, že si v aplikácii budem chcieť uchovávať aj informácie o skúške a poznámky sa budú vzťahovať len k úlohám, alebo skúškam. Tlačidlo pridaj predmet nahradí tlačidlo v tvare plus v *Toolbare*.

4.2 Pokrokovejšie používateľské rozhranie

Návrh rozloženia jednotlivých tlačidiel som už vyriešila a teraz prichádza na rad karta, kde si používateľ bude môcť uložiť údaje o svojej aktivite.

Pôvodný návrh tejto karty je zobrazený na obrázku 4.2. Vtedy tu bolo len veľmi málo komponent na uchovávanie tých najzákladnejších údajov o predmete. Keď som však aplikáciu nechala v ranom štádiu vývoja otestovať jednému z mojich blízkych ľudí, chýbal

mu na tejto karte záznam skratky predmetu, a tiež farba, ktorou by mohol dané aktivity rozlišovať. Tieto postrehy som označila ako veľmi prospešné a následne ich pridala na túto kartu.



Obr. 4.2: **Karty pre pridanie aktivity.** Prvý obrázok znázorňuje pôvodný návrh aplikácie pre pridanie predmetu. Druhý obrázok ukazuje jeho zmenu oproti prvému návrhu, na základe úvodného testovania.

Pri navrhovaní tejto karty som si dala obzvlášť záležať, keďže na nej používateľ minie najviac času zadávaním informácií o predmete. Dbala som na to, aby zadávanie údajov bolo čo najrýchlejšie, no zároveň, aby na tejto karte bolo všetko, čo si používateľ tejto aplikácie bude chcieť zaznamenať.

Na obrázku 4.2 je zobrazený aj novší návrh aplikácie, ktorý je vo svojej podstate, rovnaký aj vo finálnej podobe tejto karty. Prvé štyri elementy sú typu *EditText*, kde používateľ zadáva názvy jednotlivých informácií z klávesnice. Toto zadávanie sa nedá nijako predikovať, preto som sa rozhodla práve pre komponent *EditText*. Povinné je zadávanie názvu predmetu a jeho skratky, pretože to je, podľa mňa, na uchovaní si aktivity najdôležitejšie.

Na tejto karte sú tiež dva elementy typu *Spinner*. Sú to komponenty na vybratie dňa a typu aktivity. Keďže jednotlivé pomenovania týchto typov môžeme obsiahnuť do samostatných kategórií, rozhodla som sa ich výber uskutočňovať takýmto spôsobom. Obdobne je to aj pri výbere dňa. Hodnota pre deň je prednastavená na `Monday`, ak to používateľ nezmení.

Výber počiatočného času a koncového času sa realizuje kliknutím na tlačidlo, kde sa následne zobrazí dialógové okno *TimePicker*. Toto okno je pomôcka, ktorú ponúka *Android*

Studio, pre krajší a rýchlejší spôsob vybratia času. Hodnoty pre čas sú implicitne nastavené na osem hodín ráno, pre počiatočný čas, a desať hodín ráno, pre koncový čas.

Posledným prvkom tejto karty, ktorým som sa zaoberala, je výber farby. V tejto časti sa výber realizuje pomocou piatich tlačidiel, kde každý z nich symbolizuje inú farbu. V kapitole 4.3 opisujem, ako sa výber farieb realizuje pomocou dialógového okna *MaterialColorPickerDialog*.

Pre potvrdenie platných údajov, respektíve vymazanie predmetu, slúžia tlačidlá v dolnej časti plochy. Pôvodne som sa zamýšľala nad umiestením týchto tlačidiel v *Toolbare*, no takto mi to prišlo krajšie a intuitívnejšie.

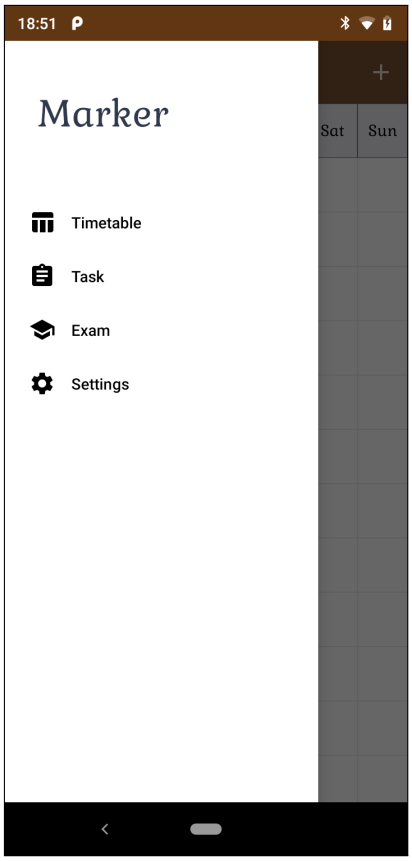
4.3 Finálny vzhľad aplikácie

V tejto časti prešla aplikácia najväčšími zmenami, čo sa týka návrhu. Veľa tu bolo pridaného, ale aj zmeneného. Najviac túto zmenu zasiahlo pridanie *NavigationDrawer* s patričnými položkami, ako aj implementácia kariet pre pridávanie úloh, skúšok a nastavení.

NavigationDrawer umožňuje vstup k cieľom a funkciám aplikácie, ako je napríklad prepínanie jednotlivých kariet. Tento komponent môže byť zobrazený trvalo na obrazovke, alebo jeho zobrazovanie môže používateľ ovládať klikaním na ikonu navigačnej ponuky [10] (konkrétne komponent Navigation drawer[1]).

Implementácia tejto komponenty mi prišla potrebná a v aplikácii ako je táto priam nevyhnutná. Uľahčuje a zjednodušuje prístup k iným kartám aplikácie. *NavigationDrawer* v mojej aplikácii obsahuje implementáciu hlavičky, ktorá v sebe nesie len názov danej aplikácie. Sú tu štyri položky, ktoré sú uložené v časti pre menu a umožňujú prepnutie sa na patričné karty. Ako vidieť na obrázku 4.3, *NavigationDrawer* v mojom projekte je pomerne strohý, no obsahuje všetko potrebné.

[1] https://material.io/components/navigation-drawer



Obr. 4.3: **Implementovaný *NavigationDrawer* v aplikácii *Marker*.** Obrázok ukazuje *NavigationDrawer*, ktorý obsahuje hlavičku, s názvom aplikácie a štyri položky v menu časti.

Ďalšiu významnejšiu zmenu, oproti predošlým verziám, zasiahlo pridanie kariet pre zadávanie úloh a skúšok. Keď som sa rozhodovala, ako budem tieto informácie ukladať, natrafila som na pomerne slušne vyzerajúci komponent, ktorý ponúka *Android*, a to *CardView*.

Tento prvok slúži ako vstupný bod k podrobnejším informáciám. Na jednu kartu sa môžu ukladať fotografie, text alebo obrázky. Umožňuje ukladať prvky rôznych veľkostí, a teda môže obsahovať fotografie alebo text rôznej dĺžky [10] (konkrétne komponent Cards[2]. Ukážka, ako vyzerá implementovaný *CardView* v aplikácii *Marker* je na obrázku 4.5.

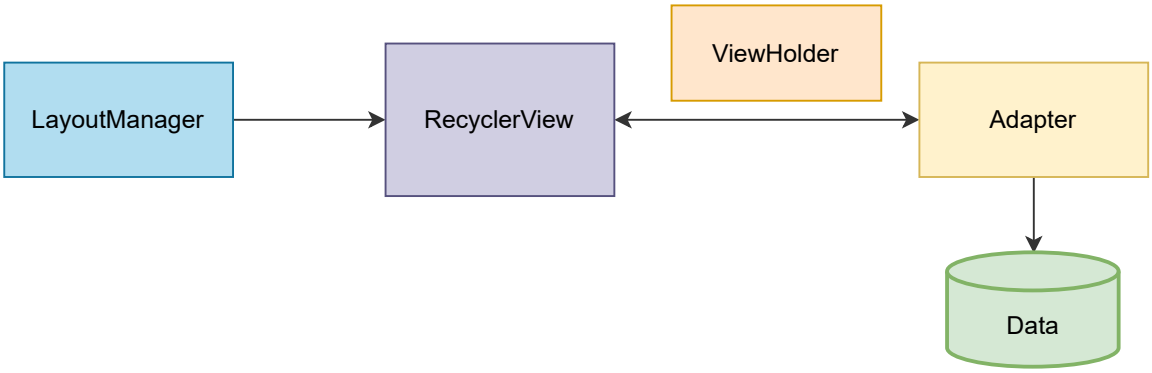
RecyclerView

RecyclerView predstavuje silný nástroj pre tvorbu používateľského rozhrania, ktorý dovoľuje používateľovi zobraziť list dát, napríklad fotky alebo text, flexibilným spôsobom. Nemusí však ísť striktne o list, zobraziť dáta možno aj vo forme mriežky alebo náhodným spôsobom [11].

Keď sa zobrazenie posunie mimo obrazovku, *RecyclerView* to dokáže znovu použiť a naplniť novými údajmi. Tento komponent umožňuje vývojárom časovo aj priestorovo zefektívniť aplikáciu, pretože nevytvára nové štruktúry ale „recykluje“ staré [11].

[2] https://material.io/components/cards

RecyclerView pracuje s dvoma triedami, ktoré musia byť taktiež implementované. Prvou je *Adapter* podtrieda a druhou *ViewHolder* podtrieda. *ViewHolder* robí v podstate len jednu činnosť, a tou je, že drží všetky zobrazenia, ktoré majú byť zobrazené. *RecyclerView* niekdy nevytvára zobrazenia, ale využíva na to *ViewHolder*. *RecyclerView* však priamo nevytvára ani *ViewHolder*, ale prostredníctvom adaptéru. Adaptér je kontrolný objekt, ktorý „sedí“ akoby medzi *RecyclerView* a dátovým súborom, ktorý by mal *RecyclerView* zobrazovať. Adaptér je zodpovedný za vytvorenie nevyhnutného *ViewHoldera* a pripojenie *ViewHoldera* k dátam z modelovej vrstvy [14]. Celé toto rozdelenie častí, súvisiacich s implementáciou *RecyclerView*, je ukázané na obrázku 4.4, pre ktorý mi bol inšpiráciou obrázok na tejto webovej stránke[3].



Obr. 4.4: **Schéma reprezentujúca fungovanie *RecyclerView*.** Obrázok ukazuje, ako sú jednotlivé položky na seba naviazané. *LayoutManager* hovorí *RecyclerView*, ako má ukladať dané zobrazenia. Následne mu *Adapter* predáva nové údaje z *Data* ak to potrebuje, a zároveň predáva tieto dáta na *ViewHolder*, ktorý ich následne zobrazí na zobrazenie a pošle ho na *RecyclerView*.

Vytvárať alebo organizovať správanie *RecyclerView* sa deje pomocou *LayoutManager*. *LayoutManager* slúži na určenie orientácie položiek na karte, usporiadanie pozície na karte, definuje typ listu, a tiež vypočítava pozíciu a veľkosť každej individuálnej položky v *RecyclerView* [22].

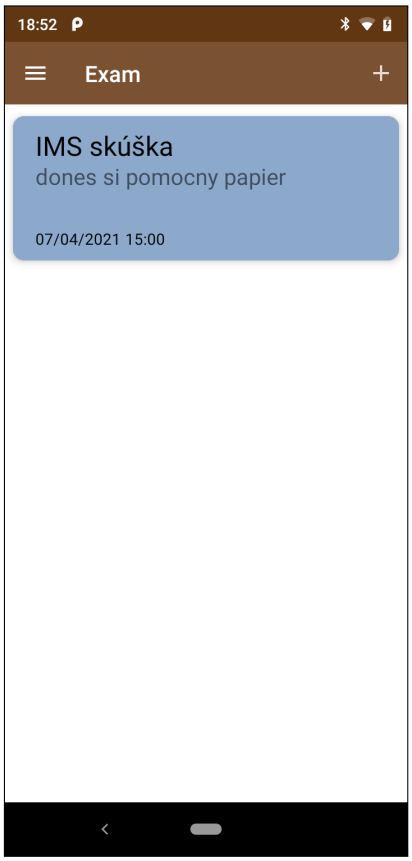
Poznáme tri typy tohto správcu:

1. **LinearLayoutManager** – ktorý zobrazuje položky v liste buď horizontálne, alebo vertikálne [14].
2. **GridLayoutManager** – usporadúva položky v mriežke, narozdiel od predchádzajúceho typu. Pri implementácii vyžaduje dva parametre, a to kontext a rozsah stĺpcov [22].
3. **StaggeredLayoutManager** – tento typ usporiadania využijeme vtedy, ak chceme zobraziť položky náhodným spôsobom. Teda, jednotlivé položky sa môžu líšiť z hľadiska rozmerov [22].

V mojom projekte, ako vidieť na obrázku 4.5, sú v *RecyclerView* postupne ukladané úlohy, respektíve skúšky, v zobrazení *CardView*. Na tento spôsob zobrazovania využívam

[3] https://www.andreasjakl.com/kotlin-recyclerview-for-high-performance-lists-in-android/android-recyclerview-flow/

LinearLayoutManager, ktorý jednotlivé položky ukladá lineárne za sebou. Implicitným nastavením pre tento typ správcu je orientácia vertikálne, ktorú taktiež využívam.



Obr. 4.5: **Implementácia *CardView* v aplikácii Marker.** Obrázok znázorňuje vloženie skúšky, s patričnými informáciami, vo forme *CardView*. Pri viacerých skúškach sa tieto zobrazenia budú ukladať pod sebou.

MaterialColorPickerDialog

V predošlej podkapitole 4.2 som spomínala, že budem chcieť zlepšiť implementáciu vyberania farieb. Táto zmena nastala vo finálnej úprave mojej aplikácie, kde som nastavovanie farieb pomocou piatich tlačidiel nahradila prvkom s názvom *MaterialColorPickerDialog*. Vedomosti o dialógových oknách a ich typoch som prevzala zo stránky o navrhovaní dizajnu položiek pre *Android* [10] (konkrétne komponent Dialogs[4]).

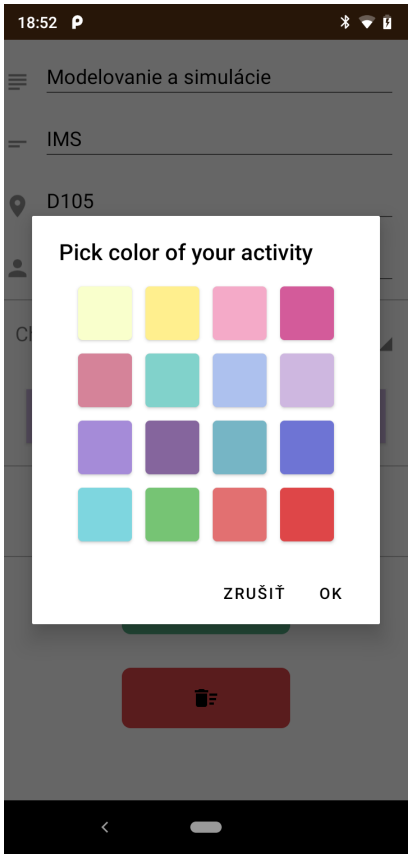
Jedná sa o dialógové okno, ktoré ponúka *Android*, na lepší, prehľadnejší a krajší spôsob výberu farieb. Dialógové okná v mobilných aplikáciach informujú používateľov o nejakých úlohách alebo varovaniach. Môžu tiež obsahovať dôležité informácie, výber z viacerých hodnôt, ako je výber farby alebo vyžadovať od používateľa dôležité rozhodnutie. Tieto okná fungujú ako modálne okná, a teda sú zobrazené v popredí aplikácie a zaniknú až keď ich používateľ zruší alebo vykoná potrebnú akciu.

Existuje niekoľko typov takýchto komponentov. Najznámejším z nich je výstražné dialógové okno, ktoré používateľa upozorňuje pred naliehavou informáciou, nejakými detailami

[4] https://material.io/components/dialogs

alebo akciami. Ďalším typom je potvrdzovacie dialógové okno, ktoré vyžaduje potvrdenie niekoľkých možností pred zatvorením tohto okna. V mojom projekte som použila ďalší typ, a to *MaterialColorPickerDialog*. Slúži na výber farieb z niekoľkých možností. Vybrať sa však dá vždy len jedna farba. Jedná sa teda o dialógové okno, ktoré používateľ môže ale nemusí otvoriť a vyberá si len jednu položku. Ak sa rozhodne nevybrať si žiadnu farbu, program má nastavenú predvolenú farbu, ktorú v takomto prípade použije.

Obrázok 4.6 ukazuje implementáciu *MaterialColorPickerDialogu* v mojej aplikácii. Pre výber farieb som použila farebnú škálu šestnástich farieb, ktoré spolu ladia a sú viac menej v pastelových odtieňoch. Pastelové odtiene farieb mám rada a prídu mi aj príjemnejšie na prezeranie, ak sú nimi zobrazené udalosti v tabuľke.



Obr. 4.6: **Implementovaný *MaterialColorPickerDialog* v aplikácii Marker.** Obrázok ukazuje, ako vyzerá implementácia *MaterialColorPickerDialog* v aplikácii *Marker*. Možno tu vidieť aj aká farebná škála bola zvolená pre výber farby.

Logo

To ako bude mobilná aplikácia reprezentovaná v *Google Play*, a aký zanechá prvý dojem sa najlepšie dosiahne kvalitným logom. Preto som si dala, pri vytváraní loga pre moju aplikáciu, veľmi záležať. Logo som tvorila v aplikácii *Canva*[5]. Názov aplikácie je *Marker*, čo v preklade znamená „značkovač“. Tento názov by mal symbolizovať to, že aplikácia neslúži

[5] https://www.canva.com/

len ako rozvrh hodín ale je to vlastne nejaký záznam pravidelných týždenných „značiek“, a teda aktivít.

Na obrázku 4.7 môžete vidieť, že logo je pomerne jednoduché. Pozostáva z názvu aplikácie nad ktorým leží starodávne pero, určené na kaligrafické písanie, ktorému tak trošku holdujem.

Obr. 4.7: Logo aplikácie *Marker*.

Kapitola 5

Implementácia aplikácie

V tejto kapitole budem bližšie popisovať implementáciu, ktorá viedla od kódu popisujúceho prázdnu tabuľky v hlavnej karte aplikácie, až po finálnu verziu aplikácie. Táto kapitola je rozdelená do troch častí. V prvej je opísaný spôsob vytvárania tabuľky pre ukladanie aktivít, vytváranie databázy a jej prvej tabuľky, implementácia kariet pre záznam informácií o aktivite a jej následné uloženie do rozvrhu. Taktiež je tu popísaná architektúra mojej aplikácie. Ďalšia časť sa venuje popisu vytvárania kariet pre pridanie úlohy, eventuálne skúšky, a všetkého čo k tomu patrí. Posledná časť ukazuje, ako som implementovala, a čo obsahuje karta pre nastavenia, a tiež implementáciu upozornení na skúšku.

Vo všetkých častiach presne pomenúvam, v ktorých triedach, môjho programu, sa jednotlivé implementácie nachádzajú.

5.1 Vytvorenie jadra rozvrhu hodín

Základom pre aplikáciu na uchovávanie záznamov je prehľadná tabuľka. Ak si tieto záznamy chceme uchovávať podľa dní a času, mala by to byť tabuľka, ktorá v prvom stĺpci, a tiež prvom riadku obsahuje potrebné informácie k tomuto uloženiu. Taktiež by tu mala byť karta, v ktorej bude používateľ zadávať informácie o svojej aktivite, ktoré sa následne zobrazia v tabuľke.

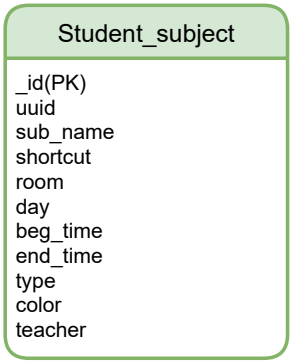
Prvým krokom v implementácii mojej mobilnej aplikácie, bolo práve vytvorenie takejto tabuľky. Táto tabuľka je vytvorená v abstraktnej triede zvanej `SingleFragmentActivity.kt`. Princíp vytvorenia tejto tabuľky je ten, že si pomocou triedy *DisplayMetrics()*[1] zisťujem, aká je výška a šírka plochy obrazovky. Tieto údaje následne vydelím patričným číslom. Pre šírku je to počet aktuálne nastavených dní, plus jedna (pretože prvý stĺpec určuje hodiny v danom dni). Pre výšku je to číslo, ktoré vznikne odčítaním hodiny, ktorá je nastavená ako koncová, od hodiny, ktorá je nastavená ako prvá, plus jedna.

Ďalším krokom nasleduje implementácia karty, do ktorej sa budú zapisovať informácie o jednotlivých aktivitách. Jedná sa o fragment triedu `TableFragment.kt` a jej prislúchajúcej vrstvy, ktorá v sebe nesie niekoľko položiek pre interakciu s používateľom. Bližšie informácie o fragmentoch a aktivitách popisujem v kapitole 3.4.

Ako je popísané v časti 5.1.1, zatiaľ sa jedná len o implementáciu zobrazenia, ktoré používateľ vidí, no v aplikácii sa neuchovávajú nijaké dáta a ani spolu objekty modelu a zobrazenia nekomunikujú. Preto ďalším krokom, ktorým som sa v implementácii ubrala, bolo vytvorenie databázy `SubjectDB.db`, ktorú som implementovala v triede `DBBaseHelper.kt`,

[1] https://developer.android.com/reference/android/util/DisplayMetrics

a databázovej tabuľky `Student_subjects`, pre uchovávanie údajov o aktivite, ktoré používateľ zadá ako vidieť na obrázku 5.1. Jednotlivé stĺpce tejto tabuľky inicializujem v triede `DatabaseTimetable.kt`. Jedná sa o *SQLite* databázu, ktorú viac popisujem v kapitole 3.3.1. V tejto databázovej tabuľke sa ukladajú informácie o názve predmetu a jeho skratke, miestnosti, učiteľovi, type udalostí, dni, v ktorom sa táto udalosť koná a počiatočnom a koncovom čase danej udalosti. Ukladá sa tu aj jednoznačný identifikátor *UUID*, ktorým jednotlivé udalosti rozlišujem. Tieto údaje sú prvkami triedy zvanej `TableField.kt`.



Obr. 5.1: **Obrázok znázorňujúci databázovú tabuľku *Student_subject* a jej atribúty.** Je to prvá zo štyroch tabuliek v databáze *StudentDb.db*. Uchovávajú sa v nej informácie o aktivite, ktorú si používateľ vytvára.

Riadiacim objektom pre túto časť aplikácie, je trieda `TableFieldLab.kt`, a taktiež trieda `TableFragment.kt`, ktoré slúžia na to, aby sa používateľom zadané informácie, zapisovali do objektov modelu, a následne zaznamenali do databázy, pre ich dlhodobé uloženie. Uskutočňujú tiež operácie pre vymazanie danej aktivity z databázy alebo aktualizáciu niektorých údajov v databázovej tabuľke.

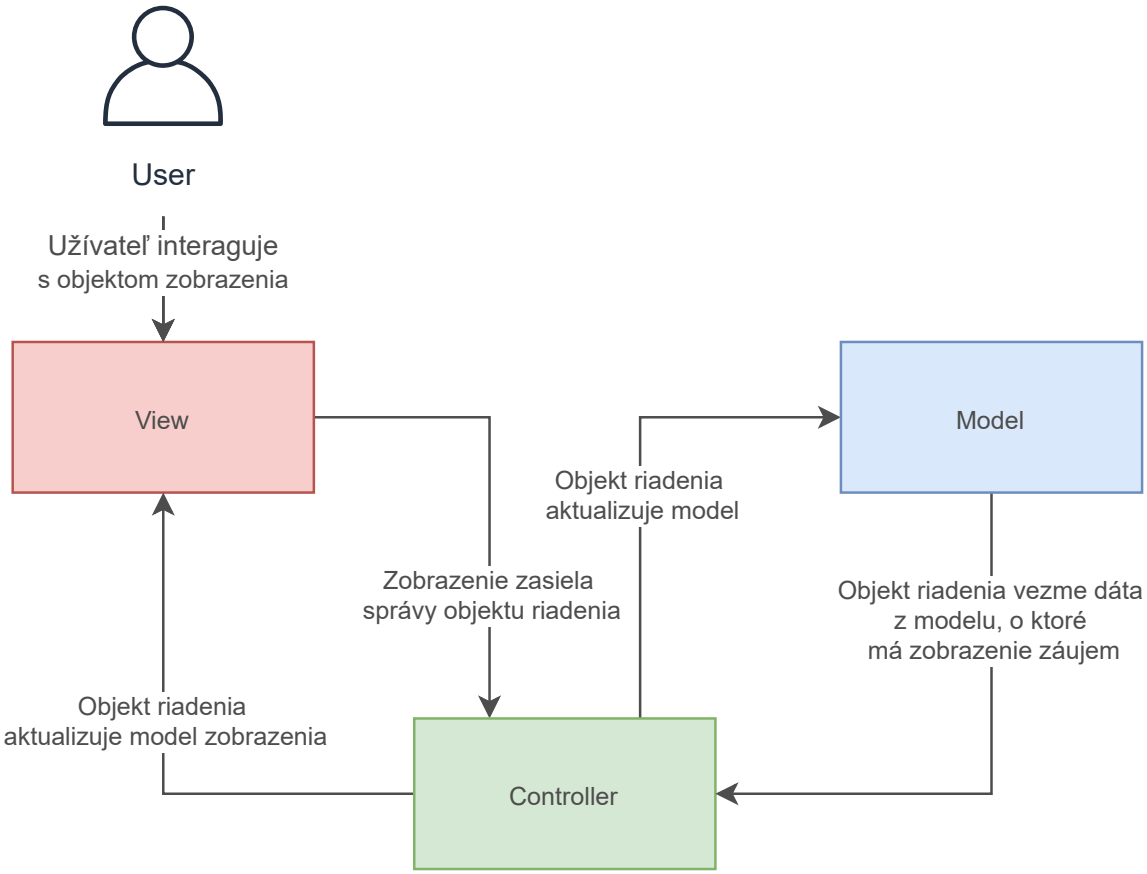
Posledným krokom je vytvorenie zobrazenia, ktoré sa bude ukladať do tabuľky na konkrétne miesta a bude v sebe niesť informácie o predmete. To sa realizuje vo fragmente `TableFieldFragment.kt`, ktorý je pripojený na aktivitu `TableFieldActivity.kt`. V tejto fragment triede sa vypočítava všetko potrebné k tomu, aby sa zobrazenie uložilo na konkrétne miesto v tabuľke podľa zadaného dňa a začiatočného času udalostí, a aby veľkosť tohoto zobrazenia vyhovovala koncovému času. Tiež sa tu vytvára zobrazenie, symbolizujúce jednu udalosť v tabuľke, ktoré sa naplní údajmi o danej aktivite..

V tomto kroku je aplikácia funkčná a obsahuje všetky časti, ktoré architektúra *MVC* musí mať. Používateľ si môže vytvárať školské predmety alebo iné mimoškolské činnosti, a zapisovať ich do tabuľky. Môže ich aktualizovať a tiež mazať. Ďalším krokom je implementácia kariet pre pridávanie úloh a skúšok, ktoré sú v aplikáciach, ako je táto, veľmi potrebné.

5.1.1 MVC architektúra

Poznatky o architektúre *Model-View-Controller* ako aj inšpiráciu na vytvorenie obrázka 5.2 som nadobudla z knihy [14].

Architektúra, podľa ktorej je navrhnutá moja aplikácie sa nazýva *Model-View-Controller* alebo *MVC*. V tejto architektúre musia byť všetky objekty buď modelové objekty, objekty zobrazenia alebo riadiace objekty. Obrázok 5.2 ukazuje, ako so sebou jednotlivé položky tejto architektúry komunikujú.



Obr. 5.2: **Obrázok znázorňujúci architektúru *MVC*.** Tento obrázok ukazuje tri časti, z ktorých sa skladá *MVC* architektúra a komunikáciu, ktorá medzi týmito časťami nastáva.

Modelové objekty uchovávajú dáta aplikácie. Triedy pre model sú typicky navrhnuté na modelovanie dát, s ktorými aplikácia pracuje. Zväčša sa jedná o uchovávanie informácií o používateľovi, dáta udalostí v rozvrhu hodín alebo to, či používateľ chce alebo nechce zasielať upozornenia pred skúškou. Tieto modelové objekty nemajú žiadnu vedomosť o používateľskom prostredí, nijako ich neovplyvňuje. Ich jediným účelom je uchovávanie a manipulácia s dátami, ktoré mu používateľ zadá.

Objekty zobrazenia predstavujú viditeľnú časť pre používateľa. Tieto objekty sa starajú o ich zobrazenie na plochu obrazovky a ako odpovedať na vstup používateľa, ako klik alebo dotyk.

Poslednou položkou v tejto architektúre sú riadiace objekty. Tieto objekty majú za úlohu „zviazať“ model a zobrazenie dohromady. Skrývajú v sebe logiku aplikácie. Sú navrhnuté tak, aby vedeli reagovať na rôzne udalosti vyvolané objektami zobrazenia. Tiež, aby riadili tok udalostí, ktorý prúdi do modelových objektov a z nich.

5.2 Prepínanie sa do kariet pre skúšku a úlohu

Pri vytvorení ďalších kariet, s ktorými bude používateľ pracovať, je potrebné vymyslieť, ako k nim bude pristupovať. Jedným zo spôsobov môže byť vytvorenie ikon v navigačnej lište hlavnej karty aplikácie. Každá ikonka by v takomto prípade symbolizovala jednu kartu.

Tento spôsob ale nepovažujem za najlepší, pretože veľa ikon môže používateľa zmiasť, a tiež to nevyzerá pekne. Ďalším, mne viac vyhovujúcim spôsobom, je použite *NavigationDraweru* a jednej ikonky zvanej *HamburgerIcon*.

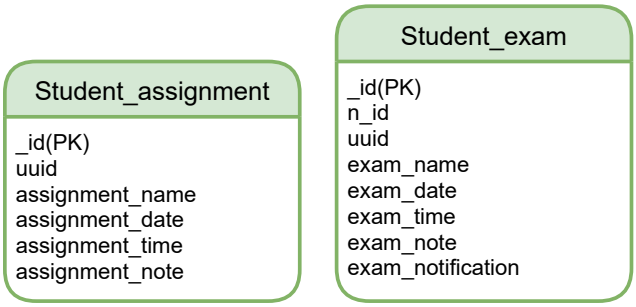
Implementáciu *NavigationDraweru* realizujem v týchto štyroch triedach:

1. `AssignmentActivity.kt`
2. `ExamActivity.kt`
3. `SettingsActivity.kt`
4. `SingleFragmentActivity.kt`

kde v každej z týchto štyroch tried implementujem triedu s názvom *NavigationView.OnNavigationItemSelectedListener*[2]. Táto trieda sa stará o manipuláciu s položkami v menu. Pri jej implementácii musí byť zahrnutá funkcia *onNavigationItemSelected(MenuItem)*, v ktorej sa definuje, ktorá karta, sa pri kliknutí na jednotlivú položku v menu, otvorí. Konkrétny príklad implementácie *NavigationDraweru* v aplikácii *Marker* je na obrázku 4.3.

V ďalšom kroku prichádza na rad samotná implementácia kariet, ku ktorým sa bude cez *NavigationDrawer* pristupovať. Tieto karty budú od používateľa vyžadovať ďalšie informácie, preto je rozumné, si tieto údaje uchovávať v databáze, ktorú som si vytvorila. Druhou tabuľkou, ktorú táto databáza obsahuje, je `Student_exam`, ktorej stĺpce definujem v triede `DatabaseExam.kt`. Do tejto tabuľky sa ukladajú informácie o názve skúšky, dátume skúšky a času, kedy sa skúška koná. Uchovávajú sa do nej prípadné poznámky a dĺžka upozornenia. Ak si používateľ zadá, že chce k tejto skúške zaslať upozornenie, implicitne sa dĺžka tohto upozornenia nastaví na nulu, čo značí, že upozornenie príde v čas konania skúšky. Ďalej sa do tabuľky zaznamenáva jednoznačný identifikátor – *UUID*, ktorý odlišuje jednotlivé skúšky a identifikátor pre notifikácie. Všetky tieto položky sú definované v triede `ExamField.kt`.

Treťou tabuľkou, ktorou databáza disponuje, je tabuľka `Student_assignment`, ktorá je obdobná ako tabuľka pre skúšku, ale neobsahuje stĺpce pre uchovávanie notifikácií. Podobne, ako v predchádzajúcom odseku, sú stĺpce pre túto tabuľku definované v triede `DatabaseAssignment.kt` a definícia položiek, na ukladanie údajov pre úlohu, je realizovaná v triede `AssignmentField.kt`. Tieto tabuľky sú zobrazené na obrázku 5.3.



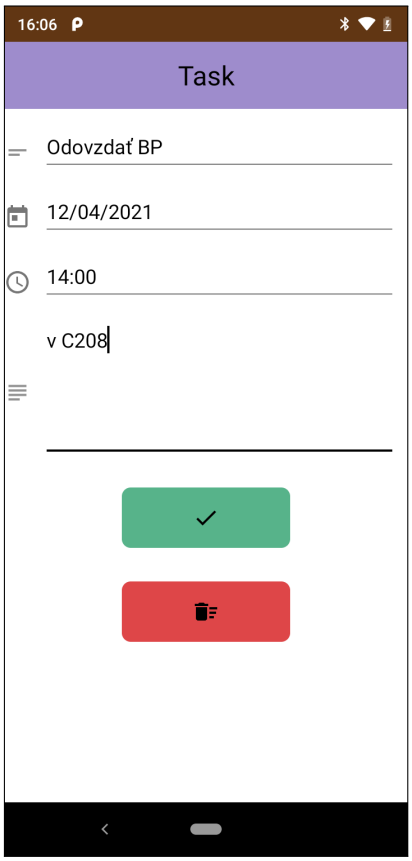
Obr. 5.3: **Obrázok znázorňujúci databázové tabuľky *Student_assignment* a *Student_exam* a ich atribúty.**

[2] https://developer.android.com/reference/com/google/android/material/navigation/NavigationView.OnNavigationItemSelectedListener

Po vytvorení tabuliek, nasleduje samotná implementácia tried, v ktorých sa budú uskutočňovať operácie s databázovými tabuľkami. Jedná sa o triedy `ExamFieldLab.kt` a `AssignmentFieldLab.kt`. Tieto triedy obsahujú funkcie, ktoré umožňujú vloženie údajov do tabuľky, výber údajov z tabuľky, aktualizovanie údajov, alebo ich samotné vymazanie.

Ako je popísané v kapitole 4.3, informácie o skúške a úlohe sa ukladajú na komponent s názvom *CardView*, a tieto karty sa vkladajú na *RecyclerView*. Postupné kroky, pre implementáciu *RecyclerView*, sú popísané v kapitole 4.3. Týmito krokmi som sa riadila v triede `ExamActivity.kt`, kde som implementovala *RecyclerView* pre skúšku a v triede `AssignmentActivity.kt`, kde som implementovala *RecyclerView* pre úlohu. Obrázok 4.5 ukazuje ako táto implementácia vyzerá na karte, kde sa ukladajú skúšky.

Podobne, ako na úvodnej karte, aj na kartách pre zápis úloh a skúšok, je v pravom hornom rohu tlačidlo plus, ktoré používateľa presmeruje na kartu, v ktorej je možné pridať úlohy, respektíve skúšky. Jedná sa o aktivitu (pre skúšku je to trieda `ExamView.kt` a pre úlohu ide o triedu `AssignmentView.kt`), ktorá zobrazuje jednotlivé položky zobrazenia a implementuje kroky, ktoré položky daného zobrazenia vykonávajú. Obrázok 5.4 ukazuje, ako vyzerá aktivita na pridanie informácií o úlohe v aplikácii *Marker*.



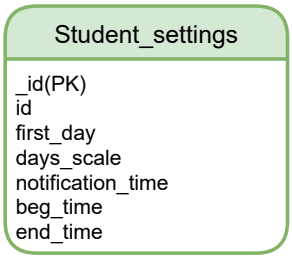
Obr. 5.4: **Obrázok znázorňujúci kartu, v ktorej sa zapisujú údaje o úlohe.** Na tejto karte je možné zapísať názov úlohy, dátum a čas, kedy má byť úloha vykonaná a poznámku k tejto úlohe.

V triede `ExamView.kt` sú ďalej implementované upozornenia. Tie však popisujem v nasledujúcej kapitole 5.3.

5.3 Implementácia nastavenia a upozornení

Poslednou kartou, ktorú obsahuje moja aplikácia, je karta pre nastavenia. Pri konzultovaní tohto projektu so svojim garantom, mi bolo odporučené, aby sa tabuľka vedela prispôsobovať používateľovi. Teda, ak používateľ nemá aktivity cez víkend, môže si v nastaveniach prispôsobiť, aké dni v týždni chce v tabuľke mať, alebo tiež, akým dňom chce v týždni začínať. V nastaveniach si ďalej môže prispôsobiť, aká bude začiatočná a koncová hodina v jeho dni, koľko minút pred skúškou chce dostať upozornenie, alebo nastaviť tmavú tému aplikácie.

Uchovávanie informácií, ktoré používateľ zadá v nastaveniach, je realizované v samostatnej databázovej tabuľke s názvom `Student_settings` a stĺpce, pre túto tabuľku, sú definované v triede s názvom `DatabaseSettings.kt`. Obrázok 5.5 ukazuje danú tabuľku s príslušnými atribútmi. Jednotlivé položky, nastavované v nastaveniach, sa ukladajú do triedy `SettingsField.kt`. Zaujímavosťou je, že po vytvorení databázovej tabuľky pre nastavenia, sa do nej vložia implicitné hodnoty pre každú časť nastavenia, a pri opakovanom menení nastavení, sa tieto položky len aktualizujú. Preto má tabuľka pre nastavenia len jeden riadok, ktorý sa aktualizuje podľa potrieb používateľa.



Obr. 5.5: **Obrázok znázorňujúci poslednú databázovú tabuľku *Student_settings* a jej patričné atribúty.**

Funkcie pre prácu s databázou, pre nastavenia, sú implementované v triede s názvom `SettingsSave.kt`. Nachádzajú sa tu funkcie pre vloženie údajov do tabuľky, vybratie týchto údajov z tabuľky, aktualizáciu a vymazanie dát z tabuľky.

Pre nastavenie činností jednotlivých položiek karty pre nastavenia, využívam triedu `SettingsActivity.kt`. Po stlačení akéhokoľvek tlačidla, sa zobrazí dialógové okno, s inštrukciami pre používateľa. Okrem implementácie, čo jednotlivé tlačidlá nastavujú, táto trieda obsahuje aj niekoľko zaujímavých funkcií, ktoré slúžia na úpravu používateľom vybraných nastavení. Napríklad funkcia *sortList()* zoberie list dní, ktoré si používateľ praje mať v tabuľke a zoradí ho chronologicky. Následne vracia zoradený list týchto dní. Vo funkcii *getDayArray()* sa list dní triedi podľa toho, aký deň používateľ zadal ako prvý. Ak si vybral všetky dni v týždni, ale za prvý deň si zvolil stredu, táto funkcia zoradí položky v liste dní od stredy po utorok. Poslednou funkciou, ktorú by som rada spomenula, je funkcia s názvom *giveIndexes()*. V nej sa ako argument predáva objekt s údajmi v tabuľke nastavení a na základe týchto údajov sa nastavia indexy v dialógových oknách pre uskutočnenie nastavení. Teda, ak si používateľ za prvý deň zvolí stredu, teda index s číslom dva (počíta sa od nuly), po opätovnom stlačení na výber prvého dňa sa mu zobrazí dialógové okno, ukazujúce stredu ako zvolenú hodnotu.

Najviac ma v tomto kroku potrápilo prispôsobovanie tabuľky používateľovi z hľadiska dní. Na tento úkon využívam práve vyššie spomínané funkcie *sortList()* a *getDayArray()*.

Používateľ si môže z dialógového okna ktoré mu zobrazí všetky dni v týždni, vyťukať, aké dni chce mať vo svojej tabuľke zaznamenané. Následne sú tieto dni zoradené chronologicky pomocou funkcie *sortList()*, a tiež zoradené od dňa, ktoré si používateľ zvolil ako prvý, funkciou *getDayArray()*. Toto radenie sa zopakuje aj v ďalšom kroku, a to pri výbere prvého dňa v tabuľke. To čo ma pri tomto najviac potrápilo bolo, implementovať to tak, aby som vzala do úvahy všetky možné spôsoby ako to používateľ môže nastaviť a predísť tak fatálnym chybám.

Nastavenie tmavého režimu

Možnosť nastavovania tmavej témy v aplikáciach je sprevádzané čoraz väčšou obľubou. Šetrí to výdrž batérie a v noci sa tmavá obrazovka lepšie číta. S týmto účelom som sa rozhodla, implementovať tmavý režim obrazovky aj v mojej aplikácii.

Implementácia prepínania z tmavého režimu do svetlého režimu je pritom jednoduchá. Stačí vytvoriť duplicitné *value* súbory s prívlastkom *night* v *AndroidStudiu*, a tieto hodnoty v týchto súboroch nastaviť na také, aké chceme zobrazovať v tmavom režime. Potom stačí, ako vidieť na obrázku 5.6, prepínať jednotlivé režimy, pomocou triedy `AppCompatDelegate`, podľa nejakého príznaku.

Uchovávanie, aký režim má používateľ nastavený, realizujem pomocou zdieľaných premenných (anglicky „*SharedPreferences*“), kde ich význam a použitie bližšie opisujem v kapitole 3.3.2 a na obrázku 3.6.

```
if (isNightTheme) {
    AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO)
} else {
    AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_YES)
}
```

Obr. 5.6: **Ukážka kódu z aplikácie** *Marker***.** Obrázok ukazuje prepínanie z tmavého režimu na denný a vice versa, využívajúc triedu *AppCompatDelegate*, podľa príznaku, či bol používateľom vybratý tento režim alebo nie.

5.3.1 Notifikácie

Ako som už vyššie spomínala, aplikácia *Marker* umožňuje nastavovať upozornenia na blížiace sa skúšky. Táto implementácia sa uskutočňuje v triede `ExamView.kt`.

Upozornenia sú položky, ktoré sa zobrazia v zásuvke s upozorneniami (anglicky „*notifications drawer*“), ku ktorým používateľ pristúpi potiahnutím obrazovky z hornej lišty nadol [14].

Nasledujúce odseky popisujú postup vytvorenia upozornení v mojej aplikácii. Vedomosti o tomto vytvorení som čerpala z kapitoly o upozorneniach v knihe *Android Programming: The Big Nerd Ranch Guide* [14].

Pri odosielaní upozornení je potrebné v prvom rade vytvoriť objekt znakov – *PendingIntent*[3]. Tento *PendingIntent* vykoná vysielanie pomocou funkcie `getBroadcast()`, v ktorej ako argumenty figurujú kontext, jednoznačný kód, zámer (anglicky „*intent*“) a príznak. Po vykonaní tejto funkcie, operačný systém vykoná zámer, ktorý bol zabalený patričným

[3] https://developer.android.com/reference/android/app/PendingIntent

spôsobom a spustí sa trieda s názvom `ReminderExam.kt`, ktorá rozširuje triedu *BroadcastReceiver*[4]. V tejto triede sa vykonáva samotná inicializácia upozornenia, pomocou triedy *NotificationCompat.Builder*[5], s nastavením ikony a textu tohto upozornenia, a ďalších vecí potrebných na zobrazenie upozornenia.

Objekt pre upozornenia je vytvorený a o jeho vyvesenie, na lištu upozornení, sa postará systémová služba s názvom *NotificationManagerCompat*[6] zavolaním funkcie `notify(int, Notification)`. Obrázok 5.7 ukazuje, ako vyzerá vyššie spomínaná implementácia, v mojom programe.

```
val buildNotification : NotificationCompat.Builder? =
        context?.let {
            NotificationCompat.Builder(it, examView.ARG_EXAM_REMINDER)
                .setSmallIcon(R.drawable.ic_exam)
                .setContentTitle(
                    "${intent?.getStringExtra(NOTIFICATION_TITLE)}
                    o ${intent?.getStringExtra(NOTIFICATION_TIME)}")
                .setContentText(intent?.getStringExtra(NOTIFICATION_TEXT))
                .setPriority(NotificationCompat.PRIORITY_MAX)
                .setVisibility(NotificationCompat.VISIBILITY_PUBLIC)
                .setDefaults(NotificationCompat.DEFAULT_ALL)
                .setCategory(NotificationCompat.CATEGORY_REMINDER)
}
val managerNotification : NotificationManagerCompat? = context?.let {
    NotificationManagerCompat.from(it)
}
```

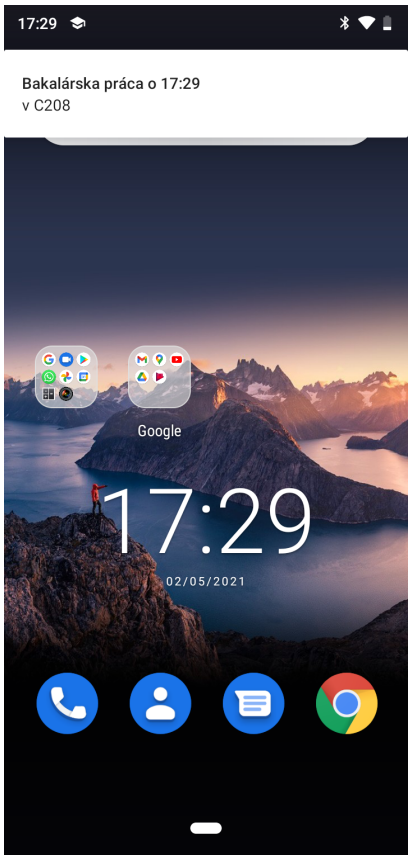
Obr. 5.7: **Ukážka kódu s vytvorením *NotificationCompat.Builder* a *NotificationManagerCompat*.** Na tomto obrázku je ďalej ukázané nastavovanie titulného nadpisu, obsahu upozornenia a ikony aplikácie. Kategória `CATEGORY_REMINDER`, sa používa na pripomienky. Priorita je nastavená na maximum a viditeľnosť je verejná, čo značí, že sa upozornenia budú zobrazovať aj na zatvorenej obrazovke.

Posledným krokom je nastavenie *AlarmManageru*[7], pomocou ktorého sa nastaví čas, kedy má upozornenie prísť. To sa začne zavolaním funkcie `getSystemService(ALARM_SERVICE)`, pomocou ktorej získa prístup k systémovým službám na platformách *Android*. V mojom prípade ide o službu pre upozornenia. Nasleduje samotné nastavenie času, kedy má upozornenie prísť, pomocou funkcie `set(AlarmManager.RTC_WAKEUP, eExamField.eNotification, pendingIntent)`. V tejto funkcii ako argumenty figurujú príznak, čas v milisekundách, kedy sa má upozornenie spustiť a *PendingIntent*, ktorý zobrazuje upozornenie v hornej lište obrazovky. V mojom projekte je príznak nastavený na hodnotu `RTC_WAKEUP`, čo značí, že systém zobrazí upozornenie, aj keď je zariadenie vypnuté.

Samotný čas, kedy sa notifikácia má zobraziť, vypočítavam sčítaním používateľom nastaveného dátumu v milisekundách a používateľom nastaveného času, taktiež v milisekundách. Tento čas začne plynúť v momente, keď používateľ prepne prepínacie tlačidlo v karte, kde sa vytvárajú skúšky, na zapnuté.

[4] https://developer.android.com/reference/android/content/BroadcastReceiver
[5] https://developer.android.com/reference/androidx/core/app/NotificationCompat.Builder
[6] https://developer.android.com/reference/androidx/core/app/NotificationManagerCompat
[7] https://developer.android.com/reference/android/app/AlarmManager

Príchod notifikácií bol spustený a otestovaný na rôznych zariadeniach, a to ako vyzerá je ukázané na obrázku 5.8.



Obr. 5.8: **Obrázok znázorňujúci upozornenie ku skúške.** Upozornenie ukazuje názov danej skúšky, čas, kedy sa koná a poznámku, ktorú si k tejto skúške zadal používateľ.

Kapitola 6

Testovanie

Dôležitou časťou tvorby každej mobilnej aplikácie je aj jej testovanie. Inak to nebolo ani v mojom prípade. Zhodnotenie, či aplikácia, ktorú vyvíjam, má vôbec zmysel, som realizovala doma so svojimi blízkymi. Kde som im moju aplikáciu dala v ranom štádiu vyskúšať. Pre overenie správnosti fungovania aplikácie na rôznych mobilných platformách som ju zverejnila na *Google Play*.

Testovanie je síce náročný a zdĺhavý proces, no prinieslo mi nové poznatky o slabých miestach mojej aplikácie. Toto testovanie som sa rozhodla realizovať v troch krokoch, a to prvotné testovanie, interné testovanie a otvorené zverejnenie aplikácie na trh, ktoré popisujem v tejto kapitole.

Poslednou časťou, ktorej sa budem venovať v kapitole 6, sú budúce zámery s mojou aplikáciou.

6.1 Prvotné overenie funkčnosti aplikácie

Vôbec prvé testovanie, ktoré som realizovala, bolo s mojimi súrodencami, ktorí sú taktiež študenti. Nechala som ich vyskúšať ešte neúplnú a nezverejnenú aplikáciu. Povedala som im, aby si založili nejaký konkrétny predmet, ktorí majú v danom semestri. Sledovala som pri tom, či zakladanie daného predmetu je pre nich jednoduché, a či sú tam všetky potrebné informácie, ktoré by si o predmete chceli uchovať.

Toto úvodné testovanie mi prinieslo dva nové poznatky, ktoré som neskôr implementovala vo svojom projekte. Prvým bolo uchovávanie skratky danej aktivity. Pôvodne som zamýšľala, že budem, ako skratku predmetu, uchovávať prvé tri alebo štyri písmená z názvu predmetu. Keď však, pri zadávaní predmetu, môj tester hľadal, kde by si mohol uchovať skratku, prišlo mi to ako lepší nápad ako to, čo som pôvodné zamýšľala. Ďalším bolo uchovávanie mena učiteľa alebo profesora daného predmetu. Prvotne som tento údaj vôbec nechcela dávať, keďže moja aplikácia by nemala slúžiť len študentom, ale všetkým ľuďom ktorí si chcú uchovávať týždenné aktivity. No po tejto poznámke som si povedala, že tam tento údaj dám. Používateľ si tu môže zaznamenať meno lektora nejakej mimoškolskej aktivity, nie len učiteľa a tento údaj nebude povinný.

Táto forma testovania mi ukázala, že aplikácia má zmysel, minimálne pre študentov, ktorí by s ňou dokázali pracovať, a že som na správnej ceste, čo sa týka jej vývoja.

6.2 Interné testovanie

Po zverejnení aplikácie na *Google Play* som si do tejto fázy testovania pozvala ľudí, ktorých som mohla priamo pozorovať pri narábaní s aplikáciou. Nechala som ich používať ju, kým som ich pozorovala, kde sa zasekávali, a kde boli pri jej použití zmätení [8]. Išlo o takzvané používateľské testovanie. Všímala som si, ako intuitívna je pre nich aplikácia, či obsahuje všetky prvky, ktoré by mala aplikácia pre zaznamenávanie činností mať, či sa komponenty zobrazujú správne na rôznych zariadeniach a tak ďalej. Výsledky z tohto testovania mi slúžili na odstránenie závažných chýb v mojej aplikácii.

Pri spustení tohto testovania som natrafila na jednu závažnejšiu chybu, a to, že na nižších verziách *Androidu*, konkrétne sa jednalo o zariadenie s úrovňou *Androidu* API 21, aplikácia padala, pri stlačení na tlačidlo `Add subject`. Je to tlačidlo v pravom hornom rohu obrazovky v tvare plus. Bolo to spôsobené použitím enumerácie `DayofWeek`, ktorá je podporovaná až od API 26 [1]. Keďže som chcela, aby sa moja aplikácia zobrazovala aj na nižších verziách, túto enumeráciu som nahradila jednoduchým textovým reťazcom, ako je to vidieť na obrázku 6.1.

```
"Monday" -> tTableField.tDay = DayOfWeek.MONDAY


"Monday" -> tTableField.tDay = "Monday"
```

Obr. 6.1: Príklad nahradenia enumerácie `DayofWeek` jednoduchým textovým reťazcom.

Pri práci s *Androidom*, ktorého verzia bola 5.0, som natrafila na ďalšie závažnejšie chyby. Keďže ani ja a ani nikto z môjho okolia nevlastní mobil s takouto verziou *Androidu*, rozhodla som sa, pre testovacie účely, nastaviť si takéto virtuálne zariadenie v *Android Studiu*. Ďalšia chyba s touto nižšou verziou nastala pri práci s *TimePickerom*. Keď som sa snažila dostať čas, z tohto výberu, pomocou funkcie `getHour()` alebo `getMinute()`, pri tejto verzii to nešlo a hádzalo to chybu. Pri bližšom skúmaní som sa z článku [20] dozvedela, že táto funkcia sa používa až od úrovni *Androidu* API 23. Vo verzii 5.0 sa používa funkcia `getCurrentHour()` respektíve `getCurrentMinute()`. Tento problém sa rieši jednoduchou podmienkou s využitím `Build.VERSION.SDK_INT` ako vidieť na obrázku 6.2.

```
//if level is less than 23, use getCurrentHour() method
if(Build.VERSION.SDK_INT < 23){
    int getHour = timePicker.getCurrentHour();
    int getMinute = timePicker.getCurrentMinute();
} else{
    int getHour = timePicker.getHour();
    int getMinute = timePicker.getMinute();
}
```

Obr. 6.2: Ukážka získania času z výberu času na staršej a novšej verzii *Androidu*.

Po dlhšom uvážení som sa však rozhodla využiť dialógové okno `TimePickerDialog`, ktoré je jednoduchšie na implementáciu a nemusela som využiť podmienku zmienenú vyššie, na obrázku 6.2.

Ďalšia chyba, na ktorú som pri tomto testovaní narazila, bola spôsobená upozorneniami. Na *Androidoch*, ktorých verzia bola 8.0 a nižšia, upozornenia síce prišli, ale bez zvuku. Od úrovne *Androidu*, väčšej ako 26, musia byť zvuky a vibrácie nastavované pomocou *NotificationChannel* a Android, na ktorom som tieto prvotné nastavenia upozornení testovala, spadal do tejto kategórie. Keď som neskôr prišla na to, že to na niektorých verziách síce upozornenie prichádza, ale bez zvuku, v oficiálnej dokumentácii ku *NotificationCompat.Builder*[1] som sa dočítala, že pri týchto verziách musím zvuk nastaviť funkciou `setDefault(NotificationCompat.DEFAULT_ALL)`.

Posledný poznatok, ktorý som nadobudla týmto testovaním bolo o migrácii databáz, ktorá je popísaná v kapitole 3.1.1. Keď som pridávala novú verziu aplikácie, pre svojich interných testerov, po spustení im padala a nemohli si ju spustiť. Jediné riešenie bolo ju odstrániť úplne a následne znova nainštalovať.

Na pravidelných konzultáciách, mi môj garant odporučil implementovať migráciu databázy, ktorú sa mi podarilo nasadiť do kódu (viz. obrázok 6.3), a pri ďalších zmenách verzie databázy, to už nehádzalo testerom chyby. Viac o migrácii databázovej schémy popisujem v kapitole 3.1.1.

```
/***
 * Upgrade databases table
 * @param db - database
 * @param oldVersion - old version of database
 * @param newVersion - number of new version of database
 */
override fun onUpgrade(db: SQLiteDatabase?,
    oldVersion: Int,
    newVersion: Int) {
        db?.execSQL("DROP TABLE IF EXISTS ${DatabaseTimetable.TABLE_NAME}")
        db?.execSQL("DROP TABLE IF EXISTS ${DatabaseExam.TABLE_NAME}")
        db?.execSQL("DROP TABLE IF EXISTS ${DatabaseAssignment.TABLE_NAME}")
        db?.execSQL("DROP TABLE IF EXISTS ${DatabaseSettings.TABLE_NAME}")
        onCreate(db)
}
```

Obr. 6.3: **Ukážka kódu pre migráciu databázy.** Ak sa zmení aktuálna verzia databázy, odstránia sa všetky jej tabuľky, a následne sa zavolá funkcia `onCreate(db)`, kde sa tieto modifikované tabuľky znovu vytvoria.

6.3 Zverejnenie aplikácie na *Google Play*

Po dôkladnom otestovaní aplikácie je na čase ju zverejniť širokej škále ľudí. Ide o takzvanú „*ostrú verziu*“, ktorá umožní, aby bola aplikácia dostupná širokej škále ľudí a nie len tým, ktorým ju sprístupním.

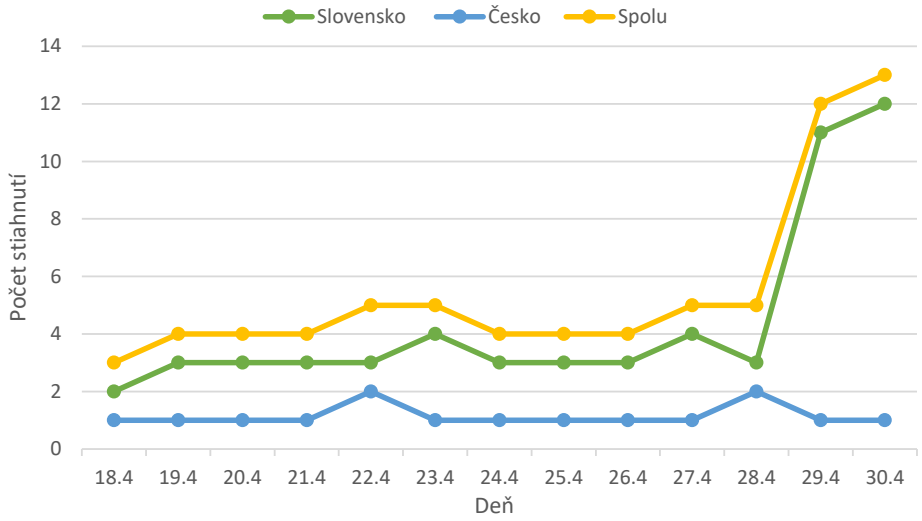
Pri zverejňovaní aplikácii na *Google Play* je potrebné vytvoriť si účet a zaplatiť jednorázový poplatok $25. Následne prejsť nejakými základnými nastaveniami k novej aplikácii, ako je názov, logo a pár snímok obrazovky, reprezentujúcich túto aplikáciu. Pri zadávaní týchto údajov sa očakáva, že aplikácia je už pripravená na zverejnenie, minimálne na testovacie účely. Preto ďalším krokom je generácia .apk (z anglického „*Android Package*“) alebo

[1] https://developer.android.com/reference/android/app/Notification.Builder.html

.aab (z anglického „*Application Bundle*“) súboru. Na záver stačí tento súbor nahrať v *Play Console* účte.

Aplikácia *Marker* má v súčasnej dobe zverejnené piate vydanie. Hoci som prvé štyri vydania generovala ako .apk súbory, pre toto piate som sa rozhodla generovať to ako .abb súbor, hlavne kvôli menšej veľkosti pri sťahovaní.

Svojim známym a kamarátom, ktorí by túto aplikáciu mohli využiť, som oznámila, že je voľne dostupná na *Google Play*. Následne som na mojom *Play Console* účte sledovala, ako krivka sťahovania pomaly rastie. Graf 6.4 ukazuje toto sťahovanie. Čakala som aj na spätnú väzbu, od týchto nových používateľov, aby som sa mohla od nej odraziť pri budúcich zámeroch s aplikáciou. Prvé ohlasy, hlavne zo strán študentov, boli, že je to užitočná aplikácia. S pozitívnymi ohlasmi prišli aj nejaké pripomienky na vylepšenie tejto aplikácie, ktoré popisujem v kapitole 6.4. Najviac ma v tomto kroku potešil fakt, že sa všetkým novým používateľom podarilo aplikáciu nainštalovať bez problémov.



Obr. 6.4: **Graf ukazujúci počet sťahovaní aplikácie *Marker* v *Obchod Play* na dennom intervale.** Ku dňu 30.4.2021 si aplikáciu stiahlo spolu trinásť ľudí v Česku a na Slovensku.

6.4 Budúce zámery s aplikáciou

Pri dôslednom testovaní aplikácie, a tiež zo spätnej väzby od nových používateľov tejto aplikácie, som prišla k istým poznatkom, ktoré by som mohla aplikovať v budúcom rozvoji tejto aplikácie.

Prvým takýmto vylepšením je pridávanie záznamov do tabuľky podľa hlasu. Predstavovala by som si to asi takým spôsobom, že v karte, kde sa vyžadujú nejaké údaje, buď o aktivite, úlohe alebo skúške, by si používateľ mohol tieto údaje zapísať pomocou hlasu. Jednoducho by stlačil na tlačidlo symbolizujúce diktafón, povedal by názov predmetu, a ten by sa následne zobrazil v položke, kde sa zapisuje názov predmetu. Toto by sa dalo aplikovať aj pri skratke predmetu, názve miestnosti alebo mene učiteľa. Určite by to uľahčilo, zjednodušilo a hlavne urýchlilo pridávanie týchto údajov. K tejto časti nadväzuje aj ďalšie vylepšenie, a to pripojenie aplikácie ku *Google Assistent*. Používateľ by si tak mohol rýchlo nastaviť skúšku alebo úlohu, bez zapnutia aplikácie.

V aplikácii *Timetable* 2.2.1 sa mi páčilo, že záznam predmetu mohol používateľ vytvoriť kliknutím na konkrétne políčko v tabuľke. Následne ho to presmerovalo na kartu, kde sa pridávajú údaje o novom predmete s už pred-vyplneným časom a dňom podľa toho, kde políčko, na ktoré používateľ klikol, ležalo. Tento spôsob by tiež mohol uľahčiť pridávanie nových aktivít.

Myslím si, že užitočnou vecou v tejto aplikácii, by taktiež mohol byť mesačný kalendár, do ktorého by si používateľ vedel zapísať vopred dôležité dátumy alebo iné veci, ktoré ho čakajú. Alebo by si tu mohol zaznamenať dátumy narodenín a menín svojich blízkych. Do tohto kalendára by si mohol zapisovať veci pár mesiacov dopredu, a takto by mu nič neuniklo.

Posledným vylepšením, ktoré by som v budúcej aplikácii *Marker* chcela mať je pridávanie známok, dosiahnutých bodov, alebo iných hodnotení, k aktivitám. Toto by sa mohlo zísť hlavne študentom a žiakom, pre ktorých sú známky alebo body podstatnou časťou každého predmetu.

Kapitola 7

Záver

Po úvodnom objasnení, ako si predstavujem, aby aplikácia vyzerala, a tiež po prehliadnutí podobných aplikácií som prešla k samotnému zhotoveniu návrhu. Ten som však behom implementácie niekoľkokrát „*vylepšovala*“. Následne som prešla k implementácii, kde som sa učila novým veciam a spoznávala, ako môže byť vývoj mobilných aplikácií pozoruhodný. Veľmi ma potrápilo vkladanie políčok, znázorňujúcich aktivitu, do tabuľky. Pôvodne som to zamýšľala vkladať ako zobrazenia na *RecyclerView*, to sa mi však nepodarilo, tak som komponent *RecyclerView* využila aspoň na kartách pre skúšku a úlohu. Samotnú implementáciu sprevádzalo neustále testovanie mnou a mojimi blízkymi. Následne bola aplikácia zverejnená v internej fáze testovania na *Google Play*[1], kde ju mohlo testovať viac ľudí a zasielať mi spätnú väzbu, ktorú som aplikovala pri dokončovaní implementácie.

Vo výsledku však celá aplikácia prekonala moje pôvodne očakávania. Aplikácia dokáže zaznamenať udalosti do tabuľky. Dokáže vytvárať záznamy skúšok a úloh, a dokonca zasielať upozornenia na skúšky. V nastaveniach je možné meniť rozmery tabuľky, ktorým sa vie obsah pôvodnej tabuľky prispôsobiť. Poslednou vecou, ktorú aplikácia umožňuje je nastavenie tmavého režimu a umožniť tak používateľovi šetriť batériu zariadenia a aj jeho zrak. K tejto aplikácii bolo vytvorené propagačné video[2] a plagát, ktoré sú uložené aj na priloženom CD.

Ak by som začínala s implementáciou aplikácie teraz, určite by som venovala väčší dôraz na to, aby vytváranie udalostí bolo ešte viac efektívnejšie a rýchlejšie. Tiež by som venovala väčšiu pozornosť designu. Hlavne tomu, aby jednotlivé farby so sebou viac ladili, a boli tak oveľa viac príjemnejšie pre používateľa.

Táto bakalárska práca mi dala množstvo nových poznatkov. Naučila ma pracovať s novým programovacím jazykom a novým databázovým systémom. Tiež mi ukázala, ako funguje vývoj mobilných aplikácií od úplného začiatku, až po oficiálne zverejnenie aplikácie na trh. Mohla som si vyskúšať, aké to je pracovať na veľkom projekte, ktorý bol od začiatku až do konca len v mojej réžii.

[1] https://play.google.com/store/apps/details?id=com.timetableRApp
[2] https://www.youtube.com/watch?v=v3iJy02gn38

Literatúra

[1] *DayOfWeek* [online]. 2021 [cit. 2021-03-07]. Dostupné z: https://developer.android.com/reference/java/time/DayOfWeek.

[2] *AppCode Smart IDE for iOS/macOS development* [online]. 2021 [cit. 2021-03-14]. Dostupné z: https://www.jetbrains.com/objc/.

[3] Biessek, A. *Flutter for Beginners: An introductory guide to building cross-platform mobile applications with Flutter and Dart 2*. 1. vyd. Packt Publishing, 2019. ISBN 978-1788996082.

[4] Chebbi, A. *Choosing the best programming language for mobile app development* [online]. 2019 [cit. 2021-03-14]. Posledné zdieľanie 02 Júl 2019. Dostupné z: https://developer.ibm.com/technologies/mobile/articles/choosing-the-best-programming-language-for-mobile-app-development/.

[5] Hagos, T. *Learn Android Studio 3 with Kotlin: Efficient Android App Development*. 1. vyd. Apress, 2018. ISBN 978-1-4842-3906-3.

[6] *Kotlin language documentation* [online]. 2021 [cit. 2021-03-23]. Dostupné z: https://kotlinlang.org/docs/kotlin-reference.pdf.

[7] Kreibich, J. A. *Using SQLite: Small. Fast. Reliable. Choose Any Three.* 1. vyd. O'Reilly Media, 2010. ISBN 978-0596521189.

[8] Krug, S. *Don't make me think, Revisited: A Common Sense Approach to Web and Mobile Usability*. 3. vyd. Pearson, 2014. ISBN 0-321-96551-5.

[9] Lacko Luboslav. *Android – kompletní průvodce vývojáře*. 1. vyd. Computer Press, 2017. ISBN 978-80-251-4875-4.

[10] *Material Design – Components* [online]. 2021 [cit. 2021-04-30]. Dostupné z: https://material.io/components?platform=android.

[11] Mehta, M. *Getting to know RecyclerView* [online]. 2020 [cit. 2021-04-25]. Posledné zdieľanie 24 September 2020. Dostupné z: https://medium.com/androiddevelopers/getting-to-know-recyclerview-ea14f8514e6.

[12] *What are database migrations?* [online]. [cit. 2021-03-15]. Dostupné z: https://www.prisma.io/dataguide/types/relational/what-are-database-migrations.

[13] Neuburg, M. *IOS 13 Programming Fundamentals with Swift: Swift, Xcode, and Cocoa Basics*. 1. vyd. O'Reilly Media, 2019. ISBN 978-1492074533.

[14] Phillip, B., Stewart, C. a Marsicano, K. *Android Programming: The Big Nerd Ranch Guide*. 3. vyd. Big Nerd Ranch, 2017. ISBN 978-0134706054.

[15] Servidoni, A. *Android Basics - Activities & Fragments* [online]. 2017 [cit. 2021-04-25]. Posledné zdieľanie 22 Marec 2017. Dostupné z: https://blog.avenuecode.com/android-basics-activities-fragments.

[16] *SharedPreferences* [online]. 2021 [cit. 2021-04-28]. Dostupné z: https://developer.android.com/reference/android/content/SharedPreferences.

[17] *SQLiteOpenHelper* [online]. 2021 [cit. 2021-03-15]. Dostupné z: https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.

[18] Stone, D., Jarrett, C., Woodroffe, M. a Minocha, S. *User Interface Design and Evaluation*. 1. vyd. Morgan Kaufmann, 2005. ISBN 978-0120884360.

[19] Suhail, M. *Android ViewPager – A Quick Guide* [online]. 2015 [cit. 2021-04-26]. Posledné zdieľanie 22 December 2015. Dostupné z: https://blog.appliedinformaticsinc.com/android-viewpager-a-quick-guide/.

[20] Thung, F., Haryono, S., Serrano, L., Muller, G., Lawall, J. et al. *Automated Deprecated-API Usage Update for Android Apps: How Far Are We?* 2020. 602-611 s. Dostupné z: https://hal.inria.fr/hal-02889832/document.

[21] *A Universally Unique IDentifier (UUID) URN Namespace* [online]. 2005 [cit. 2021-04-28]. Dostupné z: https://tools.ietf.org/html/rfc4122.

[22] Vaid, K. *RecyclerView LayoutManager Types* [online]. 2018 [cit. 2021-04-25]. Posledné zdieľanie 29 December 2018. Dostupné z: https://medium.com/@kamalvaid/recyclerview-orientation-types-9d15c3424d85.

[23] *Slide between fragments using ViewPager* [online]. 2021 [cit. 2021-04-28]. Dostupné z: https://developer.android.com/training/animation/screen-slide.

Príloha A

Obsah priloženého pamäťového média

Štruktúra popisujúca rozloženie jednotlivých súborov uložených na priloženom CD je popísaná nižšie.

```
/
├── Marker-app/
├── Marker-documentation/ .....zdrojové súbory technickej správy vo formátu LaTeX
├── app-release.aab ..... inštalačný súbor
├── Marker-documentation.pdf  technická správa k mobilnej aplikácii vo formáte .pdf
├── README.md ..... úvodné informácie o aplikácii
└── media/
    ├── Marker-screenshots/.....snímky obrazovky mobilnej aplikácie Marker
    ├── Marker-plagát.pdf .....propagačný plagát k mobilnej aplikácii
    └── Marker-video.mp4 .....propagačné video k mobilnej aplikácii
```

Príloha B

Plagát

Na propagáciu a stručné vysvetlenie toho ako aplikácia *Marker* funguje, som vytvorila plagát, ktorý je možno vidieť nižšie a jeho .pdf verzia je uložená aj na priloženom CD.

Obr. B.1: Plagát prezentujúci aplikáciu *Marker*.