

Gymnázium Arabská, Praha 6, Arabská 14

Programování

MATURITNÍ PRÁCE



Gymnázium Arabská, Praha 6, Arabská 14

Arabská 14, Praha 6, 160 00

MATURITNÍ PRÁCE

Předmět: Programování

Téma: DayPlanner

Autor: Jan Sváček

Třída: 4.E

Školní rok: 2024/25

Vedoucí práce: Mgr. Jan Lána

Třídní učitel: Mgr. Blanka Hniličková

Čestné prohlášení

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona č. 121/2000 Sb. (tzv. autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne

Anotace

Tímto projektem je mobilní aplikace pro Android zaměřená na plánování dne. Umožňuje spravovat jak jednorázové úkoly, tak opakované činnosti - návyky. Díky notifikačnímu systému uživatel dostává upozornění před začátkem úkolů. Návyky lze zaznamenávat a sledovat pomocí přehledných statistik ve formě map, grafů a dalších zajímavých údajů.

Pro lepší organizaci je součástí aplikace také časová osa, která pomáhá vytvořit realistickou představu o denním rozvrhu. Celkový koncept vychází z ověřených metod a přístupů k efektivnímu učení a budování návyků.

Abstract

This project is a mobile application for Android focused on daily planning. It allows users to manage both one-time tasks and recurring activities — habits. With the notification system, users receive alerts before tasks begin. Habits can be recorded and tracked through comprehensive statistics in the form of maps, charts, and other insightful data.

To enhance organization, the app includes a timeline, providing a realistic overview of the daily schedule. The overall concept is based on proven methods and approaches for effective learning and habit-building.

Úvod

Plánování času je klíčovým faktorem pro zvýšení produktivity a dosažení osobních cílů. Zaznamenal jsem, že mnoho existujících nástrojů se však zaměřuje buď na jednorázové úkoly, nebo na sledování návyků, přičemž obě oblasti jsou často oddělené. Tato aplikace se snaží spojit tyto dvě disciplíny do jednoho přehledného systému, čímž nabízí ideální orientaci v plánování dne.

Hlavním principem aplikace je blokové plánování, které umožňuje realisticky rozvrhnout čas a předejít přeplněným či neefektivním harmonogramům. Aplikace dělá návyky atraktivní pomocí přehledných statistik a umožňuje uživateli začít s návykem pomalu a postupně v průběhu času se posouvat k tíženému cíli. Tyto důležité principy vycházejí z knihy Atomové návyky od Jamese Cleara.

Aplikace kombinuje časovou osu, která vizualizuje denní plán, notifikační systém pro připomínání úkolů a statistiky ve formě grafů a map, které pomáhají uživateli sledovat jeho pokrok. Díky této integraci nejen podporuje plánování jednorázových akcí, ale zároveň motivuje ke konzistentnímu budování návyků.

Cílem bylo inovativním přístupem umožnit uživateli nejen plnit všechny úkoly, které má udělat, a nezapomínat na důležité schůzky, ale také mu pomoci s dlouhodobým rozvojem prostřednictvím budování zdravých návyků.

Zadání ročníkového projektu

Jedná se o mobilní aplikaci na Android, která je napsána v programovacím jazyce Java. Aplikace bude jednak sloužit pro plánování jednorázových akcí (úkoly, schůzky) a jednak opakovaných akcí (návyky). To uživateli umožní, že nebude zapomínat na to, co má daný den udělat, a navíc si uživatel bude moci sledovat svoje návyky v grafech nebo mapách. Cíl je udělat intuitivní aplikaci, kde si člověk bude moci pěstovat zdravé návyky a zapisovat si úkoly a události, aby na ně nezapomněl.

Hlavní myšlenkou projektu je, že se den zpravidla skládá z akcí, které jsou pro nás buď nové, nebo se často neopakují, a z našich návyků. Většina aplikací se soustředí pouze na jednu z disciplín, což vede k tomu, že uživatel musí mít více aplikací. Právě proto vzniká tato aplikace, která spojí obojí dohromady.

Obsah

1	Řešená problematika	3
1.1	Jednorázové akce a návyky	3
1.2	Problém současných aplikací	4
1.3	Přístup mé aplikace k problematice	5
2	Použité technologie	7
3	Architektura aplikace	8
3.1	Konfigurační nastavení aplikace	8
3.2	Adresářová struktura	10
3.3	Prezentační vrstva (Frontend)	12
3.3.1	Main activity	12
3.3.2	Statistiky	14
3.3.3	Tmavý a světlý režim	16
3.4	Vrstva aplikační logiky (Backend)	19
3.4.1	Rozdělení logiky	19
3.4.2	Autentizace a přihlášení uživatele	20
3.4.3	Habit	25
3.4.4	Task	27
3.4.5	Časová osa	28
3.4.6	Statistiky	29
3.4.7	Notifikační systém	31
3.5	Struktura databáze	33
4	Hlavní funkce aplikace Dayplanner	35
4.1	Hlavní aktivita	35
4.2	Uživatelský profil	37
5	Postup instalace	38

1 Řešená problematika

Tato kapitola pojednává o tom, jaký reálný problém moje aplikace řeší. Jsou zde zmíněny ověřené metody a postupy, ze kterých aplikaci vychází, aby byla pro uživatele co nejvíc nápomocná. Také je potřeba vysvětlit a upřesnit určité termíny, které se v této dokumentaci vyskytují.

1.1 Jednorázové akce a návyky

Každý den – a život obecně – se skládá ze dvou základních typů událostí. Na jedné straně stojí ty, které jsou nám důvěrně známé, stálé a tvoří pevnou součást naší rutiny. Mohou to být každodenní činnosti, jako je cesta do školy či do práce, venčení psa po návratu domů, ranní běh nebo víkendová návštěva sauny. Jsme na ně již zvyklí nebo si na ně začínáme zvykat.

Na druhé straně však vstupují do našeho dne události nové, nečekané a často nepředvídatelné. Ty vybočují z našeho běžného rytmu a vyžadují, abychom na ně reagovali a přizpůsobili se jim. Může jít o neohlášený test z matematiky, návštěvu nového kadeřníka, pracovní schůzi nebo i události, na které se má ročníková práce nevztahuje, jako je například přestěhování se do jiného města či náhlé onemocnění. Lze tedy říci, že den se skládá z prvků řádu a chaosu, dvou protikladů symbolizovaných principy Jin a Jang. To, co je známé a stabilní, odpovídá Jangu – představuje pevný řád a strukturu. Naopak vše nové, nečekané a neuchopitelné spadá pod Jin, tedy symbol chaosu. Společně vytvářejí dynamickou rovnováhu, která utváří nejen naše dny, ale i celý život. *„Řád a chaos jsou zobrazeny jako jin a jang ve slavném taoistickém symbolu dvou do sebe zaklesnutých draků. Řád ztělesňuje bílý, maskulinní drak, chaos pak jeho černý, femininní protějšek. Černá tečka v bílém poli a bílá v černém naznačují možnost transformace – zrovna když si říkáme, že svět je stabilní a známý, objeví se něco nového, nějaká nečekaná hrozba. A když se naopak vše zdá ztraceno, může ze zkázy a chaosu povstat nový řád.”*[1]

Co se mé aplikace týče, řád reprezentují návyky a chaos - jednorázové akce (úkoly, testy, schůze, termíny apod.).

1.2 Problém současných aplikací

Dovolím si říct, že nejsem zdaleka jediný, kdo již vyzkoušel nespočet různých aplikací, zabývajících se tématy jako je: plnění úkolů, sledování návyků, připomínky a tak dále. Většinou mi na nich něco chybělo. Vadilo mi, že abych měl přehled o všem, musel jsem mít aplikací několik, a to není praktické. Při navrhování své aplikace jsem vyšel ze zkušeností a snažil se přijít na řešení různých problémů, které jsou zmíněny níže.

Například problém takzvaných to-do listů, ať už papírových nebo digitálních, je ten, že si člověk zpravidla velmi často zapíše na daný den tolik věcí, že je realisticky nikdy nemůže všechny stihnout. Další den udělá to samé a to následně vede k tomu, že se věci stále hromadí, odsouvají a k žádnému pozitivnímu výsledku to nevede. Za prvé člověk nepracuje tak efektivně a za druhé se může mnohdy cítit přehlcený. Přestože za den stihne udělat větší množství úkolů, tak se nemusí dostavit žádný pocit uspokojení, protože v seznamu stále ještě mnoho položek zbývá. Takže místo dobrého pocitu ze sama sebe se dostaví nespokojenost, stres nebo přehlcení.

Také vznikají aplikace, které tento problém řeší tím, že dané akce mají konkrétní datum a časovou délku, což uživateli pomáhá uvědomit si, jak dlouho konkrétní akce trvají, a dokáže si pak den naplánovat realisticky. Tyto aplikace ale pak často neumožňují si dané úkoly odklikávat a fungují spíše jako kalendář. I když uživateli umožňují zakládat si opakované akce, problém je v tom, že přidané položky zase slouží spíše jako přehled toho, co má uživatel před sebou a nemotivuje uživatele k tomu, aby si dané návyky tvořil a zlepšoval se v nich. Také většinou neposkytují žádné statistiky a zpětnou vazbu opakovaných akcí.

Poté je tu celá řada sledovačů návyků. Tyto aplikace se velmi hodí na budování a udržování zdravých a prospěšných návyků, případně zbavování se těch špatných. Většinou ale pak neumožňují si zapisovat akce, které se neopakují a vést si kalendář s důležitými termíny.

Proto jsem se rozhodl vytvořit tuto aplikaci. Netvrdím, že je moje aplikace úplně jedinečná, nicméně poskytuje spojení již zmíněných dvou disciplín do jedné uživatelsky přívětivé aplikace, která adresuje tyto problémy a je navržena tak, že vychází z ověřených metod efektivního plnění úkolů a tvoření návyků.

1.3 Přístup mé aplikace k problematice

V této kapitole je popsáno, jak jsem ke zmíněným problémům přistupoval. Aby bylo zajištěno, že si uživatel nenaplánuje tolik úkolů, že je nebude moci stihnout, nebo si jich naopak naplánuje zbytečně málo, k plánování se v mé aplikaci používá časová osa. Když uživatel přidává novou akci, musí zadat datum, čas začátku úkolu a dobu, jak dlouho se chce danou aktivitou zabývat. Už jen to, kdy si člověk přesně stanoví datum a čas, zvyšuje pravděpodobnost, že člověk úkol opravdu splní. *„Podněty, které mohou spustit návyk, mají širokou škálu forem, ale dvě nejčastější jsou čas a místo. Implementační záměry využívají oba tyto podněty. Obecně řečeno, formát pro vytvoření implementačního záměru je: “Udělám [TOTO] dne [DATUM] na [MÍSTĚ] v [ČAS].”[2]*

To, že uživatel zadává dobu trvání, pomáhá zachovat si realistickou představu a zároveň umožňuje využít čas efektivně. Vždy, když uživatel přidá novou akci, všechny existující akce v rámci dne se seřadí. Vizuální výška daných úkolů se odvíjí od toho, jak je úkol časově dlouhý. To znamená, že časově delší akce bude vyšší než časově kratší akce. To je takzvané blokování času. *„Blokování času je technika, kdy přiřazuješ takzvaný časový blok konkrétním úkolům nebo aktivitám. Na rozdíl od to-do listů se při blokování času musíš rozhodnout předem, kolik času nad úkolem strávíš. To ti pomůže lépe se soustředit po dobu práce a zaměřit se jen na daný úkol.”[3]*

Uživatel si může jednak přidat jednorázovou akci a jednak návyk, tedy opakovanou akci. U návyku musí zvolit začáteční datum návyku a frekvenci, jak se bude daný návyk opakovat. Úkoly můžeme označit za dokončené a u návyků si můžeme každý den zapisovat pokrok. Kdyby se opakované akce jen plnily nebo neplnily, vznikl by problém, kdyby uživatel například četl deset minut z plánovaných patnácti, návyk by byl nesplněný. V mé aplikaci si ale může zapsat, kolik daného návyku splnil, což má za výsledek to, že není demotivovaný, když místo patnácti minut čte deset. Nebo naopak čte jeden den knihu dvacet minut z cílových patnácti a do aplikace si může zapsat i víc, než je cílová hodnota. To má za účel motivovat uživatele a pomáhá to k vytvoření statistik, které co nejvíce odpovídají realitě.

Uživateli je umožněno zvyšovat si cíl návyku v průběhu času. To znamená, že cíl je například: číst každý den třicet minut. Uživatel ale začne pouze s jednou minutou a postupně si zvyšuje cíl, když už mu současný připadá snazší. Pokud si formujeme nový návyk, doporučuje se používat takzvané pravidlo dvou minut. „*Pravidlo dvou minut říká, že jakýkoli nový návyk by měl být zmenšen na akci, která trvá méně než dvě minuty. Tento přístup činí návyky méně skličujícími a zvyšuje pravděpodobnost, že začnete. To může vypadat následovně: Číst před spaním každou noc se stává Přečíst jednu stránku Dělat třicet minut jógy se stává Vytáhnout podložku na jógu. Tyto malé akce slouží jako "brána k návykům", které vedou k většímu chování, které chcete přijmout. Jakmile začnete, je snazší pokračovat. Klíčem je učinit návyky co nejjednoduššími na začátek, což umožňuje setrvačnosti vás posunout vpřed.*“[4] Uživateli je umožněno začít návyk zlehka a postupně zvyšovat cílovou hodnotu. Dále se také vytváří statistika, díky které uživatel může sledovat, jak se blíží nebo neblíží svému cíli.

2 Použité technologie

Při vývoji mobilní aplikace pro Android jsem se rozhodoval mezi programovacími jazyky Java a Kotlin. Oba jsou nativní pro tuto platformu, ale zvolil jsem Javu, protože ji již dobře znám a mám s ní zkušenosti. Uživatelské rozhraní aplikace je napsáno v XML, což je standardní formát pro definování vizuálních prvků v Android aplikacích.

Jako vývojové prostředí jsem vybral Android Studio, které je speciálně navrženo pro vývoj mobilních aplikací v Javě. Nabízí předpřipravenou strukturu projektu, nástroje pro ladění a integraci s různými službami, což výrazně usnadňuje práci. Součástí tohoto prostředí je také Gradle, moderní systém pro správu sestavení a závislostí. Gradle umožňuje efektivní správu knihoven, optimalizaci sestavování aplikace a zjednodušení celého vývojového procesu. Díky němu lze snadno přidávat externí knihovny, nastavovat různé build konfigurace a využívat pokročilé funkce, jako je cacheování sestavení nebo podpora multimodulové architektury.

Pro ukládání dat jsem použil dvě odlišné databáze. První je SQLite, což je vestavěná lokální databáze v Androidu. Použil jsem ji především pro ukládání úkolů a uživatelských preferencí, jako je volba tmavého nebo světlého režimu či nastavení notifikací. Dále jsem implementoval Firebase, která se stará o autentizaci uživatelů prostřednictvím e-mailu a Google účtu. Ve Firebase jsou uloženy návyky uživatelů, což umožňuje synchronizaci mezi zařízeními. Tímto způsobem uživatel o svá data nepřijde, ani když si pořídí nový telefon – stačí se pouze přihlásit pod stejným účtem a všechna jeho data se automaticky načtou.

3 Architektura aplikace

Tato kapitola pojednává o tom, jak je aplikace sestavená. Věnuje se konfiguračnímu nastavení, prezentační vrstvě, vrstvě aplikační logiky a struktuře databáze. Jsou zde zmíněny klíčové prvky aplikace.

3.1 Konfigurační nastavení aplikace

Gradle je buildovací nástroj používaný v Android vývoji k automatizaci procesu sestavení aplikace. Umožňuje spravovat závislosti, optimalizovat kód a zajišťovat kompatibilitu mezi různými verzemi Androidu. Gradle používá soubory s příponou `.gradle.kts` k definování konfigurace projektu. V mé aplikaci se nacházejí následující Gradle soubory. **build.gradle (Project-level)** se nachází v kořenové složce projektu a obsahuje globální konfigurace pro celý projekt. Definuje správu pluginů a repozitářů, odkud se stahují závislosti. **build.gradle (Module-level)** se nachází v modulu DayPlanner. Pokud by bylo modulů více, každý by měl svůj vlastní Gradle soubor. Zde jsou definována specifická nastavení aplikace, jako je verze SDK, závislosti a optimalizace. **settings.gradle** určuje, jaké moduly jsou součástí projektu, a definuje správu závislostí a repozitářů pro celý projekt. Tyto soubory dohromady řídí způsob, jakým se aplikace sestavuje.

Zde jsou uvedeny důležité závislosti mého projektu. **Firestore BoM** používá se k automatickému správnému verzování závislostí souvisejících s Firestore. Díky tomu není nutné u každého Firestore produktu uvádět konkrétní verzi. **Firestore Analytics** poskytuje analytická data o používání aplikace, jako je chování uživatelů a interakce s aplikací. **Firestore Realtime Database** je cloudová databáze umožňující synchronizaci dat v reálném čase mezi zařízeními. **Firestore Authentication** slouží k autentizaci uživatelů v aplikaci, například pomocí e-mailu, hesla nebo přihlášení přes Google. **Google Play Services Auth** poskytuje funkce pro autentizaci uživatele přes Google účet. **MPAndroidChart** je knihovna pro vykreslování grafů a vizualizaci dat v aplikaci.

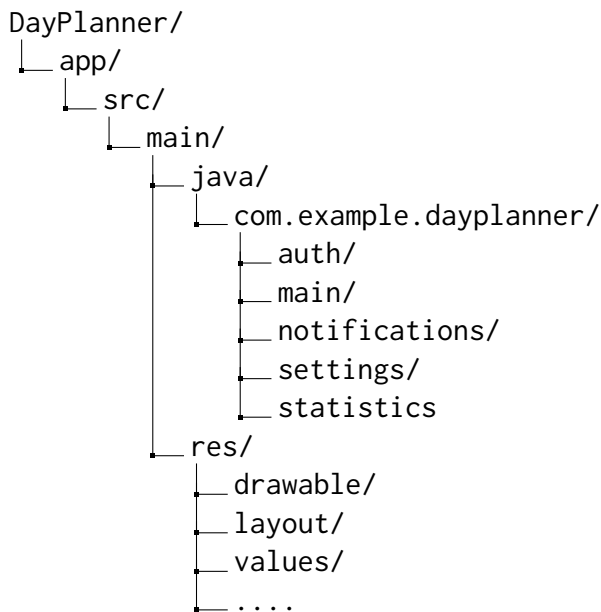
```
1 dependencies {  
2     implementation(platform("com.google.firebase:firebase-bom:33.8.0"))  
3     implementation("com.google.firebase:firebase-analytics")  
4     implementation("com.google.firebase:firebase-database")  
5     implementation("com.google.firebase:firebase-auth:23.1.0")  
6     implementation("com.google.android.gms:play-services-auth:21.3.0")  
7     implementation("com.github.bumptech.glide:glide:4.16.0")  
8     implementation("com.github.PhilJay:MPAndroidChart:v3.1.0")  
9 }
```

Zdrojový kód 3.1: build.gradle

V adresáři DayPlanner/app je soubor google-services.json, který obsahuje všechna důležitá data a konfigurace, aby se uživatel mohl úspěšně přihlásit přes Google účet.

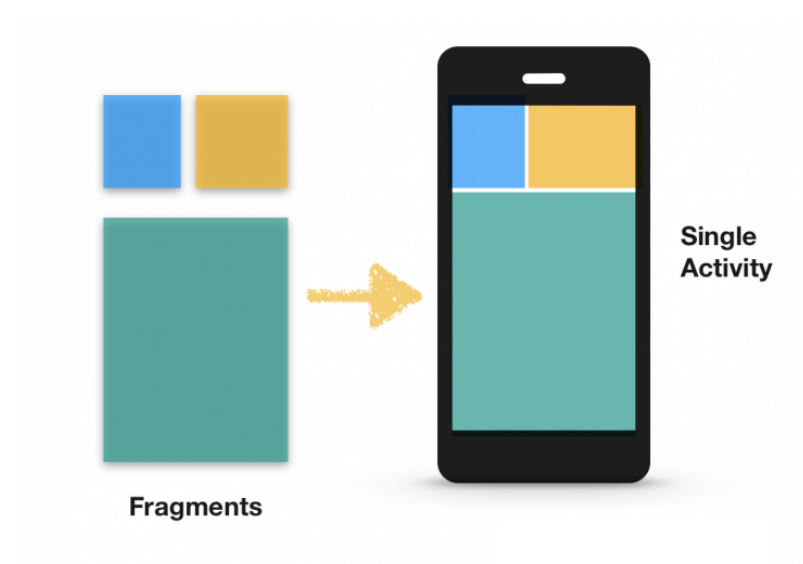
3.2 Adresářová struktura

Nyní bude krátce zmíněno, jak je aplikace rozdělená do adresářů. V adresáři `java/com.example.dayplanner` se nachází backend aplikace a v adresáři `res/` se nachází frontend aplikace. Jelikož bylo v backendu poměrně velké množství tříd, rozhodl jsem třídy rozčlenit do jednotlivých balíčků(package) pojmenovaných podle funkcionality.



Package `auth` se stará o autentikaci uživatele. Uživatel se může přihlásit přes Google nebo přes vlastní email. V případě, že si uživatel zakládá účet přes email, přijde mu na email odkaz na ověření. Jedině pokud uživatel email ověří, tak může pokračovat dále do aplikace a založit si účet. Package `main` se člení dál. Nachází se v něm funkcionality pro časovou osu, vybírání dne z kalendáře, modely pro jednorázové a opakované akce a třída `main`. Package `notifications` se stará o vytváření a posílání notifikací. V package `settings` se nachází logika pro ukládání nastavení uživatele a package `statistics` se stará o počítání statistik a vytváření přehledného UI.

Jak již bylo řečeno adresář `res/` se stará o design aplikace. V podadresáři `drawable` se nachází xml soubory, které se pak používají v `layout`. Jedná se například o různé ikony, obrázky, pozadí nebo tvary. V podadresáři `layout` se nachází xml design pro aktivity a fragmenty. Aktivita se zpravidla skládá z více menších oddělitelných částí neboli z fragmentů. To slouží k znovupoužitelnosti a přehlednosti kódu. Pro lepší ilustraci aktivity a fragmentů je dáno k dispozici schéma níže(viz obr. 3.1).



Zdroj: <https://medium.com/ibtech/activity-vs-fragment-703c749c1bbd>

Obrázek 3.1: Activity & Fragments schema

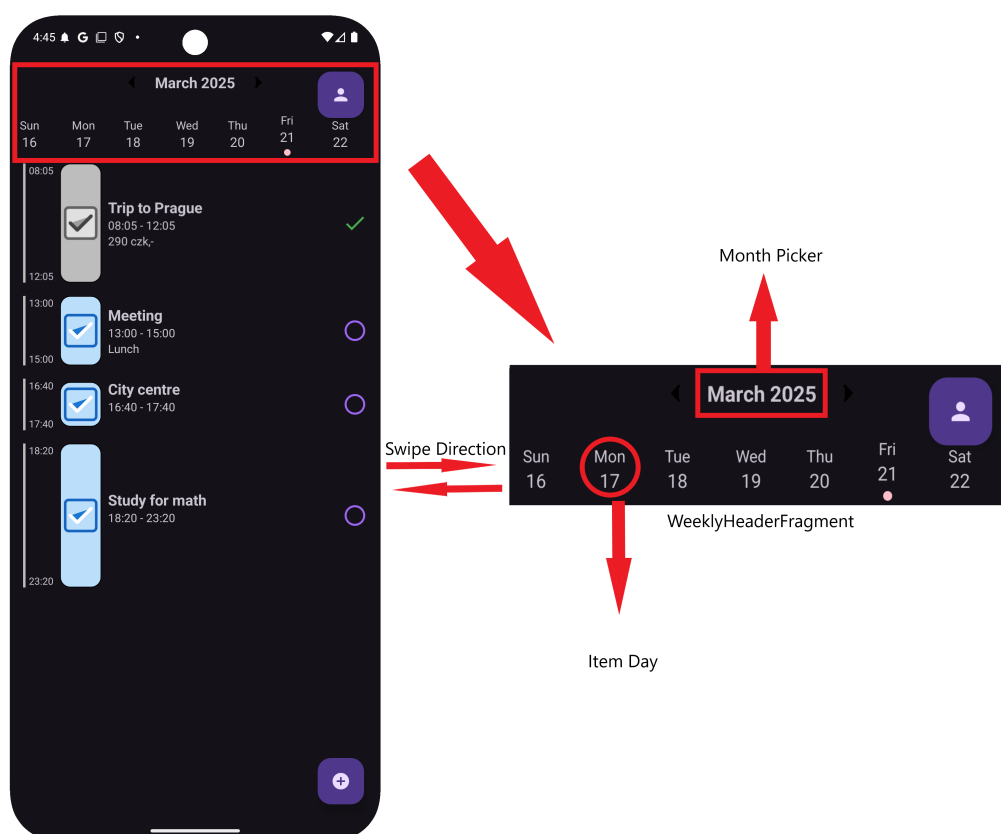
V adresáři values se nachází hodnoty. Nachází se zde hodnoty textových řetězců, polí, barev a podobně. Je zde uložen design například pro světlý a tmavý režim.

3.3 Prezentační vrstva (Frontend)

Tato kapitola se věnuje aktivitám, fragmentům a důležitým designovým prvkům aplikace.

3.3.1 Main activity

Třída main activity se skládá z několika komponent. Důležité jsou dva fragmenty. Za prvé WeeklyHeaderFragment, který je zodpovědný za ukazování dnů po týdnech, aktuální měsíc a rok a možnost vybrat si konkrétní den pomocí kalendáře. Za druhé TimelineFragment, který ukazuje časovou osu s jednorázovými a opakovanými akcemi.

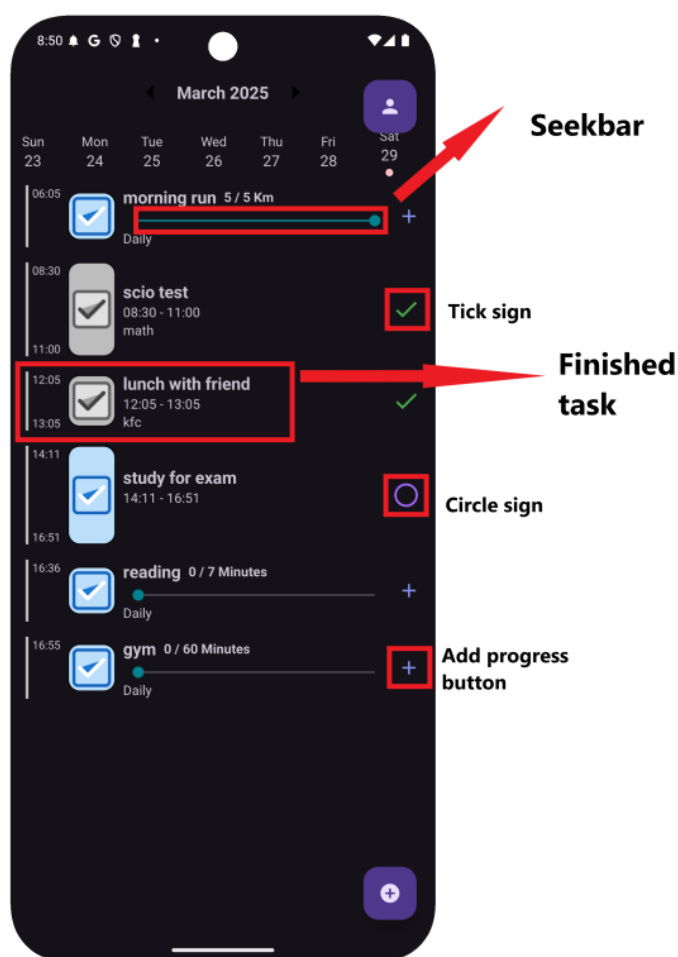


Obrázek 3.2: WeeklyHeaderFragment

WeeklyHeaderFragment se stará o ukazování aktuálního týdne po dnech. Umožňuje uživateli pohybovat se mezi dny. Uživatel může táhnutím prstu (Swipe Direction viz obr. 4.3) posouvat týdny dopředu nebo dozadu. Pokud uživatel nechce swipovat po týdnech, může kliknout na Month Picker (viz obr. 4.3) a vybrat z kalendáře konkrétní den. Když uživatel klikne na Item Day (viz obr. 4.3), vybere tím konkrétní den a je poslán

signál pomocí interface do timeLineFragment. Tento fragment získá z databáze potřebná data a zobrazí je v časové ose.

TimelineFragment získá potřebná data z databáze, seřadí jednotlivé položky podle času a upraví jejich výšku podle doby trvání. Následně je postupně pod sebou zobrazí. Jak lze vidět, jednorázové akce se dají pomocí tlačítka(circle sign) označit za dokončené. Poté změní barvu na černobílou a zobrazí se u nich fajfka indikující(finished task, tick sign), že je úkol hotový. U opakovaných akcí může uživatel pohybovat seekbar, aby zadal hodnotu. Pokud je návyk například sníst denně 3000 kalorií, stává se seekbar poněkud nešikovný a nepřesný, protože počítá hodnotu podle toho, jak daleko uživatel potáhl prstem. Proto si pomocí tlačítka "+"(add progress button) může hodnotu zadat manuálně. Může buď celou hodnotu nastavit znovu nebo ke stávající hodnotě přidat pokrok.



Obrázek 3.3: Timeline Fragment

3.3.2 Statistiky

Statistiky v aplikaci mají za cíl co nejintuitivněji a nejprehledněji zobrazit uživateli jeho pokrok, aby snadno zjistil, kde má prostor pro zlepšení, a zároveň se mohl radovat z dosažených úspěchů. Na základě vlastních zkušeností považuji za velmi užitečné na konci každého měsíce vyhodnotit plnění návyků, a proto jsou statistiky vždy rozděleny podle jednotlivých měsíců. Uživatel si mezi nimi může snadno přepínat pomocí nástroje Month Picker(viz obr. 3.4).

Nejdůležitější součástí statistik je ukazatel celkové úspěšnosti v daném měsíci, který se zobrazuje v sekci Overall Month Progress(viz obr. 3.4). Tento ukazatel může zobrazovat průměrnou úspěšnost všech sledovaných návyků dohromady nebo jednoho konkrétního, který si uživatel vybere ze seznamu. Výpočet celkové úspěšnosti vychází z průměru všech dní v měsíci. Pokud například jeden den splní návyky jen na padesát procent a druhý den na sto procent, celková úspěšnost bude sedmdesát pět procent. Kolem procentuálního ukazatele se nachází barevný kruh, který vizuálně odráží úspěšnost v daném měsíci. Barevné spektrum přechází od červené, která značí nízkou úspěšnost, přes oranžovou a žlutou až po zelenou, která indikuje vysokou úspěšnost. Pokud uživatel za daný měsíc nezaznamenal žádné údaje, kruh zůstává šedý.

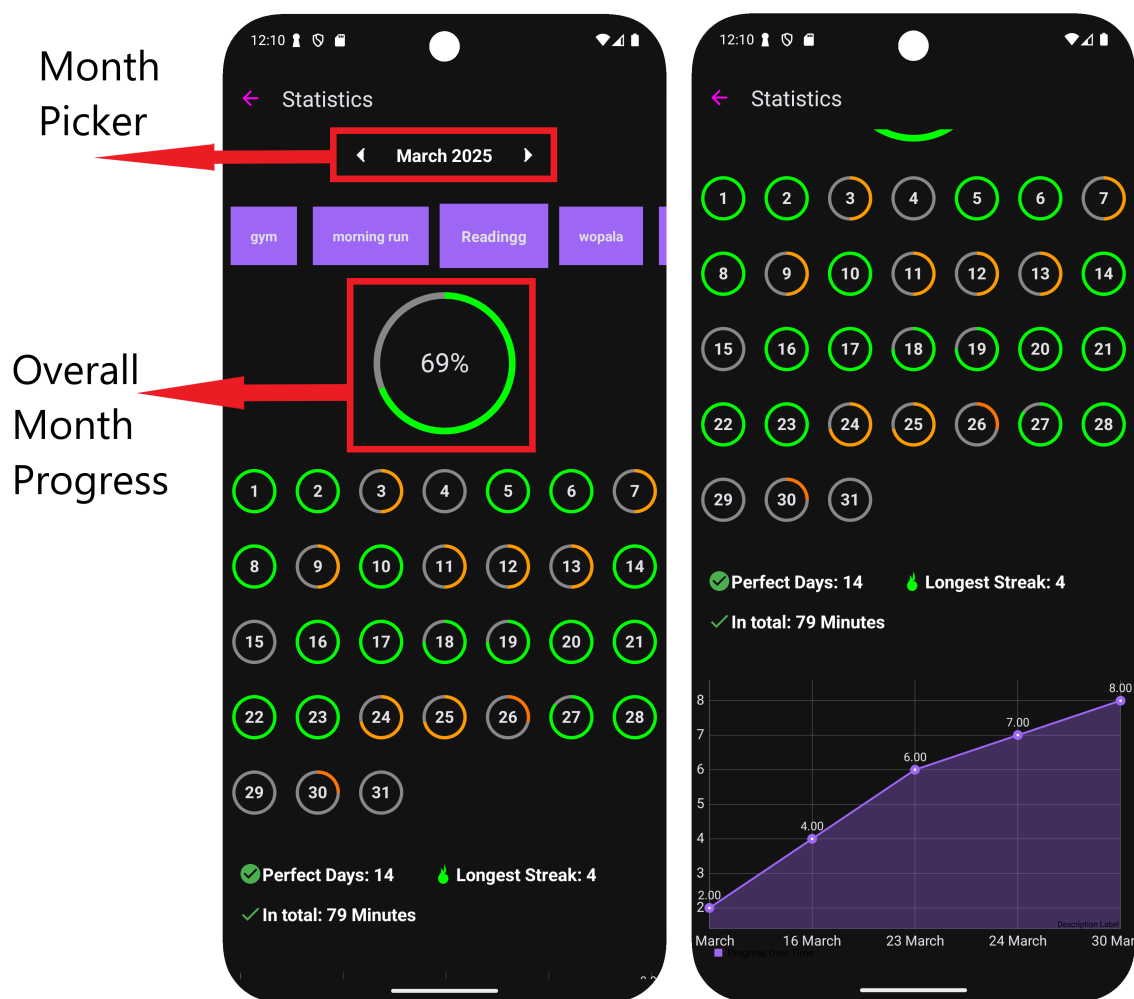
Pod tímto ukazatelem se nachází kalendářová mapa, která rozděluje měsíc na jednotlivé dny. Každý den je zobrazen samostatně a jeho úspěšnost je vyjádřena pomocí kruhového indikátoru. Díky tomu má uživatel detailní přehled o tom, jak se mu v průběhu měsíce dařilo.

Kromě vizualizace jednotlivých dní aplikace zobrazuje i dvě další zásadní statistiky. První z nich je počet perfektních dní, tedy takových dní, kdy uživatel splnil všechny své návyky na sto procent. Druhou důležitou statistikou je nejdelší nepřerušená sekvence perfektních dní, označovaná jako "streak". Tato hodnota ukazuje, jak dlouhou dobu se uživateli dařilo udržet si pravidelný návykový režim bez přerušení.

Pokud si uživatel zobrazí statistiky pro konkrétní návyk, získá ještě jednu doplňkovou informaci. V takovém případě se mu zobrazí také celkový počet splnění návyku za daný měsíc. Tato hodnota může být vyjádřena v různých jednotkách, například v minutách strávených cvičením, v celkové vzdálenosti v kilometrech nebo ve spálených kaloriích.

Pokud aplikace zaznamená, že v průběhu měsíce došlo ke změně cílové hodnoty návyku, zobrazí se uživateli interaktivní graf. Tento graf umožňuje sledovat, jak se cílová hodnota měnila v průběhu měsíce, a pomáhá lépe pochopit, jak se uživatelův výkon a cíle vyvíjely.

Celkově jsou statistiky navrženy tak, aby uživateli poskytly nejen detailní přehled o jeho pokroku, ale také ho motivovaly k dalšímu zlepšování a pomohly mu lépe pochopit jeho vlastní návyky.



Obrázek 3.4: Statistics

3.3.3 Tmavý a světlý režim

Tato aplikace podporuje světlý a tmavý režim pomocí Material 3 a SQLite databáze pro uchování preferencí uživatele. Hodnoty motivů jsou definovány ve `values/themes.xml` a `values-night/themes.xml`. `ThemePreferenceHelper.java` se stará o ukládání uživatelem preferovaného motivu do lokálního úložiště a pochopitelně zajišťuje i získání této hodnoty, aby s ní aplikace mohla pracovat.

Konkrétněji to vypadá následovně. Barvy pro motivy jsou definované stejně, jediné co se liší jsou hodnoty barev (Zdrojový kód 3.2, Zdrojový kód 3.3).

```
1 <style name="Base.Theme.DayPlanner" parent="Theme.Material3.  
   DayNight.NoActionBar">  
2     <item name="colorPrimary">@color/lavender</item>  
3     <item name="colorOnPrimary">@color/colorOnPrimaryLight</item  
   >  
4     <item name="colorPrimaryVariant">@color/lavender</item>  
5     <item name="colorSecondary">@color/colorSecondaryLight</item  
   >  
6     ...  
7 </style>
```

Zdrojový kód 3.2: Hodnoty světlého motivu

```
1 <style name="Base.Theme.DayPlanner" parent="Theme.Material3.  
   DayNight.NoActionBar">  
2     <item name="colorPrimary">@color/darkGrey</item>  
3     <item name="colorOnPrimary">@color/black</item>  
4     <item name="colorPrimaryVariant">@color/darkGrey</item>  
5     <item name="colorSecondary">@color/colorSecondaryDark</item>  
6     ...  
7 </style>
```

Zdrojový kód 3.3: Hodnoty tmavého motivu

Hodnoty barev, na které odkazuje 'themes.xml', se nacházejí v 'colors.xml' (Zdrojový kód 3.4).

```
1 <color name="colorPrimaryLight">#CE9932</color>
2 <color name="colorPrimaryDark">#2F0385</color>
```

Zdrojový kód 3.4: Hodnoty barev

O správné nastavování motivu a ukládání preference do databáze se stará již zmíněná třída ThemePreferencesHelper.java. Hlavní metody jsou setThemePreference (Zdrojový kód 3.5), která uloží motiv do lokální databáze, aby se uživateli po každém opětovném zapnutí zobrazoval jím zvolený motiv.

```
1 public void setThemePreference(String theme) {
2     SQLiteDatabase db = this.getWritableDatabase();
3     db.execSQL("UPDATE preferences SET theme = ? WHERE id = 1", new
4     String[]{ theme});
5     db.close();
6 }
```

Zdrojový kód 3.5: setThemePreference

Když se aplikace inicializuje, tedy uživatel ji zapne, je potřeba hodnotu z databáze získat. K tomu slouží metoda getThemePreference (Zdrojový kód 3.6). Když hodnotu získám, pak mohu během inicializace aplikace ve třídě main nastavit barevné hodnoty podle uživatelem preferovaného motivu. Při startu aplikace se načte uložená preference a aplikuje se odpovídající motiv (Zdrojový kód 3.8).

```
1 public String getThemePreference() {
2     SQLiteDatabase db = this.getReadableDatabase();
3     String theme = "light"; // Default Value
4     Cursor cursor = db.rawQuery("SELECT theme FROM preferences WHERE id
5     = 1", null);
6     if (cursor.moveToFirst()) {
7         theme = cursor.getString(0);
8     }
9     cursor.close();
10    db.close();
11    return theme;
12 }
```

Zdrojový kód 3.6: getThemePreference

```

1 ThemePreferencesHelper dbHelper = new ThemePreferencesHelper(this);
2 if (dbHelper.getThemePreference().equals("dark")) {
3     AppCompatActivity.setDefaultNightMode(AppCompatActivity.
4         MODE_NIGHT_YES);
5 } else {
6     AppCompatActivity.setDefaultNightMode(AppCompatActivity.
7         MODE_NIGHT_NO);
8 }

```

Zdrojový kód 3.7: Nastavení motivu

Pokud uživatel změní režim v nastavení aplikace, preference se uloží a téma se přepne pomocí metod v themePreferenceHelper.

```

1 switchMode.setOnClickListener(new View.OnClickListener() {
2     @Override
3     public void onClick(View v) {
4         if (nightMode) {
5             AppCompatActivity.setDefaultNightMode(AppCompatActivity.
6                 MODE_NIGHT_NO);
7             themePreferencesHelper.setThemePreference("light");
8         } else {
9             AppCompatActivity.setDefaultNightMode(AppCompatActivity.
10                 MODE_NIGHT_YES);
11             themePreferencesHelper.setThemePreference("dark");
12         }
13         nightMode = !nightMode;
14     }
15 });

```

Zdrojový kód 3.8: Změnění motivu

Tento přístup dynamicky nastavuje hodnoty barev pro různé komponenty aplikace podle toho, co si uživatel vybral za motiv. Motiv se navíc ukládá do lokálního úložiště, takže kdykoliv se uživatel do aplikace vrátí, zobrazí se mu aplikace ve správném motivu.

3.4 Vrstva aplikační logiky (Backend)

Tato kapitola pojednává o klíčových funkcích, algoritmech a strukturách, které se v mé práci nacházejí.

3.4.1 Rozdělení logiky

Moje aplikace často využívá těchto tří komponent: Activity, Fragment a RecyclerView. Každá z těchto součástí má svůj specifický účel a společně tvoří strukturovaný a efektivní způsob práce s uživatelským rozhraním.

Activity je základní stavební jednotka aplikace, která představuje jednu obrazovku. Každá aplikace obsahuje alespoň jednu aktivitu, která je prvním vstupním bodem pro uživatele. Aktivita řídí celý proces zobrazení, reaguje na uživatelské akce a může obsahovat další komponenty, jako jsou fragmenty nebo seznamy dat.

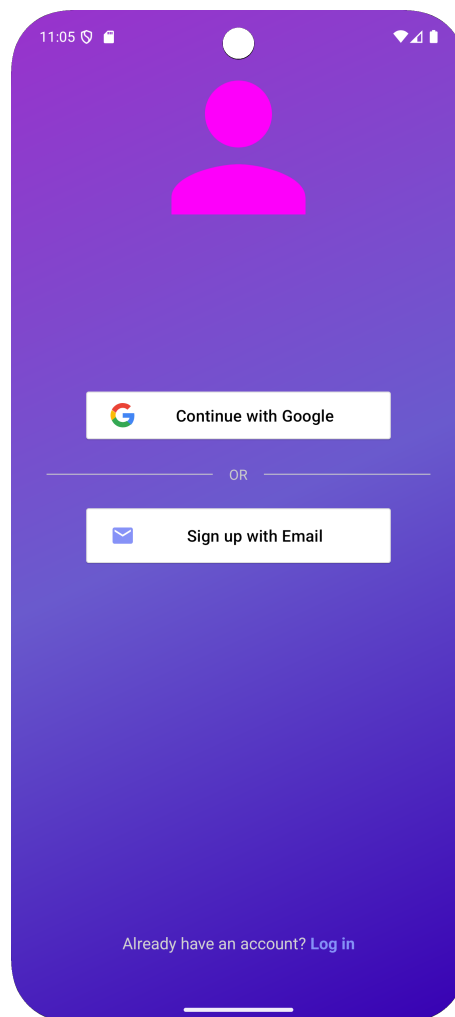
Fragment je menší část uživatelského rozhraní, která může být vložena do aktivity. Hlavní výhodou fragmentů je jejich flexibilita – umožňují dynamické změny v rozhraní bez nutnosti přepínání celých aktivit. Fragmenty lze kombinovat na jedné obrazovce (viz. kapitola frontend) a znovu použít v různých částech aplikace. Aktivita obvykle spravuje více fragmentů a mezi nimi přepíná podle potřeby.

RecyclerView je způsob, jak efektivně zobrazovat seznamy nebo mřížky dat. Používá adaptér, který propojuje data s jednotlivými položkami seznamu, a ViewHolder, který optimalizuje vykreslování tím, že znovu používá již vytvořené pohledy.

Typickým příkladem propojení těchto komponent je aplikace, kde hlavní aktivita obsahuje více fragmentů s přehledem dat, a tyto fragmenty využívají RecyclerView's pro zobrazení seznamu položek.

3.4.2 Autentizace a přihlášení uživatele

Uživatel se může přihlásit dvěma způsoby. Buď pomocí emailu a nebo může pokračovat přes Google účet. Authentication activity(viz obr. 3.5) je hlavní třída zajišťující přihlášení uživatele.



Obrázek 3.5: Authentication Activity

Níže je adresářová struktura balíčku auth.

```
auth/  
├── google/  
│   └── GoogleLoginFragment.java  
├── login/  
│   └── EmailLoginActivity.java  
├── signin/  
│   └── EmailSignInActivity.java  
└── AuthenticationActivity.java
```

Google přihlášení v aplikaci je implementováno ve třídě `GoogleLoginFragment`. Tento fragment se pak používá v authentication activity jako tlačítko, které má veškerou potřebnou funkcionalitu. Přihlášení probíhá přes `Firebase Authentication` a `Google Sign-In API`. Fragment inicializuje `GoogleSignInClient` (Zdrojový kód 3.9) s odpovídajícími možnostmi (`GoogleSignInOptions`), kde je vyžadován ID token a e-mail. Po kliknutí na tlačítko pro přihlášení se spustí přihlašovací intent přes `ActivityResultLauncher` (Zdrojový kód 3.10), který zachytí výsledek přihlášení. Pokud je přihlášení úspěšné, získá se účet Google a předá se metodě `firebaseAuthWithGoogle()`. Ta pomocí `GoogleAuthProvider` vytvoří přihlašovací pověření a provede autentizaci ve `Firebase` (Zdrojový kód 3.11). Při úspěchu se zobrazí potvrzení a aplikace přeměruje uživatele do hlavní aktivity.

```
1 GoogleSignInOptions googleSignInOptions = new GoogleSignInOptions.  
    Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)  
2      .requestIdToken(getString(R.string.client_id))  
3      .requestEmail()  
4      .build();  
5 googleSignInClient = GoogleSignIn.getClient(  
6     requireContext(), googleSignInOptions);
```

Zdrojový kód 3.9: `googleSignInOptions`

```
1 Intent signInIntent = googleSignInClient.getSignInIntent();  
2 activityResultLauncher.launch(signInIntent);
```

Zdrojový kód 3.10: `signInIntent`

```
1 AuthCredential credential = GoogleAuthProvider.getCredential(account.  
    getIdToken(), null);  
2 firebaseAuth.signInWithCredential(credential).addOnCompleteListener(  
    task -> {  
3      if (task.isSuccessful()) {  
4        startActivity(new Intent(getActivity(), MainActivity.class));  
5        requireActivity().finish();  
6      }  
7    });
```

Zdrojový kód 3.11: `GoogleAuthProvider`

Uživatel se může zaregistrovat nebo přihlásit pomocí svého emailu. EmailSignIn activity slouží k registraci nových uživatelů a EmailLoginActivity umožňuje přihlášení existujících uživatelů. Pro registraci uživatel zadává svůj email a heslo. Heslo je pro bezpečnost zadáváno dvakrát, obě hesla se musí shodovat a pole nesmí být prázdná.

```
1 firebaseAuth.createUserWithEmailAndPassword(email, password).
   addOnCompleteListener(new OnCompleteListener<AuthResult>() {
2       @Override
3       public void onComplete(@NonNull Task<AuthResult> task) {
4           progressDialog.hide();
5           if (task.isSuccessful()) {
6               handleEmailVerification(firebaseAuth,
7               "User registered successfully, Please verify your email
               address",
8               "Something went wrong");
9           } else {
10               Toast.makeText(EmailSignInActivity.this, "Something went
               wrong", Toast.LENGTH_SHORT).show();
11               Log.d("USR", "EXCEPTION: " + task.getException().toString()
               );
12           }
13           Intent intent = new Intent(EmailSignInActivity.this,
               EmailLoginActivity.class);
14           startActivity(intent);
15       }
16 });
```

Zdrojový kód 3.12: Create New User

Firebase metoda `createUserWithEmailAndPassword` (Zdrojový kód 3.12) provede registraci. Pokud je registrace úspěšná, je uživatel přesměrován pomocí intent na `EmailLoginActivity` a zároveň je mu na emailovou adresu poslán odkaz pro ověření emailu. Pokud uživatel email neověří kliknutím na odkaz, nemůže se do aplikace přihlásit.

```
1 public void handleEmailVerification(FirebaseAuth firebaseAuth, String
   message_on_successful, String message_on_not_successful) {
2     if (firebaseAuth.getCurrentUser() != null) {
3         firebaseAuth.getCurrentUser().sendEmailVerification().
         addOnCompleteListener(new OnCompleteListener<Void>() {
4             @Override
```

```

5      public void onComplete(@NonNull Task<Void> task) {
6          if(task.isSuccessful()) {
7              Toast.makeText(EmailSignInActivity.this,
message_on_successful, Toast.LENGTH_SHORT).show();
8              editEmail.setText("");
9              editPassword.setText("");
10             editConfirmPassword.setText("");
11         } else {
12             Toast.makeText(EmailSignInActivity.this,
message_on_not_successful, Toast.LENGTH_SHORT).show();
13         }
14     }
15 });
16 }
17 }

```

Zdrojový kód 3.13: handleEmailVerification

Funkce handleEmailVerification(Zdrojový kód 3.13) pošle prostřednictvím firebaseAuth email uživateli a kontroluje výsledek. Pokud byl email úspěšně odeslán, zobrazí se uživateli pozitivní zpráva. V opačném případě se uživateli zobrazí zpráva o neúspěchu. Pokud vše proběhlo v pořádku, pokračuje přihlášení v EmailLoginActivity. Zde se uživatel může přihlásit. Pokud pole pro heslo a email nejsou prázdná, podívá se funkce, zda byl email ověřen pomocí isEmailVerified() a zkontroluje také, zda je heslo shodné s heslem ve Firebase. Pokud alespoň jedna podmínka neplatí, přihlášení se nezdaří. Pokud vše proběhne bez komplikací je uživatel přesměrován do MainActivity.

```

1 mAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener
  (new OnCompleteListener<AuthResult>() {
2     @Override
3     public void onComplete(@NonNull Task<AuthResult> task) {
4         progressDialog.hide();
5         if(task.isSuccessful()) {
6             if(mAuth.getCurrentUser().isEmailVerified()) {
7                 Intent intent = new Intent(EmailLoginActivity.this,
MainActivity.class);
8                 startActivity(intent);
9             } else {
10                Toast.makeText(EmailLoginActivity.this, "Please verify
your email address", Toast.LENGTH_SHORT).show();
11            }
12        } else {
13            Toast.makeText(EmailLoginActivity.this, task.getException()
.getMessage(), Toast.LENGTH_SHORT).show();
14        }
15    }
16 });

```

Zdrojový kód 3.14: SignInWithEmailAndPassword

Může se stát, že email nemusí přijít nebo ho uživatel například omylem smaže. Proto je v EmailLoginActivity tlačítko, které po kliknutí pošle verifikaci emailu znovu(Zdrojový kód 3.15).

```

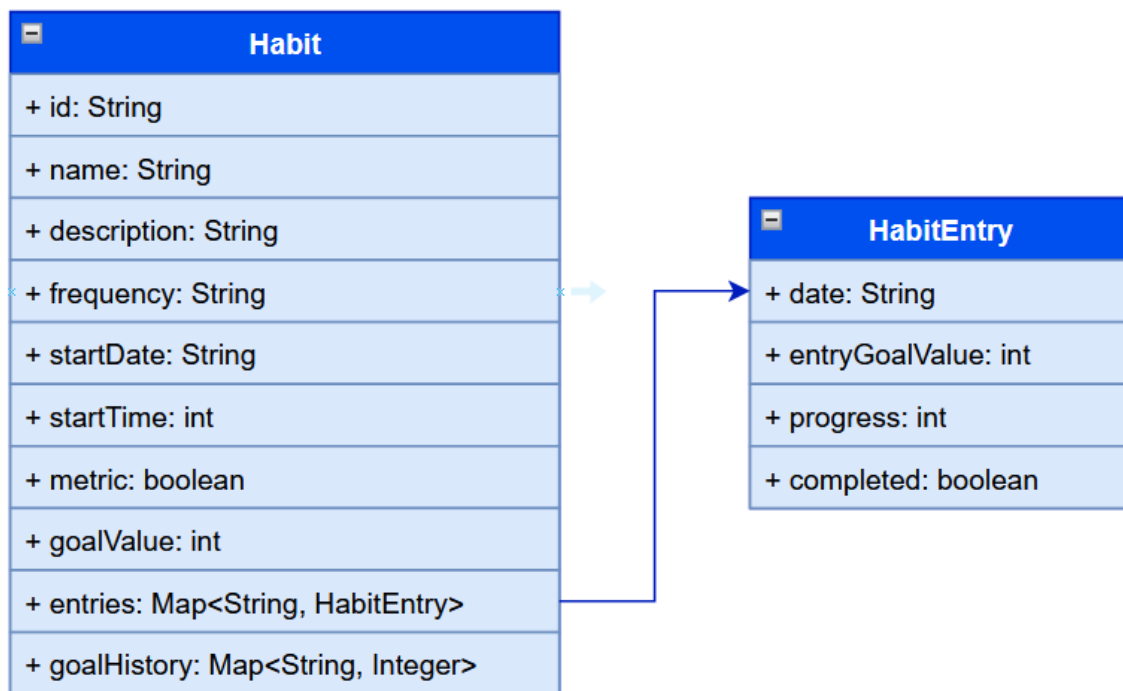
1 sendVerificationLink.setOnClickListener(new View.OnClickListener() {
2     @Override
3     public void onClick(View view) {
4         if (mAuth.getCurrentUser() != null) {
5             sendVerificationEmail(mAuth.getCurrentUser());
6         } else {
7             // Handles blank fields and parsing and non existing users
8             Whole code is in the actual project
9         }
10    }
11 });

```

Zdrojový kód 3.15: Send Email Verification

3.4.3 Habit

Model návyku (Habit) je třída, která spravuje všechny podstatné informace o individuálním návyku v aplikaci. Je navržena pro ukládání dat do Firebase Realtime Database. Má například konstruktor bez argumentů, který je vyžadován pro práci s Firebase.



Obrázek 3.6: Habit Model

Každý návyk obsahuje řadu atributů, které definují jeho charakteristiku a průběh. Patří mezi ně jedinečné identifikační číslo, název, popis, frekvence provádění, datum zahájení, čas zahájení, měřítko a cílová hodnota.

Další součástí modelu jsou dvě mapy: entries a goalHistory. Mapa entries uchovává záznamy o jednotlivých dnech (HabitEntry), kde každý záznam reprezentuje postup a stav návyku v konkrétní den. Mapa goalHistory pak uchovává historii změn cílových hodnot pro různé datumy.

Důležitou metodou je isHabitVisibleOnDate(), která určuje, zda má být návyk zobrazen v určitém dni. Metoda pracuje s různými frekvencemi:

```

1 public boolean isHabitVisibleOnDate(String dateToCheck) {
2     // If check date is before the habit start date, return false
3     if (checkDate.compareTo(start) < 0) {
4         Log.d("Habit", "Habit start date is after the check
5         date: " + dateToCheck + " > " + start.getTime());
6         return false;
7     }
8
9     switch (frequency.toLowerCase()) {
10         case "daily":
11             return true;
12         case "weekly":
13             return start.get(Calendar.DAY_OF_WEEK) == checkDate.get(
14             Calendar.DAY_OF_WEEK);
15         case "custom":
16             String customPattern = frequency.substring(7);
17             int dayOfWeekIndex = (checkDate.get(Calendar.DAY_OF_WEEK) +
18             5) % 7;
19             boolean isVisibleCustom = customPattern.charAt(
20             dayOfWeekIndex) != '-';
21             return isVisibleCustom;
22         }
23     }
24 }

```

Zdrojový kód 3.16: isHabitVisibleOnDate

Jak lze vidět uživatel si může vytvořit návyk i z libovolnou frekvencí. Funguje to tak, že si uživatel zaklikne dny v týdnu (pondělí - neděle), ve kterých chce návyk plnit. Když například chce návyk dělat každé pondělí a čtvrtek, aplikace uloží tuto informaci následovně: "M-T—", kde písmenka znamenají dny, kdy se návyk má objevovat a pomlčky indikují jeho absenci.

Metoda `getGoalValueForDate()` umožňuje dynamicky zjistit cílovou hodnotu pro konkrétní datum, přičemž preferuje specifické hodnoty z `goalHistory` před výchozí hodnotou návyku. Funguje to tak, že když vytvářím nové záznamy, program se podívá do jakého časového intervalu patří a podle toho nastaví cílovou hodnotu.

Každý návyk má také své vlastní položky (`HabitEntry`), které reprezentují denní progress. Třída `HabitEntry` obsahuje datum, cílovou hodnotu pro daný den, aktuální postup a příz-

nak dokončení. Metoda `updateProgress()` v této třídě automaticky spravuje postup a stanovuje, zda je úkol splněn.

V kontextu ukládání a správy dat spolupracuje `GoalManager` s modelem návyku. Poskytuje metody pro aktualizaci cílových hodnot, ukládání historie cílů a správu budoucích záznamů. Například, když uživatel změní cílovou hodnotu, musím tuto hodnotu změnit i pro všechny budoucí existující záznamy.

Celý model je navržen s ohledem na flexibilitu a dynamické sledování postupu uživatele při plnění jeho návyků.

3.4.4 Task

Třída `Task`, umístěná v balíčku `com.example.dayplanner.main.tasks`, představuje model jednorázového úkolu. Tento model slouží k uchování a správě informací o jednotlivých úkolech, které uživatel zadává do aplikace. Každý úkol obsahuje specifické atributy, které umožňují jeho efektivní načítání, zpracování a vizualizaci v rámci časové osy.

Struktura třídy zahrnuje identifikátor úkolu (`taskId`), název (`taskTitle`), volitelný popis (`taskDescription`), datum přiřazení (`taskDate`), čas začátku (`taskStartTime`), délku trvání (`taskLength`) a stav dokončení (`isCompleted`). Datum je uloženo ve formátu `'ddMMyyyy'`, což umožňuje snadné vyhledávání a třídění úkolů v databázi. Čas začátku a délka trvání jsou nezbytné parametry pro správné umístění a vizuální reprezentaci úkolu na časové ose.

Task
+ taskId: String
+ taskTitle: String
+ taskDescription: String
+ taskDate: String
+ taskStartTime: String
+ taskLength: int
+ isCompleted: boolean

Obrázek 3.7: Task Model

3.4.5 Časová osa

Časová osa je důležitým prvkem aplikace DayPlanner, který uživatelům umožňuje lépe si představit svůj den a efektivně plánovat pomocí blokového plánování.

Implementace časové osy zahrnuje načítání, zpracování a vizualizaci denních aktivit. Data pocházejí ze dvou zdrojů – jednorázové úkoly jsou uloženy v lokální SQLite databázi, zatímco pravidelné návyky se spravují v cloudovém úložišti Firebase Realtime Database. Proces začíná výběrem konkrétního data, kdy TimelineFragment spouští paralelní načítání úkolů a návyků. Metoda `fetchTasks()` získává jednorázové úkoly z lokální databáze, zatímco `fetchHabits()` zpracovává data o návycích z Firebase.

Načítání návyků vyžaduje několik kroků. Základní informace, jako je název nebo datum začátku návyku, se získávají snadno. Klíčovou otázkou je ale určení aktuálního pokroku a cílové hodnoty pro vybraný den. Záznamy se vyhledávají podle data ve formátu `'ddMMyyyy'` (například 15032025 pro 15. března 2025). Pokud existuje odpovídající záznam, lze z něj jednoduše získat hodnoty `entryGoalValue` a `progress`. Problém nastává, pokud záznam pro daný den neexistuje – typicky když si uživatel zobrazí návyk v konkrétní den poprvé. V takovém případě je třeba záznam vytvořit. Hodnoty `date`, `completed` a `progress` se nastaví na výchozí hodnoty, ale pro `entryGoalValue` je nutné zjistit, jaká cílová hodnota platila v daném období. K tomu slouží atribut `GoalsHistory`, který umožňuje určit správnou hodnotu na základě časového intervalu.

Důležitou součástí procesu je také ověření, zda by měl být návyk v daný den vůbec zobrazen. To řeší metoda `isHabitVisible()`, která kontroluje začátek platnosti návyku a jeho frekvenci. Například pokud návyk začal 10. března 2025 a opakuje se týdně v pondělí, nemůže být zobrazen 8. března (protože tehdy ještě neexistoval) ani 14. března (protože ten den neodpovídá nastavené frekvenci).

Po načtení všech dat se položky seřadí podle času začátku pomocí `Collections.sort`. Vizualizace probíhá pomocí vlastního dynamického `TimelineLayoutManager`, který přizpůsobuje délku a umístění prvků podle časového rozvržení. Tento přístup umožňuje intuitivní a přehledné zobrazení aktivit, kde delší úkoly zaujmají větší prostor.

Celková architektura systému je navržena s důrazem na flexibilitu a snadnou uživatelskou interakci. Výsledkem je dynamická časová osa, která efektivně podporuje time management a poskytuje přehledný vizuální plán dne.

3.4.6 Statistiky

Důležitou funkcí aplikace DayPlanner jsou již zmíněné statistiky. Zde jsou shrnuty algoritmy a výpočty použité k vytvoření smysluplných a pro uživatele zajímavých statistik.

```
statistics/  
├─ CustomCircularPorgressBar.java  
├─ DailyProgress.java  
├─ HabitListAdapter.java  
├─ HabitProgressEntry.java  
├─ MonthlyProgressAdapter.java  
└─ StatisticsActivity.java
```

Třída `StatisticsActivity` zpracovává a analyzuje data týkající se sledování návyků. Jádrem implementace jsou algoritmy schopné transformovat surová data o denních aktivitách do vypovídajících statistických ukazatelů.

Aplikace musí nejprve získat data z databáze, následně s nimi provést několik výpočtů a nakonec je zobrazit v prezentační vrstvě. Problém je, že dat je poměrně hodně, obzvlášť když uživatel aplikaci již nějakou dobu používá a má více návyků. Proto jsem se rozhodl data cachovat. Vždy když uživatel klikne na další měsíc, musí se počítat statistiky pro všechny viditelné návyky v daný měsíc. Proto se funkce `xy` prvně podívá, zda nejsou nějaká data v mezipaměti. Pokud ano zavolám funkci, která cachnuté data poskytne frontendu. V opačném případě pomocí reference na návyky získám data z Firebase a zavolám metodu `processHabitData`, která se postará o všechny výpočty a posílání dat do frontendu.

```
1 private void fetchAllHabitData(String monthId) {  
2     if (monthDataCache.containsKey(monthId)) {  
3         updateUIWithCachedData(monthDataCache.get(monthId));  
4         return;  
5     }  
6  
7     habitsRef.addListenerForSingleValueEvent(new ValueEventListener() {  
8         @Override  
9         public void onDataChange(DataSnapshot dataSnapshot) {  
10             executor.execute(() -> {  
11                 processHabitData(dataSnapshot, monthId);  
12             });  
13         }  
14     });
```

Zdrojový kód 3.17: Data Cache

Pokud uživatel klikne na konkrétní návyk, nechci ukazovat data pro všechny návyky ale jen pro jeden specifický v daném měsíci. O to se stará metoda `processHabitForMonth`. Jelikož jsou výpočty pro získání statistik pro jeden návyk a celý měsíc podobné, využívají stejných menších funkcí. Tím nevzniká redundantní kód a vše je přehlednější a udržitelnější. Níže jsou popsány již zmíněné menší funkce.

Algoritmus pro výpočet nejdelšího sledu (streaku) představuje prvek hodnocení konzistence uživatelova chování. Implementovaná metoda `countLongestStreak` provádí lineární průchod přes všechny denní záznamy a sleduje nepřerušenou sérii úspěšně splněných dnů. Algoritmu stačí lineárně projít pole. Vždy, když potká další hodnotu `true`, zvýší si počítadlo o jedna a porovná hodnotu s dosavadním maximem (dosud nejdelší nalezená posloupnost) a do proměnné dosavadního maxima uloží maximum z těchto hodnot. Perfektní dny se počítají jednoduše tak, že se nehledá nejdelší sekvence, nýbrž se jen navyšuje počítadlo o jedna vždy, když jsou všechny návyky během dne splněny na sto procent.

Graf pokroku, implementovaný pomocí knihovny `MPAndroidChart`, vizualizuje vývoj dat. Algoritmus `formatHabitProgressDataForChart` zajišťuje přípravu dat pro grafické zobrazení tím, že eliminuje redundantní body a zachovává pouze významné změny v hodnotách. Jednoduše řečeno hledá jen změny v cílové hodnotě návyku v průběhu měsíce a pokud takovou změnu najde, uloží si datum této změny. Pokud nejsou k dispozici dostatečná data (méně než dva body), je graf automaticky skryt, protože by byl nevypovídající.

Vizuální prezentace statistik je realizována prostřednictvím několika klíčových komponent. Lineární graf zobrazuje vývoj pokroku v čase, kruhový progress bar reprezentuje celkový měsíční pokrok a další textové prvky poskytují detailní číselné shrnutí, včetně počtu perfektních dnů a nejdelší série úspěšných dnů.

3.4.7 Notifikační systém

V Aplikaci sloužící k tomu, aby uživatel plnil úkoly a zlepšoval se v návycích, nesmí chybět notifikační systém neboli připomínky. To funguje následovně. Notifikace jsou naplánovány pomocí AlarmManageru, a když nastane správný čas, BroadcastReceiver je zachytí a zobrazí uživateli upozornění.

Novější verze Androidu (API 26+) vyžadují notifikační kanál. Tento kanál se vytvoří v metodě `createNotificationChannel(Context context)`, kde se nastaví název, popis a úroveň důležitosti. Pokud kanál ještě neexistuje, systém jej vytvoří(). Bez tohoto kroku by se notifikace na novějších verzích Androidu nezobrazily. Plánování notifikace

```
1
2 NotificationChannel channel = new NotificationChannel(CHANNEL_ID, name,
3     importance);
4 channel.setDescription(description);
5 manager.createNotificationChannel(channel);
```

Zdrojový kód 3.18: NotificationChannel

Hlavní metodou pro plánování notifikace je `scheduleNotification`. Tato metoda vytvoří `PendingIntent`, který se odešle do systému s požadavkem na spuštění v daný čas – konkrétně pět minut před začátkem úkolu().

```
1
2 long notificationTime = taskTimeMillis - (5 * 60 * 1000);
3 alarmManager.setExact(AlarmManager.RTC_WAKEUP, notificationTime,
4     pendingIntent);
```

Zdrojový kód 3.19: alarmManager

Před naplánováním se zkontroluje, zda aplikace má oprávnění k přesnému plánování alarmů (od API 31+), a pokud ne, uživatel je přesměrován do nastavení. Při prvním spuštění aplikace je uživatel dotázán, zda chce nebo nechce povolit notifikace. Notifikace může povolit nebo zablokovat v sekci nastavení, kdykoliv bude chtít. Jakmile systém spustí naplánovaný Broadcast, zachytí ho `TaskNotificationReceiver`, který zkontroluje, zda uživatel povolil notifikace v `SharedPreferences`.

```

1
2 boolean notificationsEnabled = preferences.getBoolean("
   notifications_enabled", true);
3 if (!notificationsEnabled) return;

```

Zdrojový kód 3.20: Povolení notifikací

Pokud jsou notifikace povoleny, vytvoří se a zobrazí oznámení pomocí Notification-Manageru().

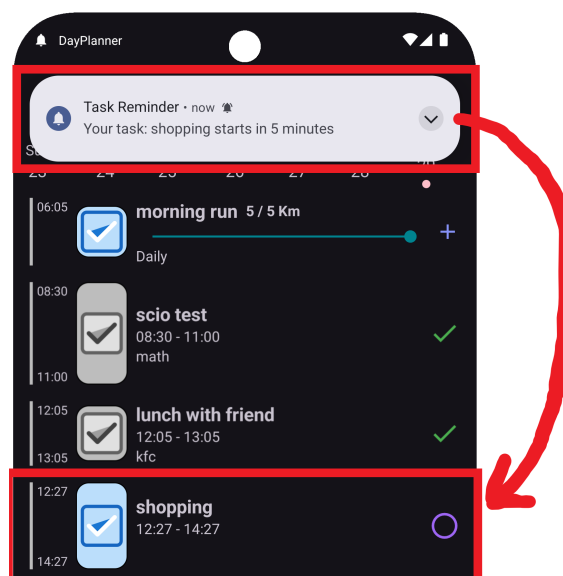
```

1
2 NotificationCompat.Builder builder = new NotificationCompat.Builder(
   context, TaskNotificationHelper.CHANNEL_ID)
3     .setSmallIcon(R.drawable.ic_notification)
4     .setContentTitle("Task Reminder")
5     .setContentText("Your task: " + taskTitle + " starts in 5
   minutes")
6     .setPriority(NotificationCompat.PRIORITY_HIGH)
7     .setAutoCancel(true);

```

Zdrojový kód 3.21: Vytvoření notifikace

Notifikace se zobrazí s titulem „Task Reminder“ a informací, že úkol začíná za 5 minut(viz obr. 3.8).

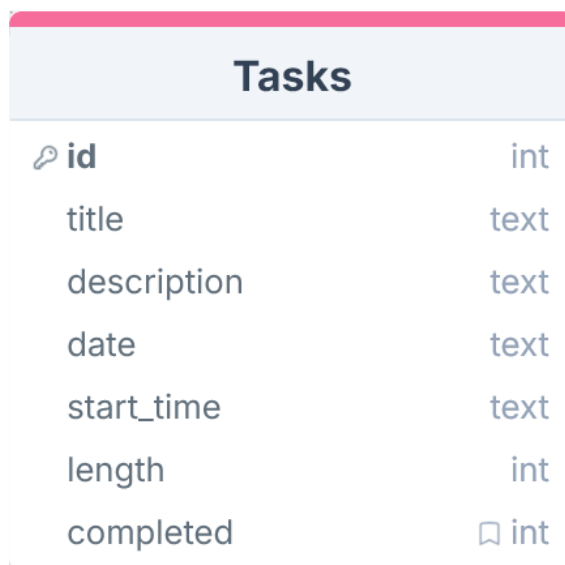




Obrázek 3.8: Připomínka

3.5 Struktura databáze

Aplikace DayPlanner využívá kombinaci lokální databáze SQLite a cloudové databáze Firebase Realtime Database, aby zajistila optimální rovnováhu mezi offline přístupem a synchronizací dat. Lokální úložiště slouží k ukládání uživatelských preferencí, jako je nastavení tmavého režimu nebo aktivace notifikací, a také jednorázových akcí, které po splnění již nejsou pro uživatele důležité. Naopak opakované akce a návyky jsou uchovávány v cloudu, protože obsahují cenné statistiky a dlouhodobá data, o která by uživatel nechtěl přijít, například při přechodu na nové zařízení. Díky tomu jsou všechny statistiky a návyky vázány na konkrétního uživatele a jsou dostupné z jakéhokoliv telefonu. Jednorázové akce nemají takovou hodnotu z hlediska dlouhodobého sledování, a proto je jejich ukládání do Firebase nadbytečné.

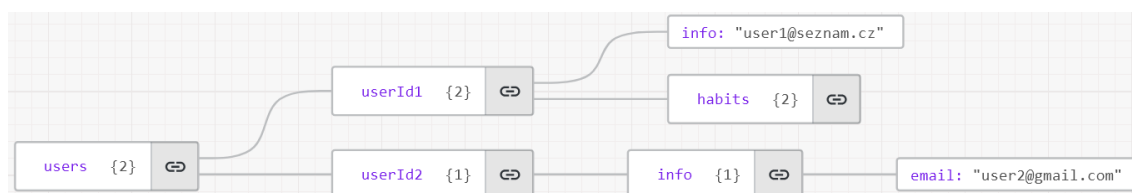
Databáze pro jednorázové akce neboli tasky vypadá podobně jako samotný model Task(viz obr. 3.7). Zde je schéma SQLite tabulky pro jednorázové akce(viz obr. 3.9).



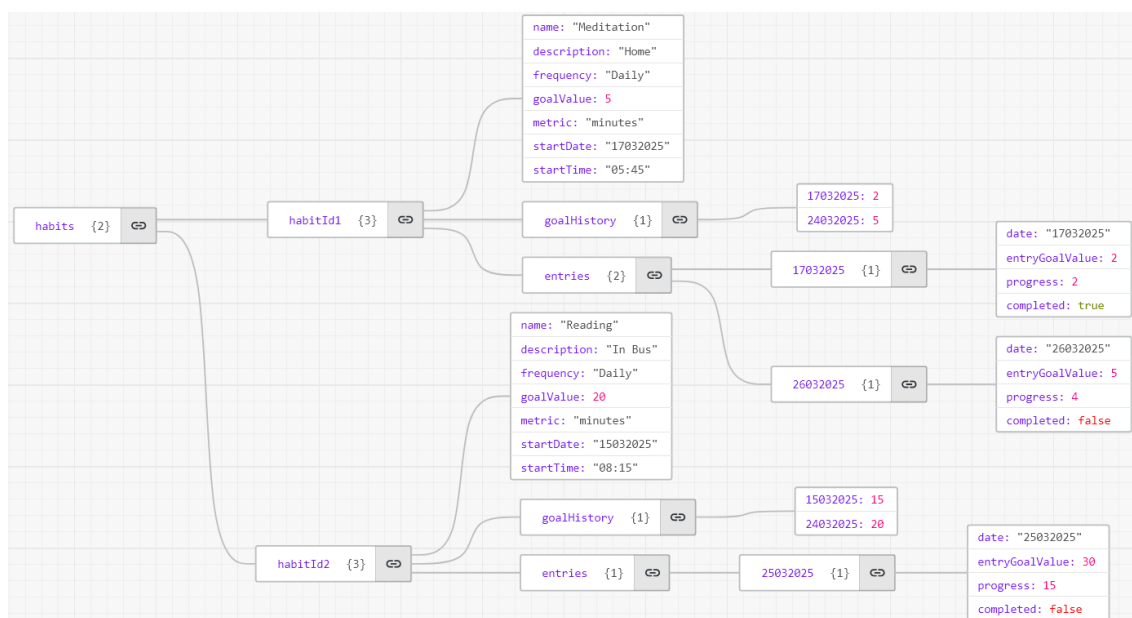
Tasks	
 id	int
title	text
description	text
date	text
start_time	text
length	int
completed	 int

Obrázek 3.9: SQLite Task Table

Firebase je takzvaná NoSQL databáze. Místo klasické relační databáze jako je například SQLite, ukládá Firebase data v takzvaných JSON objektech jako stromy. „Všechna data ve Firebase Realtime Database jsou uložena jako JSON objekty. Můžete si databázi představit jako cloudově hostovaný JSON strom. Na rozdíl od SQL databáze zde neexistují tabulky ani záznamy. Když do JSON stromu přidáte data, stanou se uzlem v existující struktuře s přiřazeným klíčem. Klíče si můžete určit sami, například pomocí uživatelských ID nebo sémantických názvů, nebo mohou být automaticky generovány pomocí metody `push()`“[5]. Schéma JSON stromu databáze mé aplikace je zobrazeno níže a rozděleno do dvou schémat pro lepší přehlednost. Každý uživatel má své unikátní userID generované Firebase(viz obr. 3.10). U každého uživatele je uloženo info s jeho emailem a návyky, pokud nějaké má(viz obr. 3.11).



Obrázek 3.10: Users JSON strom



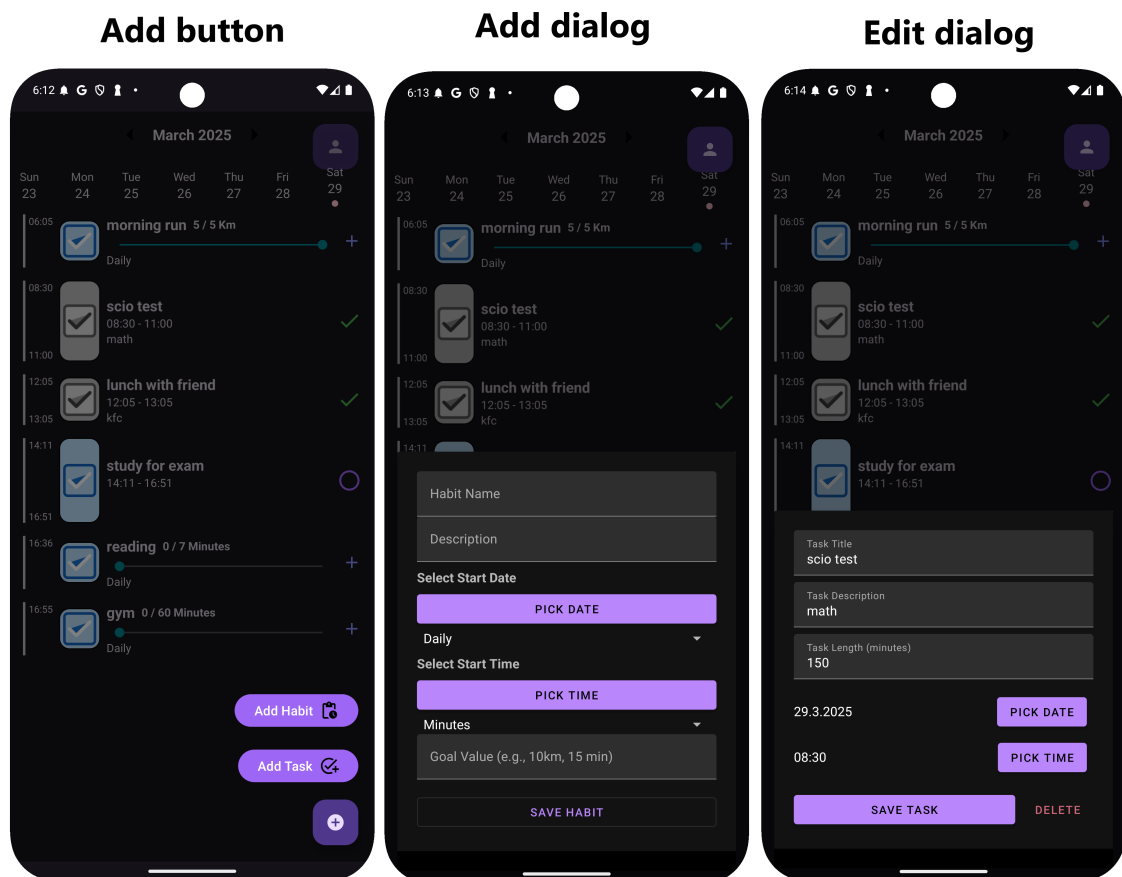
Obrázek 3.11: Habits JSON strom

4 Hlavní funkce aplikace Dayplanner

Tato kapitola stručně popisuje hlavní funkce, které uživatel může v aplikaci využívat.

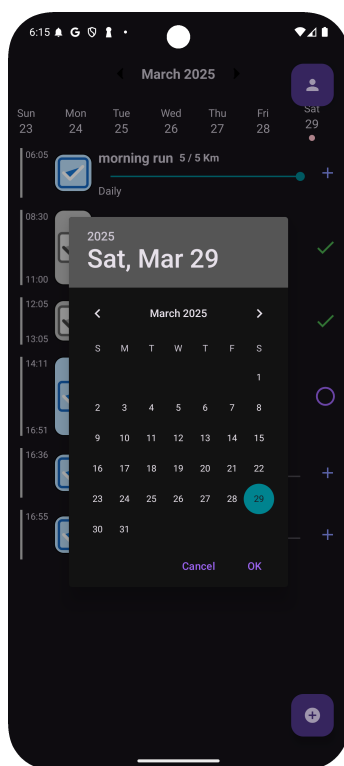
4.1 Hlavní aktivita

V hlavní aktivitě si uživatel může přidávat úkoly a návyky kliknutím na tlačítko „+“ (viz obr. 4.1 - Add button) se zobrazí přehledný dialog, ve kterém si může zvolit, zda chce přidat nový úkol nebo návyk. Poté, co uživatel klikne na jednu z možností, zobrazí se mu dialog na přidání (viz obr. 4.1 - Add dialog). Když uživatel klikne na ikonu položky, může ji i upravovat nebo smazat (viz obr. 4.1 - Edit dialog).



Obrázek 4.1: Main activity functions

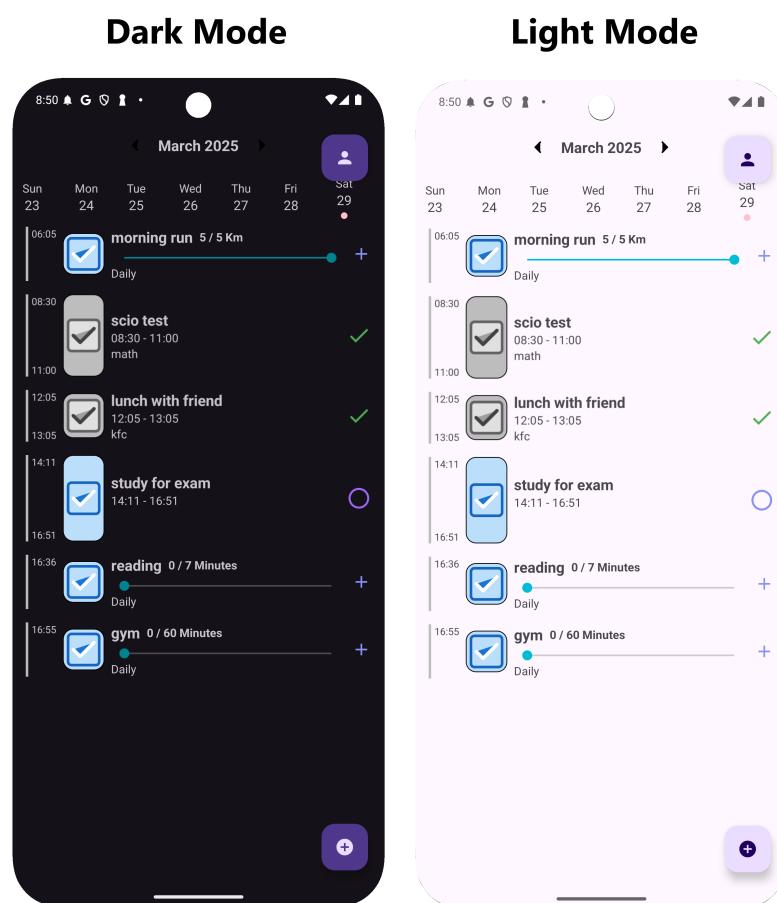
Kalendář umožňuje přepínání mezi jednotlivými dny buď swipováním do stran, nebo výběrem konkrétního dne pomocí kalendáře(viz obr. 4.2). Po kliknutí na den se přesune časová osa na vybraný den a zobrazí relevantní data.



Obrázek 4.2: Kalendář

4.2 Uživatelský profil

V uživatelském profilu si může uživatel přizpůsobit vzhled aplikace změnou mezi světlým a tmavým režimem. Pokud je přihlášen pomocí emailu, má možnost změnit heslo, avšak tato funkce není dostupná pro uživatele přihlášené přes Google účet. Dále si může nastavit, zda chce dostávat notifikace, a povolit nebo zakázat upozornění podle svých preferencí. Níže můžeme vidět ukázkou tmavého a světlého režimu.



Obrázek 4.3: Dark and Light Mode

5 Postup instalace

Nejprve je potřeba mít nebo si stáhnout **Android Studio** z oficiálního webu <https://developer.android.com/studio>. Pro stažení zdrojového kódu je potřeba projekt naklonovat pomocí příkazové řádky nebo uživatelského rozhraní.

Pro klonování pomocí uživatelského rozhraní spusťte Android Studio, na úvodní obrazovce klikněte na **Get from VCS**, vložte URL projektu: <https://github.com/gyarab/2024-4e-svacek-DayPlanner.git>. Projekt se stáhne a otevře. Pokud chcete použít příkazovou řádku, zadejte příkaz: `'git clone https://github.com/gyarab/2024-4e-svacek-DayPlanner.git'`. Po stažení projektu ho otevřete v Android Studiu.

Pro spuštění aplikace si můžete stáhnout emulátor nebo použít skutečné Android zařízení. Pro emulátor otevřete **Device Manager** v Android Studiu, klikněte na **Create Device**, vyberte emulátor (např. Pixel 9 Pro XL), stáhněte potřebný systémový obraz a dokončete nastavení emulátoru. Pokud chcete použít reálné zařízení, připojte telefon k počítači, v Android Studiu klikněte na **Pair using QR Code**, na mobilním zařízení povolte **Wireless Debugging** v **Developer Options** a naskenujte QR kód pro připojení.

Když máte nastavený emulátor nebo připojené zařízení, klikněte na **Run** a aplikace se spustí.

Závěr

Cílem této ročníkové práce bylo vytvořit mobilní aplikaci, která uživatelům umožní efektivně plánovat svůj den pomocí osvědčených metod plánování a propojit opakované úkoly s jednorázovými. Dalším klíčovým bodem bylo zahrnutí funkcionality pro sledování pokroku uživatele v oblasti návyků prostřednictvím statistik. Tento cíl se podařilo úspěšně splnit. Aplikace umožňuje uživatelům plánovat každý den pomocí časové osy, přidávat pokrok k návykům a plnit úkoly, přičemž mohou sledovat statistiky svého pokroku. Dále byla implementována autentizace přes e-mail nebo Google účet pro zajištění bezpečnosti uživatelských dat. Pro zajištění uživatelského komfortu byl přidán tmavý a světlý režim.

Do budoucna je možné aplikaci vylepšit implementací konceptu "offline-first designu", což znamená, že veškerá data by byla ukládána lokálně i na Cloud, což by umožnilo uživatelům používat aplikaci i offline, a při připojení k internetu by došlo k synchronizaci dat mezi zařízeními. Další možností rozvoje by bylo začlenění komunitních funkcí, například umožnit uživatelům sdílet své úspěchy a návyky s ostatními, což by mohlo podpořit motivaci, například formou budování nových návyků ve skupinách přátel.

Seznam použitých obrázků

3.1	Activity & Fragments schema	11
3.2	WeeklyHeaderFragment	12
3.3	Timeline Fragment	13
3.4	Statistics	15
3.5	Authentication Activity	20
3.6	Habit Model	25
3.7	Task Model	27
3.8	Připomínka	32
3.9	SQLite Task Table	33
3.10	Users JSON strom	34
3.11	Habits JSON strom	34
4.1	Main activity functions	35
4.2	Kalendář	36
4.3	Dark and Light Mode	37

Zdrojový kód

3.1	build.gradle	9
3.2	Hodnoty světlého motivu	16
3.3	Hodnoty tmavého motivu	16
3.4	Hodnoty barev	17
3.5	setThemePreference	17
3.6	getThemePreference	17
3.7	Nastavení motivu	18
3.8	Změnění motivu	18
3.9	googleSignInOptions	21
3.10	signInIntent	21
3.11	GoogleAuthProvider	21
3.12	Create New User	22
3.13	handleEmailVerification	22
3.14	SignInWithEmailAndPassword	24
3.15	Send Email Verification	24
3.16	isHabitVisibleOnDate	26
3.17	Data Cache	29
3.18	NotificationChannel	31
3.19	alarmManager	31
3.20	Povolení notifikací	32
3.21	Vytvoření notifikace	32

Seznam použitých zdrojů

- [1] Jordan B. Peterson. *12 pravidel pro život*. Accessed: 2025-03-20. 2025. URL: <https://www.jarosovi.cz/peterson-jordan-b-12-pravidel-pro-zivot/>.
- [2] Douglas Magazine. *Douglas Reads: Atomic Habits by James Clear*. Accessed: 2025-03-20. 2025. URL: <https://douglasmagazine.com/douglas-reads-atomic-habits-by-james-clear/>.
- [3] VOS Health. *Technika časových bloků: Zvyš svou produktivitu blok za blokem*. Accessed: 2025-03-20. 2025. URL: <https://vos.health/cs/story/Technika-%C4%8Dasov%C3%BDch-blok%C5%AF:-Zvy%C5%A1-svou-produktivitu-blok-za-blokem>.
- [4] SoBrief. *Atomic Habits - Shrnutí knihy*. Accessed: 20 March 2025. 2025. URL: <https://sobrief.com/cs/books/atomic-habits>.
- [5] Firebase Developers. *Structure Data in Firebase Realtime Database*. Accessed: March 29, 2025. 2025. URL: <https://firebase.google.com/docs/database/web/structure-data>.
- [6] Learning Bot. *How to Create Circular Progress Bar in Android Studio*. Accessed: 2025-03-31. 2025. URL: https://www.youtube.com/watch?v=7BVg8_WR7h4.
- [7] ComputerBerry. *How to view data from SQLite Database in Android Studio (2021)*. Accessed: 2025-03-31. 2025. URL: <https://www.youtube.com/watch?v=q0mp4krLZAQ>.
- [8] Setvdza-San. *SQLite + Android - Create Database Schema (Book Library App) | Part 1*. Accessed: 2025-03-31. 2025. URL: <https://www.youtube.com/watch?v=hJPK50p7xwA>.
- [9] Setvdza-San. *SQLite + Android - Insert Data in Database Table (Book Library App) | Part 2*. Accessed: 2025-03-31. 2025. URL: <https://www.youtube.com/watch?v=RGzblJuat1M&list=TLPQMzAwMzIwMjUggJ4Yy0oFdg&index=2>.

- [10] Easy Tuto. *How to Integrate Google Sign In in Android* | 2024. Accessed: 2025-03-31, 2025. URL: <https://www.youtube.com/watch?v=suVgcrPwYKQ&t=546s>.
- [11] BatCodes. *Firebase Email Authentication in Android*. Accessed: 2025-03-31, 2025. URL: <https://www.youtube.com/watch?v=ikfmhwvpDx4>.
- [12] BatCodes. *Send Email Verification Link to User By Implementing 2 Lines of code in your Application* | *Firebase*. Accessed: 2025-03-31, 2025. URL: <https://www.youtube.com/watch?v=2r-8xXXgpfU>.
- [13] Easy Tuto. *How to connect Firebase to Android Studio App* | 2024. Accessed: 2025-03-31, 2025. URL: https://www.youtube.com/watch?v=KSW5jyWXs_Y.