Vývoj edukativní mobilní aplikace ve frameworku Flutter

Bc. Ondřej Masný

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Ondřej Masný**
Osobní číslo:	**A22599**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Učitelství informatiky pro střední školy**
Forma studia:	**Prezenční**
Téma práce:	**Vývoj edukativní mobilní aplikace ve Frameworku Flutter**
Téma práce anglicky:	**Development of an Educational Mobile Application in the Flutter Framework**

Zásady pro vypracování

1. Popište současný stav technologií pro vývoj mobilních aplikací.
2. Analyzujte metodiky vývoje a porovnejte jejich efektivity.
3. Uveďte druhy a typy zabezpečení využívané u mobilních aplikací.
4. Znázorněte schéma komunikace API zařízení a databáze.
5. Navrhněte grafický wireframe v programu Figma.
6. Popište a demonstrujte použití testování mobilních aplikací.
7. Realizujte vývoj navržené aplikace ve frameworku Flutter.
8. Popište a demonstrujte její klíčové části řešení.
9. Uveďte výsledky a formulujte závěr.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. HOCHMUTH, Tomáš. Vývoj mobilních aplikací. 2014. PhD Thesis. AMBIS vysoká škola, as. [online].
2. ZIMA, Vratislav. Jak probíhá vývoj mobilních aplikací krok po kroku? [online]. Praha, 25. 6. 2020. Dostupné z: https://synetech.cz/cs/blog/vyvoj-mobilnich-aplikaci-krok-po-kroku.
3. AFREEN, C. Firza. Mobile Application Development [online]. 1. Lucknow: Book rivers, 2021. ISBN 978-93-90548-23-1.
4. PAYNE, Rap. Beginning App Development with Flutter [online]. Apress, 2019. ISBN 978-1-4842-5181-2.
5. HANSSON, Niclas; VIDHALL, Tomas. Effects on performance and usability for cross-platform application development using react native. 2016.
6. NORMAN, Don. The design of everyday things: Revised and expanded edition. Basic books, 2013.
7. Flutter docs [online]. Dostupné z: https://flutter.dev/.
8. Flutter – Reactive Programming – Streams – BLoC. Flutter – Didier Boelens [online]. 2020. Dostupné z: https://www.didierboelens.com/2018/08/reactive-programming-streams-bloc/.

Vedoucí diplomové práce: **Ing. David Šaur, PhD.**
Ústav matematiky

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Jméno, příjmení: Bc. Ondřej Masný

Název diplomové práce: Vývoj edukativní mobilní aplikace ve frameworku Flutter

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne …….

podpis studenta

ABSTRAKT

Cílem diplomové práce je vytvoření edukativní mobilní aplikace ve Frameworku Flutter určené pro žáky základních a středních škol, jejíž činnost bude primárně zaměřena na výuku informatiky. Aplikace Meteoratlas obsahuje funkce jako zadání výstrahy ostatním uživatelům v podobě notifikace, atlas mraků nebo předpověď počasí. Tyto funkce mohou být užitečné jako pedagogická pomůcka při výuce meteorologie, mobilní vývoj aplikací nebo matematika.

Klíčová slova: Flutter, výuka, Meteoratlas, počasí, aplikace

ABSTRACT

The goal of the thesis is to create an educational mobile application in the Flutter Framework intended for primary and secondary school students, the activity of which will be primarily focused on the teaching of computer science. The Meteoratlas application includes functions such as issuing a warning to other users in the form of a notification, a cloud atlas or a weather forecast. These functions can be useful as a pedagogical aid in teaching meteorology, mobile application development or mathematics.

.Keywords: Flutter, education, Meteoratlas, weather, app

Poděkování, motto a čestné prohlášení, že odevzdaná verze diplomové práce a verze elektronická, nahraná do IS/STAG jsou totožné ve znění:

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

V současném světě, kde jsme neustále vystaveni vlivům počasí, má schopnost předvídat a interpretovat meteorologické události klíčový význam. Mobilní aplikace se stávají nedílnou součástí našeho života a poskytují rychlý způsob, jak být stále v obraze. Můžeme jednak získat informace o počasí, jednak vytvořit základnu pro členy Meteoklubu, tedy ty, kteří by chtěli rozvíjet svou vášeň pro meteorologii. V kontextu škálovatelnosti aplikace může sloužit jako vizualizace dat pro školy s vlastní meteostanicí, jako je Gymnázium TGM ve Zlíně, odkud čerpám data pro svou aplikaci.

Tato aplikace je určena pro ty, kteří se zajímají o meteorologii a jejíž primárním účelem je rozšířit zájem o jevy, které se dějí kolem nás. Projekt Meteoratlas je rozdělen do dvou částí, jednou je zpracování hybridní mobilní aplikace, kterou realizuji v rámci této a druhou je serverová část pro získávání datových modelů počasí a vytvoření takzvaných *endpointů* pro aplikaci.

Aplikace obsahuje přehled denní, hodinového nebo specifické předpovědi počasí, dále sekci výstrah, která uživatele varuje o možných rizikovém počasí v dané oblasti, atlas mraků, kam mohou registrovaní uživatelé přidat vlastní záznam. Aplikace je také přizpůsobena změnou nastavení pro zlepšení uživatelského zážitku, a to možností anglicko-české lokalizace, nebo změnou barevného nastavení aplikace. Správce aplikace poté může přidávat v aplikaci vlastní výstrahy, které se pomocí push notifikací distribuují do mobilních klientských zařízeních, aby měl uživatel přístup k informací v reálném čase. Mimo notifikace budou výstrahy viditelné i na mapě v podobě polygonů, obsahujících informace o konkrétní výstraze.

Mimo aplikaci práce obsahuje pedagogické projektové listy do výuky, které využívají tuto aplikaci jako nástroj k dosažení projektového cíle pro různé vědní oblasti.

Cílem aplikace je rozšířit renomé meteorologie na základních a středních školách a zvětšit tak základnu příznivců této oblasti. Díky přenositelnosti mohou školy s vlastní meteostanicí využít aplikaci pro zobrazení vlastních dat, aniž by museli znovu vynalézat kolo.

TEORETICKÁ ČÁST

1 VÝUKA

Výuka dle dřívějších časů byla založena především na výkladu učitele, který má přehled v probírané látce, a snaze studentů vstřebávat tyto informace – tato forma výuky se nazývá frontální. Mimo ni však existují i jiné, které aktivizují žáky a zapojují je do školního procesu učení. Dnešní doba je plná podnětů všude okolo nás a zaujmout žáky je o to těžší, jejich pozornost je vratká, proto se musí pedagog snažit výuku obohatit. Metodik popisujících, jakým stylem vést výuku je spousta, jejich využití se mění dle potřeb a možností. Primární snaha je udělat z učení zážitek, zvýšit autonomii a zájem studentů. Základním předpokladem pro efektivní výuku je učitel, kterého baví a naplňuje učitelství.

John Dewey – americký pedagog a filozof zastával názor, že vzdělávání by mělo být více než jen předávání faktů a teoretických znalostí. Věřil, že učení by mělo být praktické, zkušenostní a relevantní pro život studentů a jejich další rozvoj. Tento přístup byl základem pro mou projektovou výuku, kde se studenti pracují prostřednictvím reálných úkolů, projektů a získávají tak zkušenosti a učí se spolupráci. [1]

Mým cílem je vyvinutí aplikace stávající pro rozvoj informatického myšlení mezi studenty díky tématu, které je pro všechny blízké, a to meteorologie. Vzhledem k potřebám pedagoga je třeba definovat metodologii, pomocí které dosáhneme vytyčeného cíle. Zvolená metodika vychází ze zdroje [3], který pojednává o metodikách výuky. Autor se v této knize zabývá metodami pro aktivní začlenění žáků do procesu výuky, jejich popisu a náležitostí, které by měly splňovat.

1.1 Výukové metody

Znaky výukových metod se skládají z několika bodů, které jsem našel ve knize českého didaktika p. Mojžíška [3], tedy metoda musí být:

- formativně účinná (tj. rozvíjí poznávací procesy);
- informativní (tj. předá nezkreslené plnohodnotné informace a dovednosti);
- racionálně a emočně působivá (tj. strhne, aktivizuje žáka k prožitku učení a poznávání);
- respektující k systému vědy a poznávání;
- výchovná (tj. rozvíjí morální, sociální, pracovní a estetický profil žáka);
- přirozená ve svém průběhu a důsledcích;
- použitelná v praxi, ve skutečném životě (tj. přibližuje školu životu);
- adekvátní k žákům a učiteli;
- didakticky ekonomická;
- hygienická

Pyramida učení ukazuje, jak různé metody výuky ovlivňují schopnost žáků si učivo zapamatovat. Jednoduše řečeno, metoda výkladu, kdy žáci pouze poslouchají, je nejméně efek-

Obrázek 1 – Pyramida učení. Zdroj [2]

tivní, protože zde mají žáci pasivní roli. Naproti tomu metoda, kdy žáci učí ostatní, je nejefektivnější, protože je zapojuje do aktivní výuky a vyžaduje hluboké pochopení látky.

Čím nižší je metoda v pyramidě, tím více jsou žáci zapojeni a tím lépe si látku osvojují. Pyramidová struktura tedy naznačuje, že aktivní zapojení vede k lepším výsledkům ve vzdělávání. Metody, které žáky stimulují k aktivní účasti, jako jsou diskuse, praktické cvičení nebo projekty, pomáhají dosahovat hlubšího porozumění a dlouhodobějšího zapamatování látky. [2]

Výukové metody můžeme rozdělit dle klasifikace [3] na tři druhy, a to:

- Klasické výukové metody – sem patří například:
  - Metody slovní – vyprávění
  - Metody názorně-demonstrační – předvádění a pozorování, práci s obrazem, instruktáž
  - Metody dovednostní praktické – napodobování, manipulování, laborování a experimentování
- Aktivizující metody
  - Metody diskusní
  - Metody heuristické (řešení problémů)
  - Metody situační
  - Metody inscenační
  - Didaktické hry
- Komplexní výukové metody
  - Frontální výuku
  - Skupinovou výuku
  - Partnerskou výuku
  - Individuální a individualizovanou výuku (samostatná práce žáků)
  - Kritické myšlení
  - Brainstorming
  - Projektovou výuku

Toto rozdělení nám poskytuje nadhled a do jisté míry sledujeme kombinování metod mezi sebou pro dovršení nejvyšší efektivity, kterou jsem představil ve formě pyramidy na obrázku č.1.

1.2 Pedagogické přístupy a metodiky

Metodiku, kterou seznáme za efektivní ještě nemusí znamenat, že je pro náš vhodná pro náš cíl a výuku. Aspekty, které ovlivňují rozhodnutí, jakou metodiku ve své výuce použít, vypadají následovně [7]:

- věk, počet žáků, čas a prostředí
- zákonitosti výukového procesu a z nich plynoucí didaktické zásady
- vymezené cíle a úkoly výuky;
- obsah a metody daného oboru transformovaného do vyučovacího předmětu;
- organizační formy (uspořádání vnějších podmínek vyučování, počet žáků,
- učební možnosti žáků a jejich osobnostní předpoklady;
- psychosociální charakteristika žáků i třídy jako celku (klima třídy, gender aj.);
- vnější podmínky výuky, např. geografické prostředí, společenské prostředí,
- hlučnost okolí, technická vybavenost školy;
- osobnost učitele, jeho odborná a metodická vybavenost, zkušenosti.

Níže uvedu rozdělení metodik, které vybíráme až po identifikaci předešlých bodů, tj. jsme si vědomi toho, co a jak chceme provést a zda je třída uzpůsobena pro tuto metodiku. Pokud ve třídě chybí třetina žáků, není vhodné zadávat projektové úkoly, ale zvolit například aktivitu diskuse ve skupinové práci na téma zadání projektu.

Dovoluji si zde uvést rozdělení, které popisuje metody, které podporují aktivní práce žáků ve výuce, ze kterých jsem čerpal. Své metodické materiály, viz přílohy, jsou zpracovány za využití projektového učení jakožto metodiky a budu se jí v příští kapitole věnovat podrobněji. Rád bych zde však uvedl i další metodiky, které čerpám z listu metodik výuky. [3]

Konstruktivismus

Konstruktivismus představuje teorii učení, která klade důraz na aktivní úlohu studenta při konstrukci vlastních znalostí a porozumění světu. V kontextu výuky meteorologie tento přístup podporuje interaktivní prostředí, ve kterém studenti aktivně zkoumají a interpretují meteorologické jevy. Důležitým prvkem je také zapojení studentů do procesu aktivního konstruování svých znalostí a porozumění prostřednictvím praktických projektů a aktivit. [1 ,8]

Projektové učení

Projektové učení je metoda výuky, která umožňuje studentům aplikovat své znalosti a dovednosti na řešení reálných problémů. V rámci výuky meteorologie projektové učení podporuje praktické učení a aplikaci teoretických znalostí do praxe prostřednictvím konkrétních projektů a úkolů. Studenti se zapojují do procesu plánování, provedení a hodnocení projektů souvisejících s meteorologickými jevy, což jim umožňuje hlouběji porozumět danému tématu a rozvíjet své analytické a řešitelské dovednosti.

Diferenciovaná výuka

Diferenciovaná výuka je pedagogický přístup, který respektuje individuální potřeby a rozdílné učební styly studentů. V rámci výuky meteorologie se diferenciovaná výuka uplatňuje prostřednictvím různých přístupů a metod, které umožňují přizpůsobení obsahu a metodiky výuky individuálním potřebám a preferencím studentů.

Kooperativní učení

Kooperativní výuka je moderní metoda skupinové práce, při které žáci společně pracují na řešení problémových úloh. Důraz je kladen na týmovou spolupráci, kde každý člen má svůj podíl zodpovědnosti a jeho individuální přínos je uznáván, stejně jako přínos celé skupiny. Výsledky jednotlivců ovlivňují hodnocení celé skupiny a podporují její soudržnost. Tato forma učení podporuje rozvoj komunikačních a sociálních dovedností studentů a posiluje jejich porozumění a zapojení do výuky meteorologie.

Experimentální učení

Experimentální učení je pedagogická metoda, která klade důraz na praktické zkušenosti a experimenty jako prostředek pro učení a porozumění novým konceptům. V rámci výuky meteorologie experimentální učení umožňuje studentům aktivně experimentovat s meteorologickými jevy a získávat praktické zkušenosti s jejich pozorováním a interpretací.

Každou z těchto metodik provádíme konkrétním způsobem a taktéž jej jinak hodnotíme. Tyto druhy hodnocení zaujímají v pedagogickém procesu velmi důležitou roli, a proto bych je rád vypsal níže v kapitole zvolil tu správnou pro své projektové učení.

1.2.1 Druhy hodnocení

Hodnocení je klíčovou součástí vzdělávacího procesu a může mít různé podoby a funkce. Učitelé využívají různé typy hodnocení podle svých potřeb, úrovně studentů a právních předpisů. Jeden z předpisů definujících známkování ve škole je ze školského zákona, a to Zákon č. 561/2004 Sb., Školský zákon. [4]

Níže jsou uvedeny hlavní typy hodnocení spolu s jejich příklady a charakteristikami.

Jedním z typů hodnocení je hodnocení formativní a je zaměřeno na identifikaci chyb a nedostatků žáků s cílem poskytovat zpětnou vazbu pro jejich zlepšení. Tato forma hodnocení se neobejde bez komunikace mezi učitelem a žáky a je bez tradičních známek nebo bodů a může být ústní nebo písemné. Jednotlivé druhy hodnocení rozepíšu níže pro získání všeobecného přehledu. [14]

1) Sumativní hodnocení je výsledné hodnocení, které poskytuje konečný přehled o výkonech žáka. Obvykle je využíváno na konci období, jako je závěrečné vysvědčení. Tento typ hodnocení se používá také mimo školu, například při certifikacích.
2) Na druhé straně je normativní hodnocení, které porovnává výkon jednoho žáka s výkonem ostatních žáků. Může to zahrnovat klasifikaci podle stupňů, jako jsou jedničkáři nebo trojkaři, a často slouží ke srovnávání mezi studenty.
3) Kriteriální hodnocení zjišťuje, zda žák splnil konkrétní úkoly nebo požadavky, bez ohledu na to, jak ostatní studenti uspěli. Učitelé stanoví kritéria, která určují, zda žák splnil požadavky, a hodnotí podle toho.
4) Diagnostické hodnocení se používá k identifikaci učebních problémů a specifických vzdělávacích potřeb žáků. Obvykle se provádí na začátku školního roku a může zahrnovat testy v pedagogicko-psychologických poradnách k detekci problémů, jako je dyslexie nebo dysgrafie.
5) Formální hodnocení je systematické a racionální. Žáci vědí předem, že budou hodnoceni určitým způsobem, což jim umožňuje se připravit.

Hodnocení má však více funkcí než jen předávání zpětné vazby žákům. Hodnocení by mělo motivovat žáky k dosažení lepších výsledků a podpořit jejich zájem o studium, informovat rodiče a žáka o výsledcích a pracovat s osobností žáka v kontextu rozvoje sebevědomí a dalších pozitivních vlastností, tedy diagnostikovat individuálně výkon žáků a adaptovat tak výuku.

1.3 Projektová výuka

Projektové učení je pedagogický přístup, který klade důraz na učení prostřednictvím projektů, řešení problémů a spolupráce. Tento přístup je oblíbený pro svou schopnost podporovat kritické myšlení, kreativitu a týmovou práci. Mimo uvedené výhody zde můžeme zařadit i rozvoj programovacích dovedností v konkrétní technologii, které vycházejí z procesu učení a opakování. Podmínky pro řízení projektové výuky musí být splněny, abychom dokázali co nejefektivněji nasimulovat tento reálný proces řízení a tvorby projektu. [1]

Očekávaným výstupem je rozvoj autonomie studentů a zodpovědnost za vlastní jednání. Spolupráce zde hraje taktéž klíčovou roli, protože studenti pracují ve skupinách nebo týmech, což zlepšuje komunikační a týmové dovednosti studentů. [2,8]

Předmětem hodnocení je výstup, ať už jde o fyzický výrobek, prezentaci, nebo jinou činnost, která je výsledkem výzkumu nebo pátrání. Tento výstup slouží jako důkaz práce studentů a může být hodnocen jako součást jejich výukového procesu. Můžeme taktéž hodnotit průběh tvorby projektu, kdy můžeme jako pedagog vnímat silné i slabé stránky žáka, a pomoci mu se na ně soustředit. Současně si jej mohou přidat do osobního portfolia díky nástroji pro správu verzí, kdy se každý programátor v ideálním případě prezentuje prvně odvedenou prací, až poté slovy.

1.4 Fáze projektové práce

Každý projekt obsahuje kroky, které popisují, jak se dopracovat k vytyčenému cíli. Tyto kroky můžeme rozdělit na čtyři fáze, a to plánování, realizace, prezentace a reflexe pro získání zpětné vazby. Níže tyto fáze podrobněji vysvětlím. V teorii se mohou názvy fází měnit v terminologii, nicméně obsahují stejné funkce kroků. Fáze čerpám z vícero zdrojů a jejich rozdělení, přičemž jsem vytvořil průnik fází tak, aby obsahovaly důležité etapy a zároveň to zapadalo do pedagogického klima.

1. Plánování

Studenti společně s učitelem plánují průběh projektu, stanovují cíle, zdroje a časový rámec. Diskutují o možných strategiích a postupech pro dosažení stanovených cílů.

2. Realizace řešení

Fáze začíná identifikací problému, který se studenti snaží vyřešit. V této fázi studenti vytváří vše, co o tématu nebo problému ví a navrhují řešení. *Dewey* věřil, že vzdělávání by mělo začínat otázkou nebo výzvou, která vzbuzuje zvědavost a vyžaduje aktivní přemýšlení.[2] Snažíme se proto jako zadavatel vytvářet příjemnou atmosféru a nabízíme odrazový můstek pro studenty, snažíme se prohlubovat jejich zájem otázkami, které jim pomohou vidět souvislosti.

3. Prezentace projektů

Studenti prozkoumávají téma projektu, shromažďují informace a hledají relevantní zdroje. Může se jednat o čtení knih, rozhovory s odborníky nebo terénní výzkum. Na základě nasbíraných informací a diskusí v týmu studenti plánují a realizují konkrétní činnosti související s projektem. To může zahrnovat experimenty, tvorbu produktů nebo aplikace získaných znalostí.

4. Vyhodnocení – reflexe

V této závěrečné fázi studenti hodnotí své výsledky a prezentují proces zkoumání. Diskutují o úspěších, obtížích a zkušenostech získaných během projektu.

Následuje vyhodnocení zadavatele, což je učitel, který projektovou výuku vede. Tento finální krok vytváří možnost pro získání zpětné vazby od zkušeného pedagoga, který má přehled nad tématem projektu a je stěžejní pro uvědomění žáků o jejich výkonu.

1.5 Návrh a realizace projektových úkolů

Pro svou diplomovou práci jsem vyvinul mobilní aplikaci, díky které mohu jako učitel rozšiřovat znalosti studentů v oblastech:

- **Meteorologie** – přichystané zadání obsahuje úkoly jako je například měření oblačnosti z atlasu mraků, porovnávání měřených dat a předpovědi výstrah
  viz. **Příloha** 1 – Projektová výuka Meteorologie.
- **Informační technologie** – vývoj mobilní aplikace a grafického návrhu, výuka mobilního vývoje aplikací, distribuce aplikace, vytváření databáze jevů viz. **Příloha** 2 - Projektová výuka Informační technologie
- **Matematika** – porozuměním numerických matematických modelů a porovnání s daty aplikace, viz. **Příloha** 3 – Projektová výuka Matematiky
- **Multimédia** – vytvoření multimediálních podkladů pro reprezentaci dat meteoklubu ve formě videí, prezentací nebo obrázků, viz. Příloha 4 - Projektová výuka Multimédií

Každá z oblastí má samozřejmě jiný průběh a cíle, kterých chceme dosáhnout a předat žákům. V mém případě se zaměřením na informační technologie, jelikož sám vyučuji na střední škole předmět webové prezentace a aplikace, kam mohu za správné volby implementace tuto aplikaci přenést a využít. Pro každou z oblastí je nutné definovat cíle, kterých chceme touto aplikací jako prostředkem dosáhnout.

Jak jsem zmiňoval v kapitole 1.3 – Fáze projektu, každý projekt se skládá ze čtyř fází. Studenti mohou pracovat samostatně nebo ve skupině. Projektová výuka umožňuje individuální a diferencovaný přístup k učení, což pedagogům nabízí možnost individualizovat projekt pro žáky, umožnit jim pracovat v jejich vlastním tempu a podle jejich vlastních potřeb. Meteorologický klub je zaměřen na počasí a na zpracování dat, pokud by účastníci měli zájem i o vývoj aplikace, vytvořil jsem zadání i pro vytvoření vlastní aplikace. Projekty jsou zaměřeny na práci s daty v aplikaci, kterou jsem vyvinul a budu se dále věnovat v praktické části. Teorii o vývoji mobilních aplikací a toho, jak co funguje, řeším v teoretické části, konkrétně v kapitole Flutter, aby čtenář pochopil základy tohoto nástroje a dokázal vytvořit vlastní aplikaci.

1.5.1 Zadání projektu

Projektové zadání pro aplikaci o meteorologii by mělo obsahovat několik klíčových prvků, které budou studenty motivovat k aktivnímu učení a zároveň jim poskytnou jasný směr pro jejich práci. Pokud má student vymyšlený vlastní projekt, může si vytvořit vlastní zadání. Zadání musí být jasně definované a vymezené, musí naplňovat využitou metodiku a její cíle. Je důležité žáka obeznámit s tím, co, jak a proč bude dělat a dle čeho bude hodnocen, zároveň je třeba přidat i seznam výstupů, které projekt přinese. [8]

Body projektového zadání jsou:

1) Popis projektu

Popis úkolu by měl poskytnout jasnou představu o tom, co studenti v projektu řeší a jak jej budou řešit. Zahrnuje specifikaci tématu projektu, cíle, kterých je třeba dosáhnout, a metodiku, která má být použita. Těmto metodikám pro vývoj software se věnuji v později v teoretické části v sekci řešení tvorby aplikace.

2) Cíle projektu

Cíle projektu by měly být formulovány tak, aby studenti věděli, co se od nich očekává a jakým směrem by se jejich práce měla ubírat. Cíle mohou zahrnovat například porozumění základním principům meteorologie, praktickou aplikaci teoretických znalostí nebo vytvoření konkrétních funkcí v aplikaci. V případě využití aplikace do výuky programování může být cílem vytvořit funkci pro převod stupňů Celsia na Fahrenheit.

3) Kritéria hodnocení

Kritéria hodnocení jsou klíčovým prvkem projektového zadání, který určuje, podle čeho budou studenti hodnoceni. Jejich podrobnost a konkrétnost jsou nezbytné pro transparentnost a spravedlivé posuzování studentských výstupů. Kritéria hodnocení by měla být jasná, specifická a měřitelná, což umožní studentům vědět, na co se mají zaměřit a jakým způsobem budou jejich práce posuzovány. Díky tomu je možné dosáhnout objektivního hodnocení a poskytnout studentům konstruktivní zpětnou vazbu. Indikátory kvality se mohou napříč projekty lišit, vypíšu níže tedy ty pro projektovou výuku programování mobilní aplikace.

1. Funkčnost řešení

   Hodnocení funkčnosti aplikace a její schopnosti poskytovat přesné a spolehlivé informace o počasí.

2. Téma projektu

   Posouzení komplexity projektu a jeho možného rozšíření. Přínos projektu a jeho využití, kvalita řešení.

3. Práce na projektu

   Individuální hodnocení řešitelů projektu a jejich výkonu během projektu. K tomuto nám slouží poznámky, které jako zadavatel během projektu můžeme sbírat pro získání přesné zpětné vazby pro studenty.

4. Prezentace projektu

   Hodnotíme kvalitu zpracování prezentace, vizuální stránku a také verbální projev žáků. Prezentování a veřejné vystupování v pracovní sféře nelze vytěsnit a je potřeba studentům poskytnout konstruktivní zpětnou vazbu. [1, 3, 6]

5. Hodnocení projektu

   Jakou formou bude žák hodnocen. Druhy hodnocení jsem popisoval v kapitole dříve, může se jednat například o formativní nebo formální hodnocení.

Jak jsem zmiňoval dříve, zadání se může změnit na základě metodiky, kterou jako pedagog využíváme pro dosažení vytyčeného cíle. Mým cílem je vytvořit podpůrnou edukativní aplikaci, která bude sloužit jako pomůcka studentům do předmětu meteorologie, matematiky a informačních technologií. Žák by měl vědět, co se od něj očekává, zadání tedy musí být jasné a pochopitelné. Žák může využít přichystané zadání, které jsem nasdílel v přílohách, nebo si může vytvořit vlastní dle zadání pedagoga. V této kapitole jsem popsal metodiky výuky, druhy hodnocení a zvolil si projektové učení jako metodu výuky.

2 APLIKACE A METEOROLOGIE

Vývoj aplikací je součástí světa programování už nějakou dobu, a prošel za tu dobu spoustu změnami, přibyly nové nástroje poskytující rozšíření v podobě knihovny – framework. Každé mobilní zařízení nyní disponuje integrovanou aplikací pro sledování počasí, která je zahrnuta ve standardní výbavě operačního systému od určité verze a je optimalizována pro plnohodnotné využívání nativních funkcí zařízení. Tato nativní aplikace nabízí uživatelům přístup k aktuálním meteorologickým informacím v jejich aktuální geografické lokalitě s minimálním úsilím. Nativní aplikace je vyvinutá pro jednu konkrétní platformu. Tedy například pro iOS nebo pro Android, přičemž se obě aplikace liší v implementaci, a způsobu komunikace, jelikož se liší architektura hardware a software zařízení.

Před vývojem je klíčovým faktorem stanovit problematiku aplikace, existuje již mnoho aplikací pro meteorologii, dokážeme najít a stanovit základní funkce, které mají všechny aplikace napříč OS sdílené. Průnikem všech funkcí v aplikaci meteorologického zaměření je například předpověď počasí, aktuální počasí a případně detailnější informace ze meteostanice, dělené na radarové a satelitní, poskytující rozličné data. Cílem je však poskytnout informace koncovému uživateli o počasí, zprostředkované komunikací aplikace se serverem a meteostanicí. Výsledkem je pak schopnost uživatele determinovat nadcházející počasí a přizpůsobit své plány.

2.1 Historie mobilních aplikací

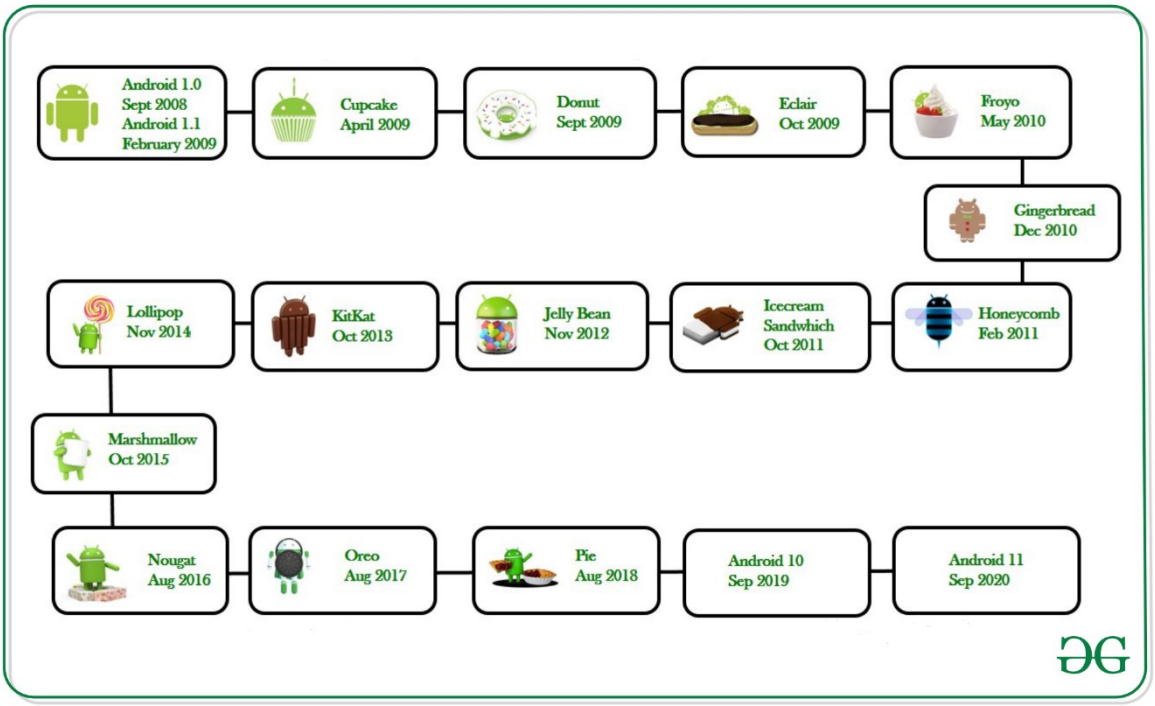
První mobilní aplikace vznikly v době, kdy se mobilní telefony začaly rozvíjet z jednoduchých zařízení sloužících pouze k telefonování a posílání SMS zpráv do více multifunkčních zařízení. První mobilní aplikace byly vytvářeny v omezeném prostředí starších operačních systémů, jako byl například Symbian, Palm OS a Windows Mobile.

Mobilní aplikace jsou klíčovým prvkem současné digitální éry, která ovlivňuje způsob, jakým lidé komunikují, pracují a tráví svůj volný čas. Historie mobilních aplikací je fascinující cesta inovace a technologického pokroku, která začala před mnoha lety a v průběhu času prošla dramatickým vývojem. Historie mobilních aplikací začala v 90. letech 20. století, kdy se mobilní telefony začaly stávat běžnými. První mobilní aplikace byly jednoduché hry a aplikace pro organizaci kontaktů či kalendáře, které byly často zabudovány přímo do operačního systému mobilního zařízení. Například první verze mobilní aplikace Snake byla

implementována v roce 1997 firmou Nokia pro její mobilní zařízení Nokia 6110, které bylo jedním z prvních mobilních telefonů s barevným displejem a možností hraní jednoduchých her. [19]

2.1.1 Android

Zařízení Android tvoří v dnešním světě velmi důležitou část obrovské části uživatelů mobilních telefonů. S globálními nepokoji v éře COVID-19 nyní populace vstoupila do digitální sféry. Příběh Androidu se datuje do roku 2003, kdy Andy Rubin, Rich Miner, Nick Sears a Chris White spoluzaložili start-up Android Inc. v Palo Alto v Kalifornii. Google vycítil potenciál, který produkt v sobě nese, a uzavřel dohodu v hodnotě 50 milionů dolarů na získání Androidu v roce 2005. První veřejná beta verze Androidu 1.0 byla konečně zveřejněna 5. listopadu 2007. Níže na obrázku uvádím chronologický vývoj platformy Android, kdy se jednotlivé verze liší například novou funkcionalitou, bezpečnostními a vzhledovými prvky a optimalizací systému. [26]



Obrázek 2 – Vývoj Android verzí. Zdroj [26]

2.1.2 Operační systém Apple

Od roku 2008 bylo staženo přes sto miliard aplikací z Apple App Store do iPhonů nebo iPadů uživatelů. Tisíce vývojářů softwaru napsaly tyto aplikace pro mobilní platformu Apple „iOS“. Technologie a nástroje pohánějící revoluci mobilních „aplikací“ však nejsou samy o sobě nové, spíše mají dlouhou historii přesahující více než třicet let, která se váže nejen k NeXTu, společnosti Steve Jobs, kterou založil v roce 1985, ale k začátkům. softwarového inženýrství a objektově orientovaného programování na konci 60. let. [19]

Steve Jobs, jméno, které zná téměř každý, kdo slyšel o značce Apple. Spoluzakladatel společnosti Apple společně se Stevem Wozniakem, opustil firmu v roce 1985 poté, co byl zbaven moci v rámci pokusu o převrat v představenstvu. Toto odcházení vedlo k založení společnosti NeXT, která se zaměřila na vývoj vlastního hardwaru a softwaru. Zajímavou inovací společnosti NeXT bylo operační prostředí NeXTSTEP, které se stalo základem pro pozdější vývojáře aplikací pro platformy Apple. [19]

NeXTSTEP poskytl základ pro vývojáře aplikací pro platformy Apple, "V roce 1997 Jobs demonstroval technologii, která se později stala Cocoa, vývojovým systémem, který byl použit tisíci vývojáři aplikací pro iOS po celém světě." Tato technologie umožnila vývojářům aplikací získat výraznou produktivitu, cituji "Kombinace Objective-C, AppKit a Interface Builderu umožnila Steveu Jobsovi chlubit se tím, že NeXTSTEP může zvýšit produktivitu vývojářů o pět až desetkrát.

První iPhone, který byl uveden na trh v roce 2007, přinesl revoluční uživatelské rozhraní a otevřel cestu pro vývoj široké škály mobilních aplikací. Jednou z největších inovací od Applu bylo uvedení App Store v roce 2008. App Store byl prvním oficiálním tržištěm pro mobilní aplikace, které umožnilo vývojářům distribuovat své aplikace a uživatelům je snadno stahovat a instalovat. Apple aktivně podporoval vývojářskou komunitu tím, že poskytovala nástroje, jako je Xcode a iOS SDK, které umožnily vývojářům vytvářet kvalitní mobilní aplikace pro platformu iOS v samotných začátcích mobilních aplikací.

3 METODIKY VÝVOJE SOFTWARE

Jako první je důležité stanovit v kontextu vývoje software co je to metodika, jak vzniká a v čem se liší. Metodika je soubor doporučených praktik a postupů pokrývající životní cyklus vývoje aplikace, to znamená od zadání až po dokončení. Metodologie je disciplína zabývající metodami vývoje software.

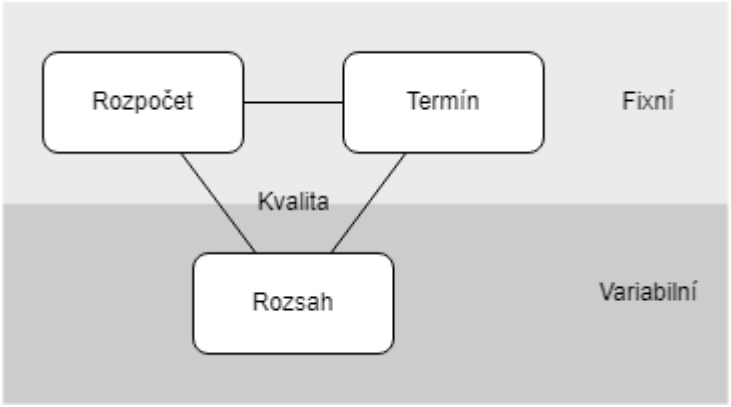
V následujících kapitolách se zabývám konkrétním popisem metodik, vlastností a preferencemi v dnešní době.

3.1 Dělení metodiky vývoje mobilních aplikací

Samotné rozdělení metodik je klíčové pro úspěšné předání aplikace klientovi, jelikož metodiky obsahují důležité kroky v rámci definování kroků potřebných ke splnění cíle, stanovení problematických oblastí implementace a jejich analýze a faktorů, dle kterých měříme úspěšnost projektu.

Důvody využívání metodik je možné shrnout do několika základních bodů a otázek [37]

- Jak – je nutné stanovit etapy, kterými by se mělo při samotném vývoji projít a postupovat
- Co – jednotlivé kroky musí mít jasně definované své vstupy a výstupy
- Kdo – uvedené kroky realizuje, zda je nutná spolupráce expertů, časté konzultace se zákazníkem, pro kterého je SW vytvářen



Obrázek 3 – faktory ovlivňující vývěr metodiky [37]

Na základě obrázku 5 výše můžeme říct, že tento troj-imperativ a jeho parametry měnit, což se nám může hodit v případě, že máme variabilní například rozpočet, nebo termín.

Metodik, které se mění organizačně a nabízí výhody oproti jiným je několik a dělíme je na tradiční a agilní metodiky.

4.1 Code and fix metodika

Možná ne nejznámější, ale nejvíce realizovanou metodikou uplatněnou ve vývoji je takzvaný Code and fix, přičemž dokážeme přijít na postup metodiky již ze samotného názvu, kontinuálně vyvíjíme nové funkce a v případě nastavených procesů buďto iterativně, nebo dle potřeb opravujeme stávající kód. [10]

Metodika se dělí na 3 fáze:

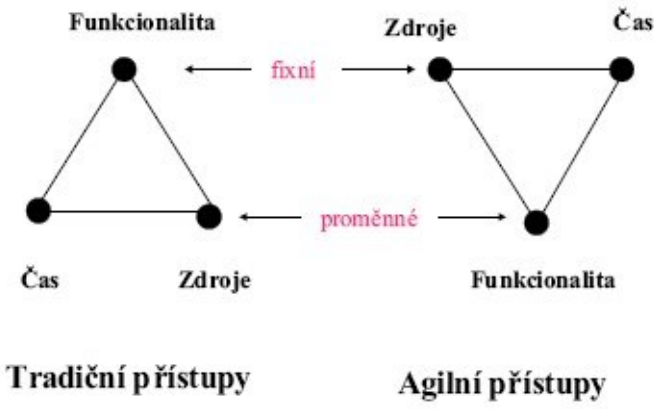
- Implementace
- Oprava chyb
- Rozšíření (opakování cyklů implementace a úprav)

Ze samotných fází lze vidět benefit agilní podoby, kdy jsme schopni klientovi, kterému děláme aplikace, rychle reagovat na změny požadavků, na incidenty nebo produkční bugy. Na druhou stranu metodika není udržitelná ve větších organizacích, kdy je například vývojový tým dělen maticově, jelikož vývoj není jen o psaní kódu, ale především o analýze, architektuře aplikace a rozhraní, evaluaci týmu, což zde chybí. Tato metodika je vhodná pro menší týmy nebo jednotlivce, kdy vynikne spíše benefit rychlosti a vysoké adaptability.

3.2 Agilní metodika

Agilní metodiky vývoje softwaru se objevily jako reakce na nedostatky tradičních modelů, jako je vodopádový model. V agilním programování se klade důraz na flexibilitu, rychlou adaptaci na změny a úzkou spolupráci mezi vývojovým týmem a zákazníky. Tradiční přístupy k vývoji softwaru se spoléhají na fixní specifikaci definovanou na začátku projektu vizualizovaných na obrázku níže. Tato specifikace je považována za neměnnou a neproměnnou během průběhu projektu, což může vést k problémům. Implementace takové specifikace vyžaduje určitý čas a zdroje, jako jsou platy vývojářů a kapitál firmy. Avšak hodnoty těchto proměnných jsou odhadnuty v úvodní fázi projektu a mohou se během jeho průběhu změnit, což vede k prodlevám a zvýšeným nákladům. U agilních metodik je fixní rozpočet a termín

dodání. Rozsah je variabilní. Rozsahem můžeme rozumět kvantita kódu, komplexita aplikace. Oproti tradičních metodik mají projekty typicky vyšší kvalitu, jelikož se projekt adaptuje vnějším faktorům. [29]



Obrázek 4 – Porovnání parametrů agilní a tradiční metodiky
[30]

Výhody agilního programování

Výhody plynou ze samotného důvodu vzniku agilní metodiky jako alternativu tradičním způsobům vývoje, jako je vodopádový model. Přesto je však vypíšu níže z odborné literatury popisu agilní metodiky, které benefity uvádí následovně [29]:

- Flexibilita – týmy mohou rychle reagovat na změny požadavků.
- Iterativní přístup – projekty jsou rozděleny do menších částí, což umožňuje průběžné doručování hodnoty zákazníkovi.
- Komunikace – důraz na spolupráci mezi členy týmu a zákazníky.
- Prioritizace – agilní týmy se zaměřují na doručování toho, co má pro zákazníka nejvyšší hodnotu.

Najdou se však i nevýhody, které jsou nastaveny spíše lidským faktorem a organizační stránkou věci, ne vždy jsou ideální podmínky pro zavedení agilní metodiky k sobě do týmu.

Mezi nevýhody můžeme zařadit nedostatek struktury v týmech, jelikož je zde potřeba spousta rolí a kompetencí, které je potřeba zajistit. Komunikace je zde vyžadována od všech členů týmu, nikoliv jen dedikovaným osobám, které komunikují se zákazníkem, ale členové

transparentně mluví o svém pokroku, jelikož je zde dodávka podstatná a s tím spjaté i deadline. [29]

Manifest agilní metodiky

Existuje oficiální dokument, přesněji manifest, který obsahuje body pro naplnění agilní metodiky, ze kterého budu čerpat pro zbytek kapitoly. [21] Autoři manifestu agilního vývoje software reagovali na potřeby vývoje softwaru, které se postupem času měnil. Kromě tradičního důrazu na spolehlivost softwaru začaly vznikat požadavky týkající se ekonomických aspektů vývoje a psychologie skupinové práce. Manifest agilního vývoje software vznikl jako výsledek společného setkání autorů v Utahu v únoru 2001, kde se shodli na několika klíčových myšlenkách.

Manifest agilního vývoje software zdůrazňuje především flexibilitu a schopnost reagovat na změny požadavků. Namísto toho, aby byly požadavky na software striktně definovány na začátku projektu, agilní metodiky preferují průběžnou adaptaci a inkrementální vývoj. Tento přístup umožňuje týmu lépe reagovat na změny v prostředí zákazníka a poskytovat mu včasnou zpětnou vazbu.

Dovolím si vypsat pár bodů z tohoto manifestu [21].

1. Nejúčinnější a nejefektivnější metoda předávání informací do vývoje je konverzace tváří v tvář.
2. Funkční software je primárním měřítkem pokroku.
3. Agilní procesy podporují udržitelný rozvoj.
4. Sponzoři, vývojáři a uživatelé by měli být schopni udržet konstantní tempo donekonečna.
5. Neustálá pozornost k technické dokonalosti a dobrý design zvyšuje agilitu.
6. Jednoduchost – umění maximalizovat množství nevykonané práce – je zásadní.
7. Nejlepší architektury, požadavky a návrhy pocházejí od samo organizujících se týmů.
8. V pravidelných intervalech tým přemýšlí, jak by se staly efektivnějšími, pak ladí a upravují podle toho dané procesy.

3.3 Scrum metodika

Další metodikou, která patří do skupiny agilních, je metoda Scrum. Duo Hirotaka Takeuchi a Ikujiro Nonaka, zakladatelé této metodologie popisují Scrum jako přístup, který má zvýšit rychlost a flexibilitu procesu vývoje. Nicméně jeho plné uplatnění přišlo až o deset let později, kdy Jeff Sutherland a Ken Schwaber popularizovali tuto metodologii a z jejichž materiálů bude čerpat pro svou kapitolu. [30]

Scrum poskytuje strukturovaný rámec pro řízení projektů různé velikosti pomocí různých nástrojů, včetně sledování produktových požadavků, denních setkání a retrospektiv.

Dle Schwaubera [30] stojí metodologie na třech základních pilířích:

- transparentnost
- inspekce
- adaptace

Transparentnost zajišťuje, že všichni účastníci mají jasný přehled a porozumění klíčovým aspektům procesu.

Inspekce umožňuje průběžnou kontrolu postupu a ujištění, že není nic, co by bránilo týmu dosáhnout stanoveného cíle.

Adaptace umožňuje flexibilní přizpůsobení procesu na základě zjištění z inspekce, což následně přispívá k průběžnému zlepšování.

Cyklická povaha Scrumu umožňuje zjištění, jaké funkce by měl produkt poskytovat, a zároveň usnadňuje experimentování a učení. Scrum v sobě integruje iterativní postup vývoje produktu, který je založen na zpětné vazbě od uživatelů, zákazníků a dalších zainteresovaných stran. Scrum se skládá z týmů a souvisejících rolí, událostí, artefaktů a pravidel. Každá část Scrumu má svůj specifický účel a je nezbytná pro úspěch této metodiky.

Role Scrum Mastera, Product Ownera a členů týmu, spolu s definovanými událostmi a artefakty, tvoří základní stavební kameny tohoto přístupu. Scrum využívá samo organizující týmy, iterativní dodávky produktu, překrývající se fáze, dynamickou hodnotu multidisciplinárních týmů a organizační učení. Klíčové faktory úspěchu této metody zahrnují práci na určitém množství úkolů v prioritním pořadí dle potřeb zákazníka a schopnost cyklického učení a experimentování.

Model týmu ve Scrumu z dokumentace Scwaubera [30] je navržen tak, aby optimalizoval flexibilitu, kreativitu a produktivitu, aby byly dodrženy základní pilíře zmíněny výše. Rád bych zmínil klíčové role a jejich kompetence.

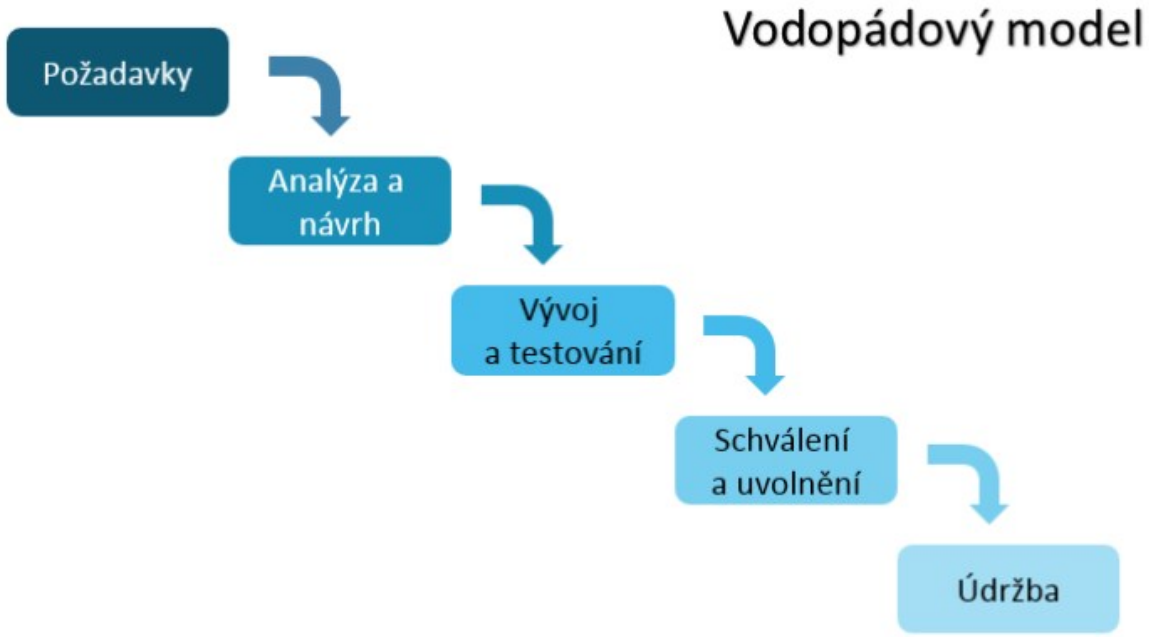
- **Produktový vlastník** je zodpovědný za maximalizaci hodnoty produktu a práce Development Teamu. Způsob, jakým se tohoto dosahuje, se může výrazně lišit v různých organizacích, Scrum týmech a jednotlivcích. Produktový vlastník je jediná osoba zodpovědná za správu Produktového backlogu, ze kterého se následně vytváří potomek jednotlivých User stories – zadání.
- **Development Team** se skládá z profesionálů, kteří provádějí práci na dodání potenciálně uvolnitelného Inkrementu "Hotovo" na konci každého Sprintu. Pouze členové Development Teamu vytvářejí Inkrement. Development Teamy jsou strukturovány a zmocněny organizací, aby si samy organizovaly a řídily svou práci. Tato synergia optimalizuje celkovou efektivitu a účinnost Development Teamu.
- **Scrum Master** je zodpovědný za to, aby byl Scrum pochopen a realizován. Dodržuje a nastavuje teorii, praktiky a pravidla Scrumu

Každý **Sprint** je tvořen časovým rámcem jednoho měsíce nebo méně, během něhož je vytvořen použitelný a potenciálně využitelný kandidát produktu. Sprinty by měly mít konzistentní délku trvání po celou dobu vývoje. Každý nová iterace začíná okamžitě po skončení předchozího Sprintu. Na konci dochází k retrospektivě uplynulého Sprintu a evaluují se různé poznatky ke zlepšení, pochvaly nebo kritická místa. [30]

3.4 Vodopádový model

Nejstarší a nejznámější klasickou metodou řízení projektů je vodopádová metoda. Tato metoda, vytvořená v sedmdesátých letech minulého století, je stále používána díky své jednoduchosti. [31]

Vodopádový model se vyznačuje postupnými kroky, které se nepřekrývají, což znamená, že každá fáze končí před zahájením další. Tato metoda je považována za jednoduchou, což usnadňuje řízení a plánování projektu. Jednotlivé fáze metody jsou navzájem propojeny lineárně, jak ukazuje obrázek níže.



Obrázek 5 – Vodopádový model – zdroj [5]

Časté změny požadavků mohou vést k potřebě revidovat fáze projektu nebo i celý projekt, což je považováno za největší nevýhodu této metody. Navíc, zpoždění jednoho úkolu může způsobit zpoždění všech následujících úkolů, což zvyšuje náklady a může vést k nespokojenosti zákazníka.

I přesto, že Vodopádová metoda má své nevýhody, je stále vhodná pro malé projekty s pevně stanovenými termíny, rozpočtem a rozsahem. [31] U velkých projektů však tato metoda obvykle není účinná, protože neumožňuje adekvátní zapojení zákazníka během vývoje a může vést k obtížím při řešení požadavků na změny. S jasně definovanými fázemi lze přesně plánovat termíny a zdroje.

3.5 Porovnání efektivity metodik

Agilní metodiky přinášejí alternativní přístup, který rozlišuje mezi fixními a volnými proměnnými. Na začátku projektu jsou odhadnuty a plánovány potřebné zdroje a čas, zatímco funkcionalita je ponechána jako volná proměnná, jejíž hodnota se mění v průběhu projektu podle aktuálních potřeb a úspěchu projektu. To znamená, že v případě potíží s implementací části funkcí se mohou odložit, aby byl zachován harmonogram.[31]

Z hlediska zdrojového kódu dle zdroje o nauce agilních metodik [30] kladou agilní metodiky důraz na zdrojový kód jako hlavní zdroj informací a indikátor kvality. Na rozdíl od tradičních přístupů, které staví na objemných dokumentacích, agilní metodiky se spoléhají na zdrojový kód jako nejlepší způsob přenosu informací. Dalším rozdílem mezi agilními a tradičními přístupy je iterativní a inkrementální vývoj. Plán projektu podle agilních metodik sestává z krátkých iterací, během nichž se postupně vyvíjejí inkrementy výsledného softwaru. Tím se získává rychlejší zpětná vazba a eliminuje se riziko nesouladu s požadavky zákazníka. Tradiční přístupy také mohou využívat iterativního a inkrementálního vývoje, ale často s delšími iteracemi.

Ve větších týmech je důležitá součinnost individuálních členů, probíhá rozvoj mezilidských vztahů, jestliže je zde pro to rozvoj. Komunikaci udržujeme jednak s klienty a jednak s týmem, avšak je zbytečné, aby se designer účastnil meetingu nastavování hostingu aplikace, a tak je potřeba, aby informace byly sdělovány a distribuovány napříč týmem nebo organizací, a zamezilo se tak entropii informací. To by mohlo mít dopad na dodávku kódu, což je pro nás jako vývojáře a budovatele své značky v podobě aplikace nežádoucí. \

Klade se důraz na tuto formu komunikace v rámci týmu, což může eliminovat mnoho komplikací a problémů vývoje. Naopak tradiční přístupy se často soustředí spíše na procesy než na mezilidské vztahy, přičemž řešení sociálních problémů je ponecháno na vedoucích pracovníků.

Závěrem je nutno podotknout, že není univerzální metodika, která by vyřešila všechny naše problémy a optimalizovala vývoj software. Dle případů projektů měníme metodiku i podle přání zákazníka a jeho možností, přesto lze však říci, že správně zvolená metodika je polovina úspěchu, druhou je proškolený tým, který dodržuje náležité nuance a je autonomní.

4 ZABEZPEČENÍ MOBILNÍCH APLIKACÍ

Android má vestavěné bezpečnostní funkce, které výrazně snižují četnost a dopad problémů se zabezpečením aplikací. Systém je navržen tak, že můžeme své aplikace obvykle vytvářet s výchozími oprávněními k systému a souborům a vyhnout se obtížným rozhodnutím o zabezpečení. Informace o zabezpečení vychází z oficiální dokumentace od konkrétního vývojáře mobilní platformy. Vzhledem k tomu, že platformy, které využívá majoritní část uživatelů, existují už nějakou dobu a musely postupem času s přibývajícími útoky a potencionálními hrozbami věnovat zdroje i do zabezpečení jejich architektury, na čemž pak můžeme stavět funkční aplikace.

Základní funkce zabezpečení dle oficiální dokumentace Android [9,]:

- Sandbox aplikací pro Android, který izoluje data aplikace a spouštění kódu od ostatních aplikací. Každé aplikaci je přidělen unikátní identifikátor uživatele neboli UID, a vytvořen separátní proces identifikovatelný pod tímto identifikátorem.
- Aplikační rámec s robustními implementacemi běžných funkcí zabezpečení, jako je kryptografie, oprávnění a zabezpečená mezi procesová komunikace (IPC).
- Technologie jako randomizace rozložení adresního prostoru (ASLR), no-execute (NX), ProPolice, safe_iop, OpenBSD dlmalloc a calloc a Linux mmap_min_addr ke zmírnění rizik spojených s běžnými chybami správy paměti.
- Uživatelem udělená oprávnění k omezení přístupu k funkcím systému a uživatelským datům. Tím se omezují možnosti zneužití a neoprávněného přístupu aplikací k citlivým informacím.
- Oprávnění definovaná aplikací pro řízení dat aplikací na základě jednotlivých aplikací.

4.1 Ochranné mechanismy pro zabezpečení mobilních aplikací

Bezpečnostní strategie je tvořena pěti klíčových pilířů dle oficiální dokumentace Flutter [16]:

- Identifikace hrozby
- Detekce v zařízení
- Ochrana (aplikace zabezpečení)
- Reakce na real-time hrozbu a obnova

Tato definice je užitečná pro rozvržení základní struktury pro vymezení zabezpečení aplikací. Znamená to, že je nejdříve potřeba znát slabé stránky aplikace a jejich možné zneužití. K tomu nám může dopomoci disciplína zvaná reverzní inženýrství.

Reverzní inženýrství

Jednou z efektivních metod ochrany proti těmto útokům je reverzní inženýrství. Reverzní inženýrství spočívá ve zkoumání různých aspektů aplikace, jako jsou zdrojový kód, souborová struktura, síťová komunikace a uživatelské rozhraní, s cílem identifikovat možná slabá místa a bezpečnostní nedostatky. Tento proces umožňuje odhalit potenciální hrozby a opravit zranitelnosti dříve, než je útočníci využijí.

Reverzní inženýrství využívá specializované nástroje a techniky, jako je rekompilace aplikace na zdrojový kód, analýza *bytecodu*, sledování sítě a debugování aplikace v reálném čase. Tato podrobná analýza může odhalit chyby v kódu, které by mohly být zneužity pro neautorizovaný přístup nebo narušení bezpečnosti.

Nad naše možnosti nám může dopomoci ještě pár mechanismů, které zvýší míru zabezpečení. Mohou to být například:

1. Runtime Application Self-Protection (RASP):

   RASP je technologie, která je schopna detekovat a blokovat škodlivé aktivity přímo na úrovni běhu aplikace. Přestože RASP není běžně implementován ve Flutter frameworku, může být využit pomocí nástrojů třetích stran nebo integrace s jinými bezpečnostními platformami. [39]

2. Opatrnost při používání knihoven a balíčků

   Při využití balíčků třetích stran je důležité pečlivě vybírat knihovny a balíčky od důvěryhodných zdrojů. Vývojáři by měli pravidelně kontrolovat

zabezpečení použitých knihoven a aktualizovat je na nejnovější verze s opravenými chybami a zranitelnostmi.

3. Zamezení ukládání dat v prostém textu na našem serveru. Místo toho zvažte použití iterované kryptografické hashovací funkce k hashovaní hesel uživatelů a poté ověření pomocí těchto hashovacích hodnot. Pokud server utrpí únik dat, hesla nezůstanou zcela odhalena. [11]

Komplexní ochrana aplikací vyžaduje statickou i dynamickou ochranu. Útočníci se neomezují pouze na statickou analýzu aplikací. Používají také různé nástroje k analýze a úpravě aplikace za běhu. Proto je také důležité používat řešení RASP, která detekují a odrazují útočníky od dynamické analýzy kódu za běhu, aby se obešli šifrovací a matovací techniky přidané do kódu. [39]

4.1.1 Šifrování dat v mobilních aplikacích

Šifrování dat je klíčovým prvkem zabezpečení v mobilních aplikacích, které chrání data před neoprávněným přístupem k souborům, kde jsou data v otevřeném textu. Tyto data mohou být citlivé informace, jako jsou osobní údaje, finanční transakce, či komunikační obsah.

Správná implementace šifrování a bezpečná správa šifrovacích klíčů poskytuje ochranu proti neoprávněnému přístupu, útokům a únikům dat. Kombinace šifrování během přenosu a uložení, spolu s používáním moderních šifrovacích algoritmů, může výrazně zvýšit bezpečnost mobilních aplikací a chránit citlivá data uživatelů. V případě, že ukládáme citlivá data do lokálního uložiště uživatele, musím brát zřetel na formát, ve kterém data ukládáme, a hlavně zda je vůbec musíme ukládat. Pakliže musíme ukládat tyto citlivé údaje, volí se metoda šifrování textu různými šiframi a nahrazeními, například funkcí hash.

4.1.2 Ochrana před různými druhy útoků

Mobilní aplikace jsou neustále vystaveny různým bezpečnostním hrozbám a útokům, které ohrožují jejich integritu, bezpečnost uživatelských dat a celkovou funkčnost. Mezi nejčastější typy útoků patří SQL injection, cross-site scripting (XSS) nebo session hijacking. Tyto útoky mohou vést k úniku citlivých informací, neoprávněnému přístupu nebo k narušení bezpečnosti aplikace.

Sociální inženýrství

Dalším druhem útoků jsou útoky směřující přímo na uživatele aplikací. Tyto útoky mají za cíl získat citlivé informace nebo provádět finanční transakce jménem uživatele. Útočníci často využívají sociálního inženýrství k oklamání uživatelů a přimění je ke stažení infikovaných aplikací nebo k poskytnutí osobních údajů.

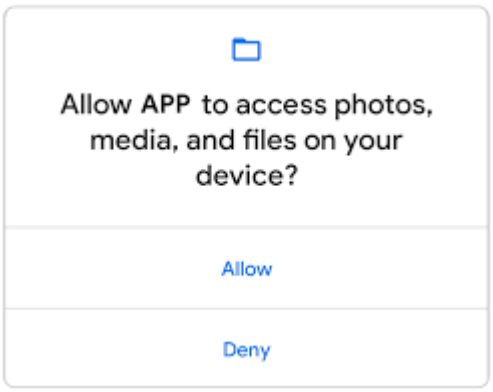
Drive-by

Drive-by útoky jsou dalším způsobem, jak mohou útočníci nainstalovat škodlivé aplikace bez vědomí uživatele. Jakmile je infikovaná aplikace nainstalována, útočníci mohou získat kontrolu nad uživatelským účtem a přístup k citlivým informacím.

Pro ochranu proti těmto útokům lze využít různé bezpečnostní nástroje a postupy. Softwarové nástroje, jako je například Mobile Security Framework (MobSF) [19], umožňují analýzu a testování bezpečnosti mobilních aplikací. Tyto nástroje mohou odhalit slabá místa a umožnit vývojářům implementovat opatření na jejich ochranu.

4.2 Správa oprávnění

Při vývoji aplikace pro Android je důležité zvážit, zda je pro funkčnost aplikace nezbytný přístup k omezeným datům nebo akcím. Pokud ano, vývojáři by měli deklarovat příslušná oprávnění a vyžadovat reakci od uživatelů. Android kategorizuje oprávnění na oprávnění k instalaci a oprávnění za běhu. Oprávnění k instalaci jsou automaticky udělena při instalaci, zatímco oprávnění za běhu vyžadují výslovné schválení uživatele za běhu. Vývojáři by se měli řídit pracovním postupem, který zahrnuje určení nezbytnosti oprávnění, jejich deklaraci a v případě potřeby vyžádání oprávnění k běhu. [23]



Obrázek 6 – Povolení o sdílení fotek a souborů v zařízení. Zdroj [23]

Oprávnění hrají klíčovou roli při zajišťování soukromí a bezpečnosti uživatelů v aplikacích pro Android. Pomáhají kontrolovat přístup k citlivým datům a akcím, jako je přístup ke stavu systému nebo nahrávání zvuku, a umožňují tak uživatelům mít kontrolu nad svými osobními údaji. Oprávnění Android jsou rozdělena do různých typů dle oficiální dokumentace Android [23], a to:

- Oprávnění v době instalace

  Tato oprávnění se udělují automaticky při instalaci a zahrnují normální oprávnění a oprávnění k podpisu. Normální oprávnění představují minimální riziko pro soukromí uživatele, zatímco oprávnění k podpisu jsou udělována pouze aplikacím podepsaným stejným certifikátem jako definující aplikace nebo OS.

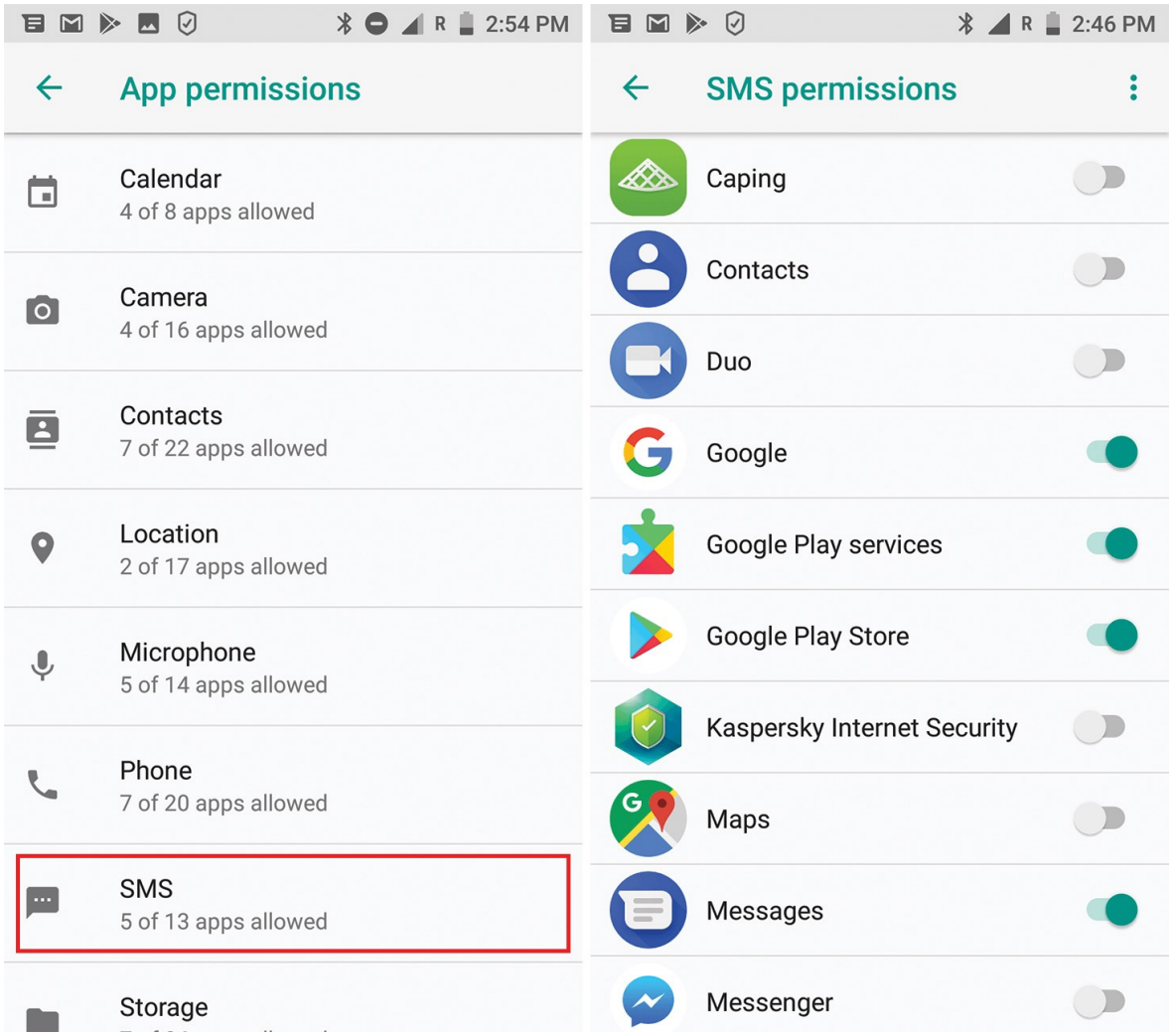
- Oprávnění za běhu

Také známá jako nebezpečná oprávnění, oprávnění k běhu vyžadují výslovné schválení uživatele za běhu. Tato oprávnění udělují přístup k citlivým údajům nebo akcím a zahrnují soukromá uživatelská data, jako je poloha a kontaktní údaje.

- Zvláštní oprávnění

Zvláštní oprávnění odpovídají konkrétním operacím aplikace definovaným platformou nebo výrobci OEM. Používají se k ochraně přístupu k výkonným akcím, jako je kreslení přes jiné aplikace, a obvykle se spravují v nastavení systému.

Uživatelé si pak mohou sami zpracovávat zacházení se jejich oprávněními v nastavení telefonu, kde lze vidět počet využití a druh oprávnění, jak lze vypozorovat z obrázku níže.



Obrázek 7 – přehled povolení na platformě Android. zdroj [23]

5 NATIVNÍ VÝVOJ APLIKACÍ

Publikace aplikací je finálním krokem v procesu vývoje softwaru, platí však pravidlo první verze. Aplikaci jako takovou musíme neustále aktualizovat a upravovat potřebám firmy nebo uživatelů. Publikace jako taková zahrnuje zveřejnění aplikace pro uživatele a umožňuje jim přístup k funkcím a obsahu, který jsme vytvořili. Tato kapitola se zaměřuje na proces publikace aplikací pro mobilní platformy Android a iOS. Při vývoji pro Android je nezbytné použít jazyk Java nebo Kotlin a využít nástrojů jako Android Studio, který je oficiálním vývojovým prostředím poskytovaným společností Google. Pro vývoj iOS je zde prostředí XCode a programovací jazyk Swift. Čerpám zde především ze zdrojů oficiálních dokumentací, které sepsaly vývojáři a obsahují detailní popis architektury. [32]

5.1 Vývoj pro Android

Pro vývoj aplikací pro Android se často používá programovací jazyk Java nebo Kotlin.

Dalšími populárními technologiemi jsou Android SDK a Android Studio. Vývoj aplikací pro Android je obvykle prováděn v Android Studiu, což je integrované vývojové prostředí poskytované společností Google, obsahuje různé nástroje pro tvorbu, ladění a testování aplikací pro Android. Android má větší fragmentaci, což znamená, že existuje mnoho různých verzí systému a zařízení s různými displeji, výkonem atd.

Android Open System Platform (AOSP)

AOSP je veřejně dostupný a upravitelný zdrojový kód Androidu. Slouží jako základ pro vytváření Androidem založených systémů a zařízení. Vývojáři si mohou stáhnout a upravit AOSP podle svých konkrétních požadavků. [32]

Softwarový stack pro AOSP se skládá z několika vrstev, které vidíme i na obrázku č.3 níže:

1. Androidová aplikace

   Distribuovaná prostřednictvím platforem jako Google Play Store nebo předinstalovaná na zařízeních výrobcem.

2. Privileged aplikace

   Využívá kombinaci Android a systémových API. Musí být předinstalována jako privilegované aplikace na zařízeních.

3. Aplikace výrobce zařízení

   Předinstalována na zařízeních a aktualizována s aktualizacemi systémového softwaru.

4. Systémové API

   Poskytuje Android API dostupné pouze partnerům a OEM. Označeno jako @SystemApi ve zdrojovém kódu.

5. Android framework

   Obsahuje Java třídy, rozhraní a předkompilovaný kód pro tvorbu aplikací.

6. Systémové služby

   Usnadňují komunikaci mezi Android framework API a hardwarem.

7. Androidové spuštění (ART)

   Překládá byte kód aplikace do instrukcí specifických pro procesor.

8. Hardware Abstraction Layer (HAL)

   Umí umožňuje Androidu být nezávislý na implementaci nižší úrovně.

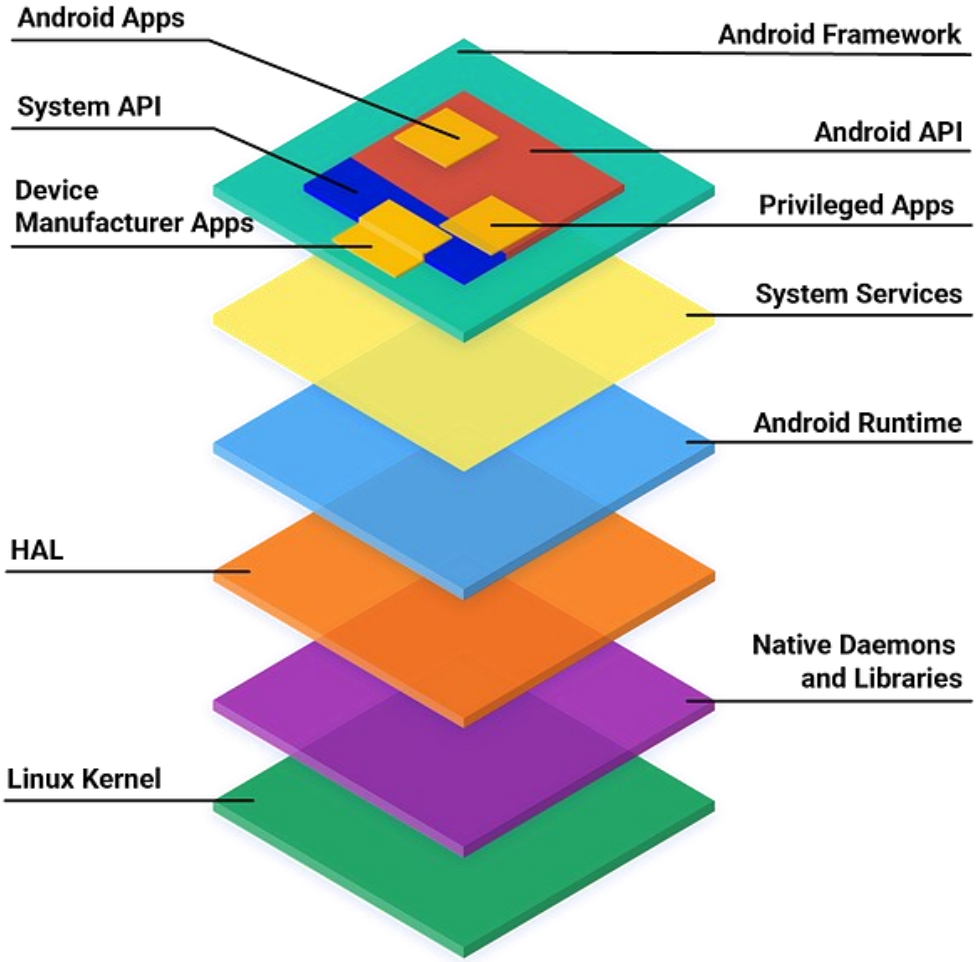
9. Nativní funkce a knihovny

   Zahrnuje nativní démony jako init, healthd, logd a storaged. Interagují přímo s jádrem nebo jinými rozhraními.

10. Jádro

    Střední část operačního systému, která komunikuje s podkladovým hardwarem.

Toto jsou jednotlivé vrstvy, které mezi sebou komunikují a které jsou viditelné na obrázku níže. Tyto vrstvy nabízejí abstrakci různých úrovní, Android je založen na Linux jádře, tudíž obsahuje zabezpečení v podobě povolení a správy disku a celkovou správu systému, kterou ze stáhnuté aplikace Android nezobrazíme – vrchní vrstva na obrázku č.8.



Obrázek 8 - Softwarový stack pro AOSP [32]

5.2 Vývoj pro iOS

iOS se zaměřuje na vytváření aplikací pro zařízení jako iPhone, iPad a iPod Touch, které běží na operačním systému iOS od společnosti Apple. Pro vývoj pro iOS je nezbytné použít programovací jazyk Swift nebo Objective-C a využít vývojové prostředí Xcode, které je poskytováno společností Apple.

NeXT, společnost založená Stevem Jobsem po jeho odchodu z Apple, představila NeXTSTEP, operační systém a vývojové prostředí postavené na objektově orientovaném programování. NeXTSTEP byl klíčový v tom, že nabízel programovací jazyk Objective-C, který umožňoval kombinovat objektově orientované myšlení s výkonností C. NeXT použil Objective-C k vybudování rozsáhlých knihoven objektů, což umožnilo vývojářům vytvářet grafické uživatelské rozhraní a aplikace rychleji než tradičními postupy. [19]

Programovací jazyk Objective-C byl klíčový pro vývoj softwaru, byl to nástupce jazyka C, který byl sám o sobě dost rychlý, protože se kompiloval přímo do strojového kódu. Byl to hybridní jazyk kombinující objektově orientované prvky Smalltalku s výkonem C, což umožňovalo vývojářům vytvářet rychlé a efektivní aplikace. Kromě toho Objective-C umožňoval přístup k rozsáhlým knihovnám C. V roce 2014 Apple představil nový programovací jazyk Swift, který měl postupně nahradit Objective-C jako primární jazyk pro vývoj aplikací pro iOS a OS X. Swift je moderní, bezpečný a rychlý jazyk, navržený tak, aby byl jednodušší a bezpečnější než Objective-C. [19]

Jelikož by bylo programování těchto jazyků bez integrovaného vývojového balíčku náročné, společnost Apple vyvinula prostředí XCode, kde mohou vývojáři vyvíjet aplikace.

Xcode je integrované vývojové prostředí (IDE) poskytované společností Apple pro vývoj aplikací pro jejich operační systémy, zejména pro iOS, macOS, watchOS a tvOS. S výkonnými nástroji a bohatými sadami funkcí je Xcode nezbytným prvkem pro vývojáře, kteří chtějí vytvářet aplikace pro Apple ekosystém. Poskytuje plně vybavený editor kódu s funkcemi jako syntax highlighting, code completion a integrací s verzovacími systémy, jako je například Git. [22]

5.3 Publikace aplikací

Publikace aplikací je klíčovým krokem v procesu vývoje mobilních aplikací, který umožňuje uživatelům přístup k výslednému produktu. Tato kapitola se zaměřuje na proces publikace aplikací, včetně podmínek pro publikaci kódu a specifického postupu pro publikování aplikací pro platformy Android a iOS. Platformy jako Google Play Store pro Android a Apple App Store pro iOS provádějí před publikací důkladné kontroly a certifikace, aby zajistily, že aplikace splňují stanovené standardy kvality, bezpečnosti a ochrany soukromí. Tím se minimalizuje riziko pro uživatele v podobě škodlivého obsahu nebo nevhodného chování aplikace. Přehled těchto standardů doporučuji hledat z oficiální dokumentace vývojářů, jelikož se průběžné aktualizují a mění. [33]

5.3.1 Podmínky pro publikaci kódu

Před publikací jakékoliv aplikace je nezbytné dodržovat určité podmínky stanovené platformou, na kterou je aplikace určena. Tyto podmínky se mohou lišit v závislosti na konkrétní platformě a mohou zahrnovat technické, obsahové a právní požadavky. Mezi běžné podmínky pro publikaci kódu patří dodržování autorských práv, absence škodlivého obsahu a dodržování technických specifikací platformy.

5.3.2 Publikování Android aplikace

Publikování aplikace na platformě Android probíhá prostřednictvím Google Play Store, který je oficiálním distribučním kanálem pro Android aplikace. Pro publikaci aplikace na Google Play Store je nezbytné vytvořit vývojářský účet, získat digitální podpis aplikace, nastavit metadata a popis aplikace a následně aplikaci nahrát do obchodu. Po nahrání je aplikace podrobená kontrolám Google Play Store a pokud splňuje všechny požadavky, je zveřejněna a dostupná pro uživatele.

Pokud chceme publikovat aplikaci Android na Google Play Store, musíme odeslat pouze soubory formátu AAB. Soubory APK již nejsou podporovány. Soubor AAB vytvoříme například pomocí vývojářského nástroje Android Studio. Tento soubor nabízí mnoho výhod – balíček AAB je optimalizovaný, takže má menší velikost souboru a menší velikost při aktualizaci aplikace. [24]

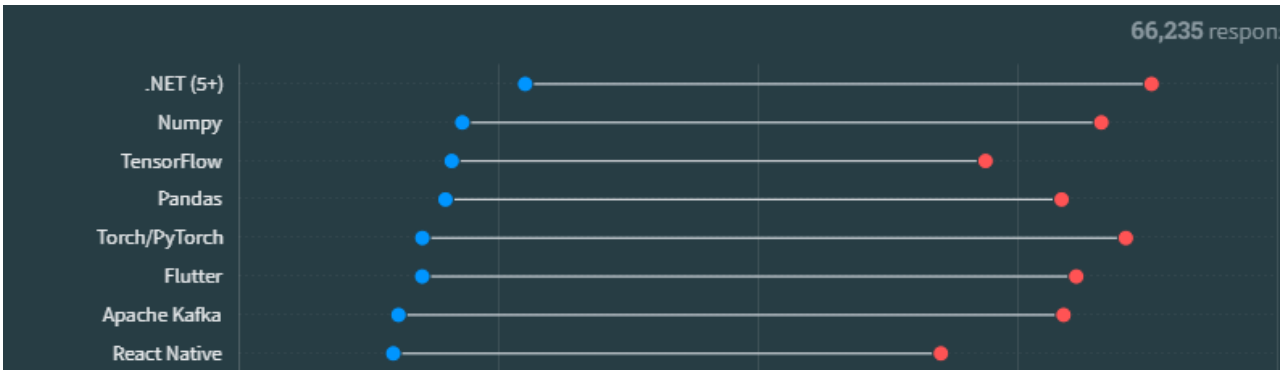
5.3.3 Publikování iOS aplikace

Publikování aplikace na platformě iOS probíhá prostřednictvím App Store, který je oficiálním distribučním kanálem pro iOS aplikace. Pro publikaci aplikace na App Store je nezbytné vytvořit vývojářský účet, vyvinout aplikaci v souladu s Apple App Store Guidelines, získat certifikát a identifikátor aplikace od Apple, vytvořit metadata a popis aplikace, a nakonec aplikaci nahrát do App Store. Apple provádí kontrolu aplikace a pokud splňuje všechny požadavky, je schválena a zveřejněna v App Store pro uživatele. [33]

6 MULTIPLATFORMNÍ VÝVOJ APLIKACÍ

Flutter je sada nástrojů UI napříč platformami, která je navržena tak, aby umožňovala opětovné použití kódu napříč operačními systémy, jako jsou iOS a Android, a zároveň to umožňuje aplikace pro přímé propojení se základními službami. Pro vydání jsou aplikace Flutter kompilovány přímo do strojového kódu, ať už jde o architekturu procesoru Intel x64 nebo ARM, popřípadě JavaScript, pokud jsou cíleny na web. Framework je open source, s povolenou licencí BSD a má prosperující ekosystém balíčků třetích stran, které doplňují základní funkce knihovny.

Tomuto nástroji náleží majoritní část programátorů, kteří pomocí něj vyvíjejí, a to 64,43 % jako oblíbených a 14 % v kategorii žádaných pro naučení. Statistická data čerpám ze Stack Overflow dotazníku určeného pro vývojáře, zobrazující uživatelské preference, jak lze vidět i na obrázku. [28]



Obrázek 9 – preference uživatelů StackOverflow. Zdroj [27]

Framework

Vzhledem k tomu, že ve své práci dosti používám slovo framework, je vhodné věnovat kapitolu právě významu tohoto slova. Framework je sada před-integrovaných funkcí a nástrojů, jejichž účelem je usnadnit práci programátorovi. Funguje jako šablona, kterou lze použít, a dokonce upravit tak, aby splňovala požadavky projektu. Obvykle jsou spojeny s určitým programovacím jazykem a jsou vhodné pro různé typy úloh.

Benefity použití

- Umožňuje vývojářům snadno pracovat v daném programovacím jazyce (i složitém).

- Jsou vysoce flexibilní a škálovatelné, to znamená, že umožňují jednoduše rozšiřovat projekt a pracovat na něm simultánně v týmech.
- Lze snadno integrovat téměř cokoli pomocí již vestavěných rozhraní API.

Nevýhody spjaté s použitím frameworku

- Kvůli komunitnímu rozvoji a specifickým požadavkům se frameworky neustále vyvíjejí a poskytují nové verze, což nemusí být pro firemní potřeby ideální, jelikož musejí kód neustále udržovat. Alternativou je tvorba vlastní knihovny pro tvorbu aplikací

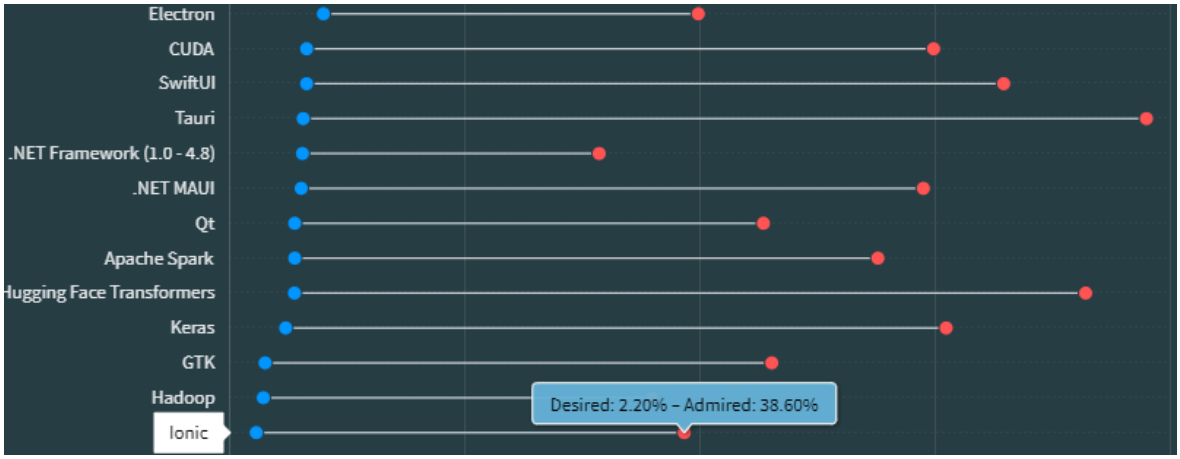
Subjektivně vnímám framework jako obří výhodu v kontextu vývoje, jelikož obsahuje spoustu nástrojů a implementovaných funkcí, které vývojáři ušetří čas a poskytne jistý komfort. Flutter je navíc specifický v překladu jednoho kódu do nativní platformy, ať už se jedná o mobilní, webové nebo televizní zařízení, čemuž bych musel investovat třikrát více času pro psaní stejného kódu, leč jinou platformu.

6.1 Framework Ionic

Aplikace Ionic jsou vytvořeny pomocí technologií HTML, CSS a JavaScript. S Ionic můžeme nasadit nativní aplikaci pro iOS nebo Android, nativní desktopovou aplikaci nebo webovou aplikaci pomocí jednoho kódu stejně jako Flutter. Při běhu na mobilu běží Ionic v nativním kontejneru pomocí Cordova nebo *Capacitor*, které umožňují plný přístup k jakýmkoli nativním funkcím zařízení nebo API. [28]

Všechny komponenty uživatelského rozhraní, které používáme v aplikaci Ionic, používají standard Web Components, takže jsou kompatibilní v jakémkoli webovém prohlížeči a jsou kompatibilní s jakýmkoliv frameworkem Javascriptu, včetně React, Vue a Angular, dle oficiální dokumentace Ionic [28].

Ionic poskytuje knihovnu více než 100 komponent uživatelského rozhraní, které můžeme dále měnit. Dle preferenčních statistik uživatelů, kteří vyplňovali dotazník se spokojeností nebo využíváním dané technologie, je Ionic méně oblíbenější oproti frameworku Flutter, avšak více než Xamarin, jak jde vidět na obrázku 10 níže.



Obrázek 10 Preference uživatelů frameworku Ionic z webu Stack Overflow [27]

6.2 Framework Flutter

V dnešní době existuje spousty nástrojů, díky kterým můžeme docílit stejného výsledku, někdy však za pomoci méně práce než s jinými nástroji. Mezi tyto nástroje, které plní svou úlohu a zefektivňují čas vývojářů patří Flutter, který umožňuje spustit jeden kód na několika cílových platformách. Odlišnost frameworku Flutter je vysoce výkonný renderovací engine, který je postaven na vrcholu grafické knihovny Skia. Tento renderovací engine umožňuje vytvářet plynulé, vlastní animace a komplexní rozvržení.

V tradiční HTML aplikaci je za vykreslování uživatelského rozhraní aplikace zodpovědný vykreslovací modul prohlížeče, jako je WebKit nebo Blink. Ve Flutter se uživatelské rozhraní aplikace vykresluje pomocí vlastního vykreslovacího enginu zmíněného výše. Každá aplikace v rámci Flutter frameworku je spuštěna vlastním procesem, který obsahuje jedno hlavní vlákno. To znamená, že ve výchozím nastavení všechny komponenty aplikace běží v tomto jednom vlákně, nazývaném také vlákno uživatelského prostředí (UI thread). Pokud je spuštěna nová komponenta aplikace, která má již existující proces, je spuštěna v tomto existujícím hlavním vlákně. Toto chování může být změněno explicitně programátorem, například nastavením komponenty tak, aby běžela ve vlastním procesu, tyto nazýváme dle oficiální dokumentace Flutter Isolates. [16]

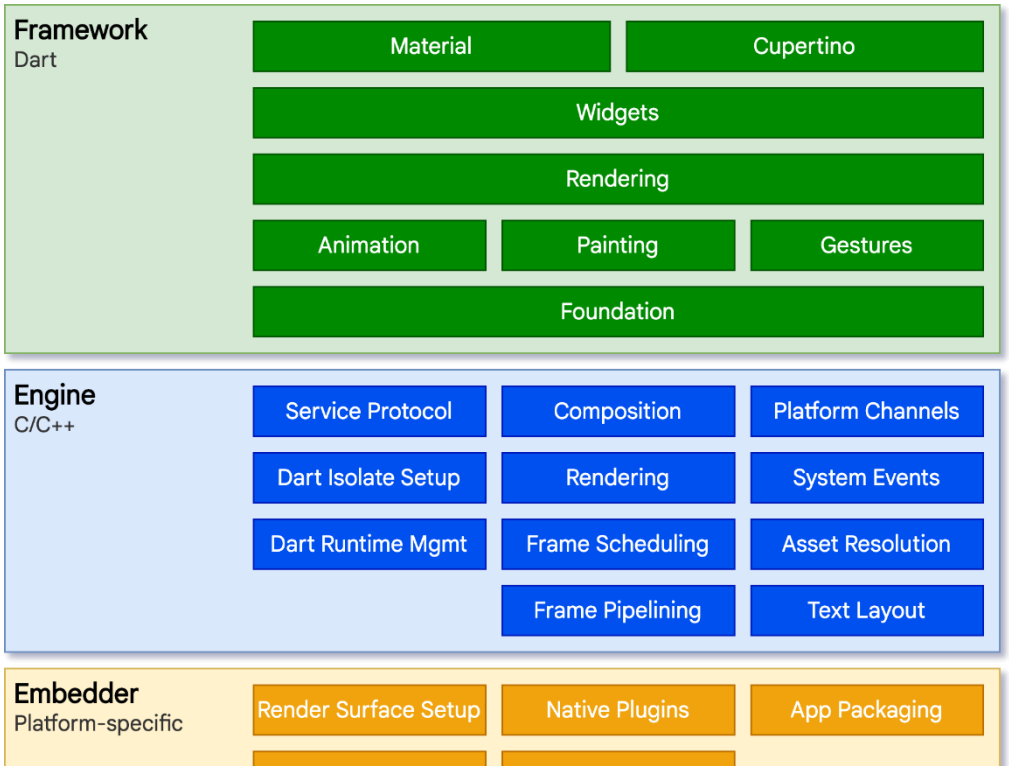
Při vytváření pro mobily používá Flutter kompilátor Dart k převodu vašeho kódu Dart na nativní strojový kód, který poběží na platformě zařízení, spolu s vlastním vykreslovacím

modulem pro zobrazení vašeho uživatelského rozhraní v mobilní aplikaci. Flutter nepoužívá nativní prvky uživatelského rozhraní, které můžeme najít v React Native, ani nepoužívá webové komponenty jako Ionic, kterému bych rád věnoval více, protože je konkurencí, nebo snad spíše alternativou hybridního mobilního vývoje, a tuto práci bych rád vedl objektivně.

Flutter používá kompilátor Dart ke generování JavaScriptu, který se vykreslí v prohlížeči. Poté nahradí grafický engine založený na Skia a vykreslování textu vlastním vykreslovacím modulem a poté znovu sestaví základní primitiva webového prohlížeče, jako je výběr textu, kopírování, vkládání a dostupnost od začátku. Každá aplikace, kterou vytvoříte, pak dodává celá tato přestavěná primitiva.

6.3 Architektura frameworku Flutter

V jádru této knihovny je engine, který je převážně napsán v jazyce C++ a podporuje primitiva nezbytná pro všechny Flutter aplikace. Engine je odpovědný za rasterizaci složených scén pokaždé, když je třeba namalovat nový snímek. Poskytuje nízko úrovňovou implementaci jádra API Flutter, včetně grafiky (pomocí Impeller na iOS a přichází na Android, a Skia na ostatních platformách), textového rozložení, I/O souborů a sítě, podpory dostupnosti, architektury pluginů a běhového prostředí Dart s nástroji pro kompilaci. [13]



Obrázek 11 Architektura Flutter. Zdroj [13]

Dart zahrnuje bohatou sadu platformových, rozvržení a základních knihoven, složených ze série vrstev, zobrazených a popsaných na obrázku níže.

1. Základní třídy a služby stavebních bloků, jako jsou animace, malování a gesta, které nabízejí často používané abstrakce nad podkladovým základem.
2. Renderingová vrstva poskytuje abstrakci pro práci s rozložením. S touto vrstvou můžete vytvářet strom renderovatelných objektů a dynamicky s nimi manipulovat, přičemž strom automaticky aktualizuje rozložení k odrazu vašich změn.
3. Vrstva widgetů je abstrakce kompozice. Každý renderovací objekt v renderingové vrstvě má odpovídající třídu ve vrstvě widgetů. Kromě toho vrstva widgetů umožňuje definovat kombinace tříd, které můžete znovu použít. Tato vrstva představuje úvod do reaktivního programovacího modelu.
4. Knihovny Material a Cupertino nabízejí komplexní sady ovládacích prvků, které využívají kompozičních primitiv vrstvy widgetů k implementaci jazyků designu Material nebo iOS
5. Engine je propojen s Flutter frameworkem přes dart:ui, který obaluje podkladový C++ kód do tříd Dart. Tato knihovna vystavuje nejnižší úroveň primitiv, jako jsou třídy pro ovládání vstupu, grafiky a textových podsystémů. Strukturu architektury a jednotlivých vrstev můžeme vidět na obrázku níže. Informace včetně obrázku čerpám z oficiální dokumentace. [13]

6.4 State management

Jedním z klíčových aspektů vytváření efektivních a udržitelných aplikací je správa stavu. Správná správa stavu aplikace zajišťuje konzistentní a plynulý uživatelský zážitek bez zbytečných chyb nebo zpoždění. V prostředí Flutteru, moderního multiplatformního frameworku vyvinutého společností Google, je stavový management kritickým aspektem vývoje aplikací. Stav řízení v aplikacích ve Flutteru se týká efektivního způsobu, jak uchovávat, aktualizovat a synchronizovat stav aplikace mezi uživatelským rozhraním a daty.

Existuje mnoho řešení pro řízení stavu aplikace a rozhodnutí, které z nich použít, může být nelehký úkol. Neexistuje jedno dokonalé řešení, protože se liší v případech užití, v náročnosti a komplexitě.

6.4.1 SetState

Základním mechanismem pro řízení stavu ve Flutteru je použití metody setState(). Tato metoda umožňuje aktualizovat stav widgetu a vyvolat překreslení uživatelského rozhraní. SetState je vhodné použít při správě stavu jednoduchých aplikací a widgetů s malým množstvím stavových dat. Stav je objekt, který uchovává data a logiku widgetu, a je přímo spojen s životním cyklem widgetu. Kdykoliv je potřeba stav aktualizovat, používá se metoda *setState*, která překreslí widget s novými daty. [16]

6.4.2 Provider

Provider je knihovna pro správu stavu vyvinutá komunitou Flutteru, která umožňuje sdílet a aktualizovat stav mezi widgety aplikace. Provider implementuje koncept "InheritedWidget", což umožňuje efektivně předávat stav mezi widgety bez nutnosti ručního předávání dat. Tento přístup je vhodný pro středně velké až velké aplikace s komplexními hierarchiemi widgetů. Jedná se formu poskytování závislosti inicializováním ve vrchním místě aplikace a používáním kdekoliv, ve světě programování se to nazývá *Dependency Injection*. Je to návrhový vzor, který umožňuje snadnou správu závislostí mezi jednotlivými částmi aplikace. Jedná se o koncept, který pomáhá oddělit vytváření a konfiguraci objektů od jejich použití. Benefitem je zvýšení modularity, testovatelnosti a znovu-použitelnosti kódu. Díky tomu můžeme abstraktně oddělit jednotlivé části a propagovat jednotlivé instance napříč celou aplikací, není tedy třeba na různých místech vytvářet nové instance nebo odkazovat na existující.

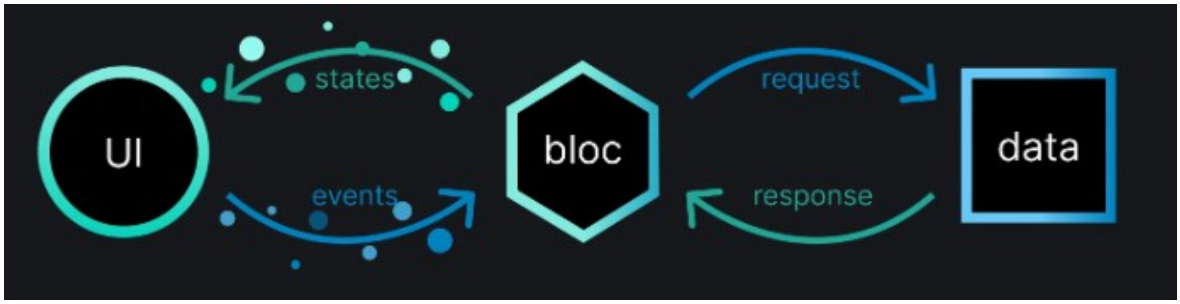
6.4.3 Bloc (Business Logic Component)

Bloc je architektonický vzor pro správu stavu v aplikacích ve Flutteru, který odděluje obchodní logiku od uživatelského rozhraní. Bloc používá streamy pro předávání událostí a stavů mezi různými částmi aplikace. Tento přístup je ideální pro velké a komplexní aplikace s mnoha interaktivními prvky a datovými toky. Zde bych se rád pozastavil a vysvětlit koncept této knihovny pro správu stavů, jelikož ji sám preferuji při vývoji aplikací.

6.5 Bloc Pattern

Bloc (Business Logic Component) je návrhový vzor, který se často používá v aplikacích vyvíjených ve frameworku Flutter nebo React. Jeho hlavním účelem je oddělení logiky od uživatelského rozhraní. To umožňuje lépe organizovat kód, zlepšuje jeho čitelnost a udržitelnost, a usnadňuje testování. Tento vzor v praxi rozděluje jednotlivé části, které bych rád zmínil, a to ze zdroje oficiální dokumentace vývojářů [13]:

- **Eventy** neboli události, které ovlivňují stav aplikace. To může být například stisk tlačítka nebo načtení dat z databáze.
- **State** neboli stav aplikace v určitém okamžiku. Může to být například načítání, získání dat nebo potažmo chyba.
- **Bloc**, objekt, který přijímá události, provádí na základě nich logiku a emituje nové stavy. Zpravidla je implementován jako stream, který produkuje sekvenci stavů aplikace.



Obrázek 12 – Architektura BLoC. Zdroj [13]

Tento návrhový vzor můžeme rozdělit dle případu užití na dvě třídy, a to je Bloc a Cubit. Oba se používají pro řízení stavu aplikace, ale existují mezi nimi určité rozdíly, zejména v jejich úrovni abstrakce a složitosti. Cubit může implementovat funkce, které lze vyvolat ke spuštění změn stavu, ty jsou následně výstupem a představují návratovou hodnotu stavu aplikace. Komponenty uživatelského rozhraní mohou být informovány o stavech a překreslit jejich části na základě aktuálního stavu. Při vytváření Cubitu musíme definovat typ stavu, který bude Cubit navracet, může se jednat o formát datumu, nastavení vzhledu, zkrátka jakoukoliv nenulovou návratovou hodnotu.

V obou případech musíme při vytváření zadat počáteční stav. Můžeme to udělat voláním super s hodnotou počátečního stavu, jak zobrazuje kód na obrázku níže. [27]

```dart
class CounterCubit extends Cubit<int> {
  CounterCubit() : super(0);

  void increment() => emit(state + 1);
}
```

Obrázek 13 –Nastavení počátečního stavu Bloc

Jak jsem uváděl v kapitole dříve, změny stavů můžeme implementovat pomocí funkcí, jak jde vidět na obrázku nahoře. V tomto případě se při každém zavolání této funkce implicitně nastaví vnitřní stav aplikace nejdříve na počáteční stavovou hodnotu a poté se inkrementálně mění. Právě tato separace od view a posléze dat nám umožňuje mít kód čitelný a v případě změn nebo škálování stačí přidávat stavy a eventy.

Balíček bloc nenabízí pouze správu stavů, ale i testování díky tomuto vzoru, čemuž se budu více do detailu věnovat v kapitole testování mobilních aplikací.

6.6 Widgety

Widgety popisují, jak by měl jejich pohled vypadat vzhledem k jejich aktuální konfiguraci a stavu. Když se změní stav widgetu, widget například změní barvu, Flutter jej porovná s předchozím popisem, aby určil minimální změny potřebné v podkladovém stromu renderu pro přechod z jednoho stavu do druhého. Ve webových technologiích, například React využíváme virtuální dokument obsahující strukturu elementů, lze tedy vidět inspiraci mechanismu fungování mezi těmito dvěma frameworky. [16]

Widgety si můžeme představit jako kostky lega, přičemž každá je unikátní, a přesto vytváří funkční celek. Flutter staví na konceptu "všechno je widget". Widgety jsou základními stavebními bloky uživatelského rozhraní ve Flutteru, můžeme si ji představit jako rozsáhlou sadu vestavěných widgetů pro různé účely, jako jsou textová pole, tlačítka, seznamy, obrázky a mnoho dalšího. Kromě toho můžeme vytvářet vlastní widgety nebo kombinovat existující widgety pro vytváření složitějších uživatelských rozhraní, jsme tedy schopni stejně jako z lega vytvořit cokoliv. Jak jsem uváděl dříve ve State management kapitole 4.1.2, widgety mohou být stavové nebo bez-stavové – tedy statické. Princip těchto dvou implementací je oddělit statické komponenty, které můžeme načíst již při zapínání aplikace, od dynamických, které mají různé stavy a mění svůj obsah v průběhu aplikace.

Uvedu menší seznam widgetů, které používám v aplikaci, téměř vždy pro tvorbu vlastních widgetů, inspirací je mi vzor znuvupoužitelnosti, který nabádá k vytváření znovupoužitelných generických widgetů, do kterých posíláme parametry v podobě dat, které chceme vykreslit.

6.6.1 Container

Container ve Flutteru funguje jako krabicový model a pomáhá vývojářům vytvořit strukturované a stylizované uživatelské rozhraní. Zahrnuje širokou škálu vlastností, jako je výplň, okraj, výška, šířka a dekorace, což umožňuje rozsáhlé přizpůsobení prostoru a vzhledu prvků uživatelského rozhraní, kterým je rodičem. Úpravou těchto vlastností mohou vývojáři dosáhnout přesného zarovnání a umístění, čímž se zvýší celková estetika a funkčnost aplikace.

Mezi běžné případy použití widgetu Kontejner umožňuje vytváření vizuálně odlišných sekcí v aplikaci, přidávání odsazení kolem widgetů a přizpůsobení pozadí pomocí barev nebo obrázků.

6.6.2 Text

Widget Text je jednou ze základních komponent Flutter [16], zásadní pro zobrazení textu v aplikaci. Nabízí různé možnosti typografie, což umožňuje vývojářům upravit velikost písma, barvu, váhu a mezery, aby vytvořili požadovaný vizuální dojem. Lze také ovládat zarovnání a přetečení textu, což zajistí, že text dokonale zapadne do rozvržení a vylepší uživatelské rozhraní.

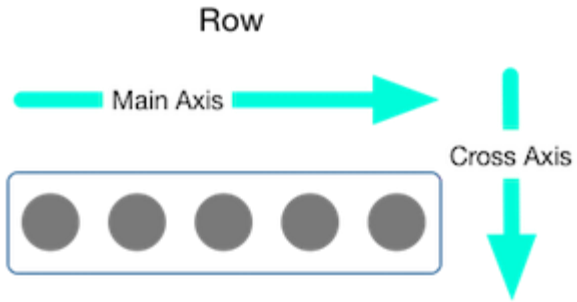
Mezi běžné aplikace widgetu Text patří informování uživatele, sbírání uživatelského vstupu a poskytování pokynů v rámci aplikace.

```
Text("Atlas mraků", style: kHeadline),
```

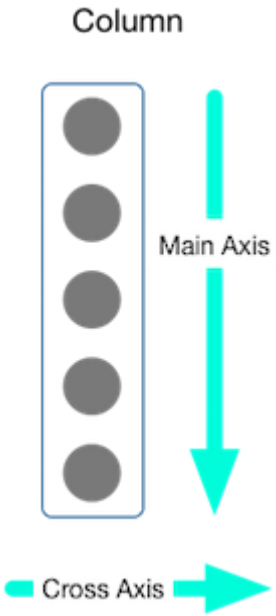
Obrázek 14 – Ukázka jednoduché implementace komponenty textu

6.6.3 Row a Column widget

Tyto dva widgety jsou nezbytné pro návrh horizontálního a vertikální rozvržení, jak už napovídá samotný název komponent. Pomáhá zarovnat widgety vedle sebe nebo pod sebou



Obrázek 16 – Komponenta Row. Zdroj [16]

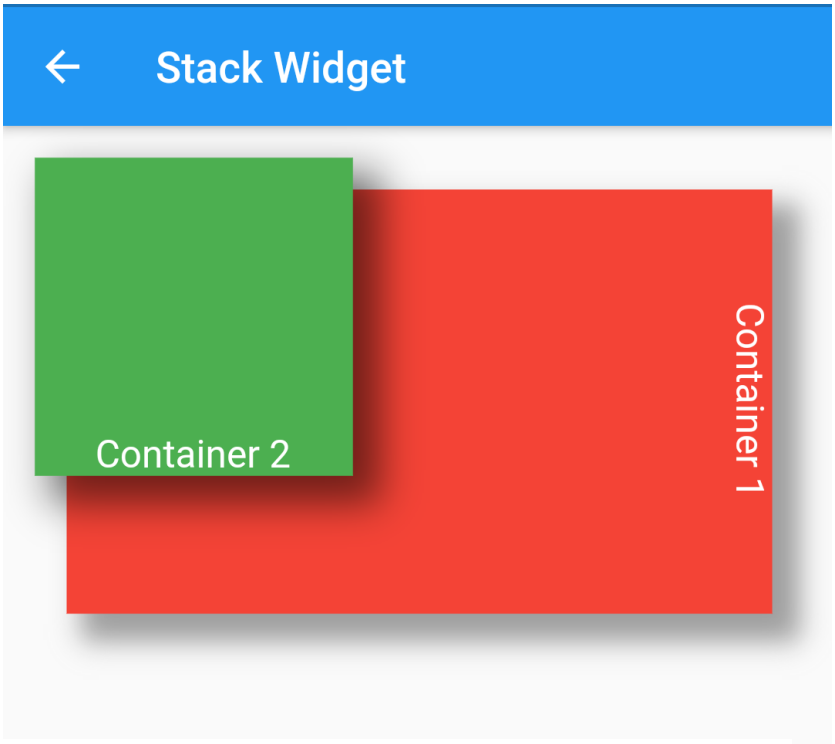


Obrázek 15 – Komponenta Column. Zdroj [16]

přes hlavní osu. Poskytuje vlastnosti jako *mainAxisAlignment* a *crossAxisAlignment* pro řízení zarovnání svých potomků a zajišťuje citlivé a dobře strukturované rozvržením. Schéma rozvržení lze vypozorovat z obrázků níže. [16]

6.6.4 Stack widget

Stack představuje jedinečný přístup k návrhu rozvržení ve Flutteru a umožňuje vývojářům překrývat widgety přes sebe a vytvářet strukturu jednotlivých vrstev. To se ukazuje jako neocenitelné pro scénáře vyžadující absolutní umístění nebo překrývání mezi widgety, jak lze vidět na obrázku níže. Tuto funkcionalitu u sebe v aplikaci využívám často v kombinaci překrývání obsahu stránky a pozadí aplikace, kdy je spodní vrstva tvořena právě z widgetu obsahujícího gradient barvy, nad níž je vykreslen obsah v podobě nadpisů, mapy, karet. Základní znalost a popis komponenty čerpám ze oficiální dokumentace. [16]



Obrázek 17 –Stack komponenta umožňující překrývaní komponent

6.6.5 SingleChildScrollView

Někdy je rozvržení navrženo kolem flexibilních vlastností sloupce, ale existuje možnost, že v některých případech nemusí být dostatek místa pro zobrazení celého obsahu. Může to být způsobeno tím, že některá zařízení mají neobvykle malé obrazovky, nebo protože aplikaci lze používat v režimu na šířku, kde poměr stran neodpovídá původní představě, nebo protože se aplikace zobrazuje v malém okně v režimu rozdělené obrazovky. V každém případě by

```
SingleChildScrollView(
      child: Padding(
        padding: const EdgeInsets.symmetric(
          horizontal: 20,
          vertical: 20,
        ),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: const [
            Text("Atlas mraků", style: kHeadline),
            SizedBox(
              height: 600,
              child: AtlasList(),
            ),
          ],
        ),
      ),
    );
  }
```

Obrázek 18 – Komponenta SingleChildScrollView

ve výsledku mohlo mít smysl zabalit rozvržení do SingleChildScrollView. Jak ale na první pohled nelze vidět, obvykle to vede ke konfliktu mezi widgetem *Column*, sloupcem, který využívá dostupného místa, jak jen může, a SingleChildScrollView, který poskytuje svým potomkům nekonečné množství prostoru. Právě tento konflikt vyústí v přetečení obsahu

obrazovky mimo viditelnou obrazovku. Níže demonstruji ukázku využití komponenty pro zobrazení obsahu, který může být větší, než je obrazovka uživatele, a umožňuje tak posouvat vertikální obsah. Potomkem je sloupec obsahující nadpis a list určený pro výpis atlasu mraků, což je vlastní komponenta. Díky parametrům sloupce můžeme nastavit zarovnání

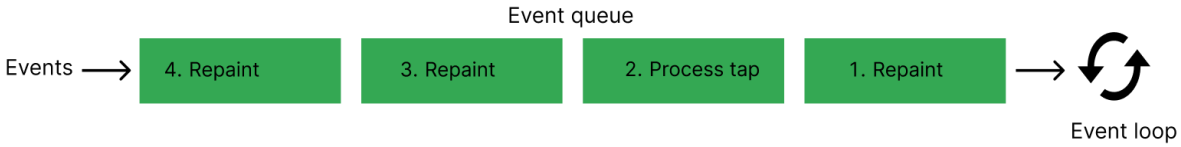
protější osy – tedy horizontu, nastavit zarovnání doleva pomocí třídy CrossAxisAlignment.start. Asynchronní programování jazyka Dart

Asynchronní operace nám umožňují dokončit další práci, zatímco čeká na dokončení jiné operace. Mezi asynchronní operace můžeme zařadit například:

- Načítání dat přes síť
- Zápis do databáze
- Čtení dat ze souboru

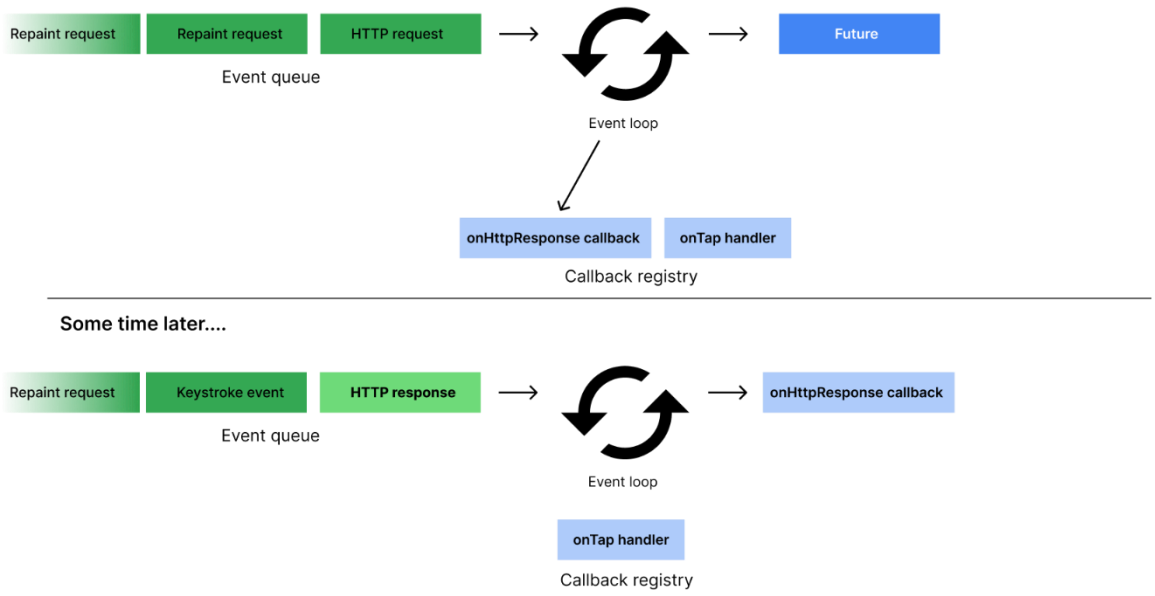
Skvěle rozepsaná je dokumentace od vývojářů jazyka Dart [12], ze které budu čerpat pro zbytek kapitoly. Jako první chci uvést, že se veškerý děj odehrává jako ve většině jiných programů v metodě main.

Flutter operuje na jednom vlákněs, která do smyčky sbírá eventy a vrací stavy – state, jak lze vidět na schématu níže. Během běhu aplikace se všechny události přidávají do fronty, která se nazývá fronta událostí. Události mohou být cokoli od požadavků na překreslení uživatelského rozhraní, po klepnutí uživatele a stisknutí kláves až po čtení nebo zápis disku.



Obrázek 19 – Smyčka událostí [12]

Za zmínku stojí oddělit paralelismus od nynějšího fungování, kdy při tomto jevu jádro zpracovává několik procesů nebo vláken simultánně. Jak jsem ale zmiňoval dříve, Flutter operuje pouze na jednom vlákně, takže zde dochází k iluzi paralelismu, ačkoliv se o něj nejedná. Během asynchronní volání se provádí synchronní kód, takže nejsme omezeni rychlostí přijetí odpovědi, pokud to však pro další implementaci nechceme. Schéma níže zobrazuje registrování asynchronní funkce a přidání do smyčky eventů, a následnému zařazení funkce zpět pro dokončení s výsledkem operace.



Obrázek 20 – Smyčka událostí – asynchronní operace [12]

Může se ale stát, že nastane situace, kdy budeme potřebovat provést simultánně několik metod, například API volání k cílovému serveru, a nevíme, kdy a jakou odpověď získáme. Proto se ve vlákně pouze odešle požadavek a přesune se na pozadí, dokud nedorazí response ze serveru

Funkce Future představuje výpočet bez příslibu okamžitého výsledku. Tam, kde normální funkce vrátí výsledek, asynchronní funkce vrátí datový typ Future, který můžeme z webových technologií přirovnat k premisám. Každá taková funkce musí být označena klíčovým slovem async a asynchronní operace označena jako await.

```
Future<List<Forecast>> getDailyForecast(double lat, double lon) async {
  final response =
      await http.get(Uri.parse('$apiUrl/data/forecast/daily/$lat/$lon'));

  if (response.statusCode == 200) {
    final List<dynamic> jsonList = json.decode(response.body);
    return jsonList.map((json) => Forecast.fromJson(json)).toList();
  } else {
    throw Exception('Failed to load daily forecast');
  }
}
```

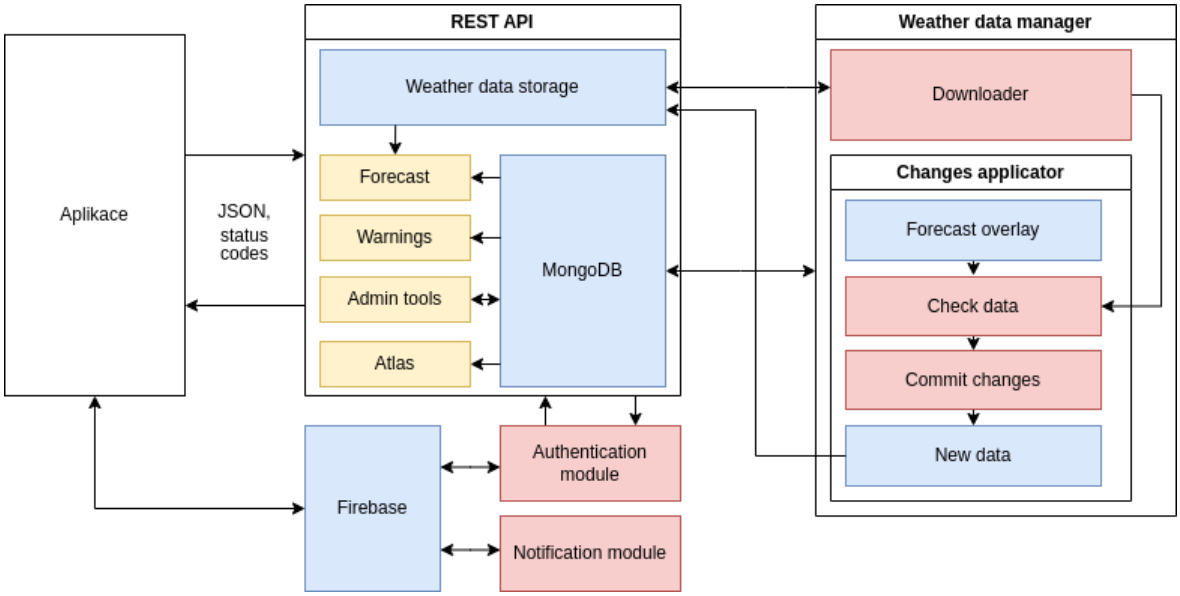
Obrázek 21 – Funkce pro získání listu počasí ze serveru

Tento kód je funkce, která slouží k získání denní předpovědi počasí na základě zadaných geografických souřadnic, používá se knihovna http k provedení HTTP GET požadavku na danou URL adresu, která je sestavena pomocí zadaných souřadnic a proměnné apiUrl. Funkce http.get vrací objekt Future, proto se používá await k asynchronnímu čekání na odpověď serveru.

Na druhou stranu návratová hodnota Stream je posloupnost asynchronních událostí. Je to jako asynchronní Iterable – kde místo získání další události, když o ni požádáte, vám stream řekne, že existuje událost, když je připraven.

6.7 Schéma komunikace rozhraní API

Firebase je sada nástrojů pro “ sestavení, vylepšení a růst aplikace. Nástroje, které poskytuje, pokrývají velkou část služeb, které by vývojáři normálně museli budovat, ale opravdu nechtějí stavět, protože by se raději soustředili na samotný zážitek z aplikace. To zahrnuje věci jako analytika, autentizace, databáze, konfigurace, ukládání souborů, zasílání zpráv a seznam pokračuje.



Obrázek 22 – Schéma API. Zdroj: Vlastní

Ve své implementaci využívám třídu dle návrhového vzoru Singleton, která obsahuje pouze jednu instanci třídy při běhu aplikace. Níže na obrázku lze vidět kontejnerizace koncového endpoint serveru, který se může na základě vlastních požadavků měnit. V ukázce chci

```
ForecastApiProvider._internal({required this.apiUrl});

Future<List<Forecast>> getDailyForecast(double lat, double lon) async {
  final response =
      await http.get(Uri.parse('$apiUrl/data/forecast/daily/$lat/$lon'));

  if (response.statusCode == 200) {
    final List<dynamic> jsonList = json.decode(response.body);
    return jsonList.map((json) => Forecast.fromJson(json)).toList();
  } else {
    throw Exception('Failed to load daily forecast');
  }
```

Obrázek 23 – příklad vytvoření klientského požadavku na získání listu počasí

především ukázat způsob komunikace mezi klientskou aplikací, ze které jde požadavek na náš server pro získání dat z meteostanice pro danou lokaci, a následnému zpracování odpovědi uložené v proměnné response. Podstatným prvkem je využití balíčku http, který jsem zvolil pro navázání komunikace se vzdáleným serverem. V porovnání s využitím platformy Firebase v kontextu komunikace v kódu nemusíme využívat napřímo protokoly http, jelikož za nás veškerou logiku řeší Firebase SDK balíček, který máme stažený vně aplikace jako knihovnu, a jejíž implementaci a fungování budu věnovat kapitolu. Dle demonstrace níže lze vidět metodu pro přihlášení právě pomocí rozšíření pro autorizaci, ze které lze vidět zjednodušení v podobě vytvoření instance a následnému vložení dat.

6.8 Požadavky na mobilní aplikaci zaměřenou na meteorologii

Sběr všech požadavků je počáteční krok ve vývoji. Pro získání požadavků širší veřejnosti jsem čerpal ze studie věnující se očekávání uživatelů od meteorologické mobilní aplikace.

6.8.1 Meteorologické mobilní aplikace

Vývoj meteorologických mobilních aplikací se zaměřuje na vytváření uživatelsky přívětivých rozhraní umožňující uživatelům snadno a rychle získávat potřebné informace. Tyto aplikace využívají data poskytovaná meteorologickými službami, jako je například NASA, NOAA (Národní úřad pro oceán a atmosféru) nebo dalšími globálními i lokálními

poskytovateli meteorologických dat. Nativní meteorologické aplikace obsahují specializované funkce a nástroje, jako jsou interaktivní mapy s teplotními a srážkovými vrstvami, grafy větrů, upozornění na extrémní počasí, historická data a další. Co však chybí je možnost vlastního testování, customizování vlastních funkcí nebo ohnutí aplikace dle vlastních potřeb.

Vyhledal jsem průzkum věnující se důvodům používání mobilních aplikací uživateli, abych mohl snáze identifikovat potřeby uživatelů pro svůj projekt. [15]

Studie zkoumala motivace uživatelů mobilních aplikací pro počasí (MWA) pomocí faktorové analýzy a interpretace získaných faktorů. První faktor identifikovaný jako "Spravedliví nebeští" [15] zahrnoval jednotlivce s vysokým zájmem o denní změny počasí a astronomii, kteří chtěli být informováni o konstelacích, východu a západu slunce a dalších eventech. Druhá skupina nazvaná jako "Rozumní dešťoví nadšenci" zahrnovala praktičtější jedince, kteří si aplikace pro počasí cenili pro svou jednoduchost a použitelnost. Aplikaci využívali pro sledování počasí pro nadcházející plány a opatrování svých blízkých informováním o předpovědích.

Výstupem z této studie je pro mě důraz na jednoduchost a intuitivní ovládání, předpověď počasí je v meteorologické aplikace standardem.

S těmito požadavky budu dále ve vývoji své aplikaci pracovat a zařadím je do akceptačních kritérií, které jsou esenciální pro naplnění cíle, což je implementace edukativní mobilní aplikace pro žáky středních a základních škol.

Požadavek	Popis požadavku
Přehled aktuálního počasí	Zobrazování aktuálních meteorologických podmínek v reálném čase, včetně teploty, vlhkosti, srážek, rychlosti větru a dalších relevantních informací.
Předpověď počasí na následující dny	Poskytování spolehlivé předpovědi počasí na několik dalších dnů dopředu s detaily

	jako teplota, srážky, oblačnost, rychlost a směr větru a další.
Interaktivní mapy a vizualizace	Možnost prozkoumávat interaktivní mapy s meteorologickými daty, včetně radarových snímků srážek, teplotních map, map větrů, či vizualizací tlaku vzduchu.
Upozornění na extrémní podmínky	Oznámení uživatelům o případných extrémních meteorologických podmínkách, jako jsou bouřky, silné větry, přívalové deště, či extrémní teploty.
Možnost nastavení lokalizace	Umožnění uživatelům nastavit svou geografickou polohu pro přesnější zobrazení aktuálního počasí a předpovědi.
Snadné použití a přehlednost	Uživatelské rozhraní by mělo být intuitivní a snadno ovladatelné pro studenty středních škol bez předchozích znalostí meteorologie.

PRAKTICKÁ ČÁST

7 GRAFICKÝ NÁVRH APLIKACE

Tvorba grafického návrhu má své opodstatnění a hraje klíčovou roli ve vývoji mobilních aplikací. V teoretické části jsem uvedl metodiky vývoje, které obsahují procesy pro části aktérů. Vzhledová část aplikace, dále UI, je tvořena grafickým návrhářem, který manuál pro vývojáře, knihovnu komponent, typografie, barevné palety využité v aplikaci. Důležitým aspektem při tvorbě aplikace je brát zřetel na cílové platformy, na kterých aplikace bude, konzistentnost komponent a dalších parametrů UI. V mobilním vývoji existují oficiální knihovny, které obsahují seznam nativních komponent pro platformu.

- iOS – Cupertino library
- Android – Material Component library

```dart
import 'package:flutter/material.dart';
import 'package:flutter/cupertino.dart';
```

Obrázek 24 – Vložení nativních balíčků do aplikace

Obě knihovny obsahují komponenty vlastní svým standardům a UI/UX návrhům, proto si můžeme všimnout různorodosti při porovnání Android knihovny s iOS, kdy Apple klade důraz na minimalismus a jednoduchost. Obě z těchto knihoven můžeme využít při tvorbě grafického návrhu, já pracoval pouze s Android verzí s ohledem na komplexitu projektu. Nativní knihovny dle obrázku výše sami dle logiky importují vlastní nastavení vzhledu, kdy Flutter na nativní úrovni získá informaci o platformě a na základě toho zvolí správný balíček.

7.1 Nástroje pro tvorbu grafického návrhu aplikace

Figma je grafický editor pro společný návrh webových stránek, aplikací a dalších designových produktů. Objevil se v roce 2016 jako obdoba Sketche a Adobe XD, ale o pár let později se stal jedním z nejoblíbenějších nástrojů pro designéry. Důležitým požadavkem, který kladu na grafický nástroj škálovatelnost, jelikož nerad vyvíjím úsilí na psaní už jednou napsaného kódu, a stejný přístup aplikuje i v grafickém návrhu. Dalším pilířem kvalitního nástroje pro tvorbu je komunitní základna, která vytváří a rozvijí grafické knihovny.

Tento nástroj představuje flexibilní nástroj umožňující konzistenci napříč projekty. Tento samostatný prostor umožňuje návrhářům vytvářet obrazovky aplikací a díky standardním velikostem pro různá zařízení usnadňuje práci designerům. Jsou zde zabudovaná plátna pro specifické mobilní zařízení, webové rozlišení.

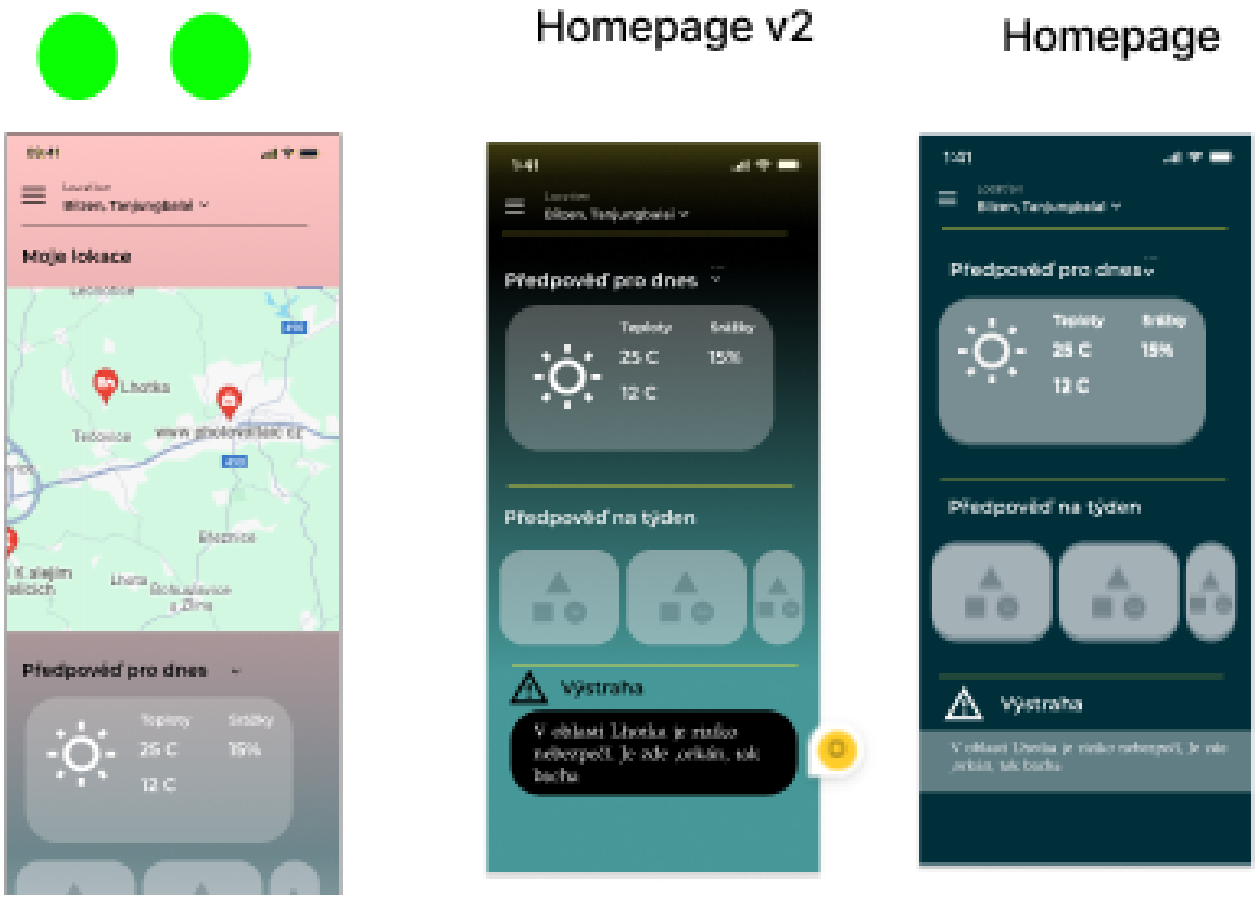
Komponenty umožňují standardizovat designové prvky a usnadňují úpravy v návrhu. Díky nim můžete rychle změnit obecný styl rozvržení a změnit tak komponentu na více místech zároveň, což je při využití globální knihovny veliký benefit. Režim úprav pro více uživatelů odstraňuje problémy spojené s kolizemi designérských a vývojářských prací. Všechny soubory jsou uloženy v cloudu, což usnadňuje týmovou spolupráci a umožňuje mnoha uživatelům pracovat na jednom projektu současně.

7.1.1 Figma

Figma nabízí bezplatnou verzi s omezeními, která je vhodná pro individuální práci. Pro týmovou spolupráci můžete přejít na verzi Professional za měsíční poplatek, která odstraňuje omezení a poskytuje další výhody. Využívání Figmy online umožňuje rychlou práci na projektech a efektivní komunikaci v průběhu procesu designu. [36]

Figma je webová aplikace pro spolupráci na návrzích uživatelských rozhraní, kterou lze také používat offline pomocí desktopových aplikací pro Windows a macOS. Figma je bohatou platformou zaměřenou na návrh uživatelských rozhraní a přívětivosti pro uživatele, s důrazem na spolupráci v reálném čase prostřednictvím několika editorů vektorové grafiky a nástrojů pro prototypování. Na mobilních zařízeních a tabletech umožňuje mobilní aplikace Figma pro iOS a Android prohlížení a interakci s Figma prototypy v reálném čase. [36]

Díky tomuto nástroji jsem byl schopen vytvořit základní prototypy aplikace v průběhu doby vývoji od základní myšlenky až po téměř finální verzi vzhledu. Na obrázku níže lze vidět tři verze hlavní stránky aplikace, kdy zleva vidíme označení zelenými body, což pro mě sloužilo k označení kandidáta pro produkční verzi.



Obrázek 25 – Návrh domovské stránky pro aplikaci Meteoratlas

7.2 UI/UX Aplikace

V mém grafickém návrhu jsem pracoval s několika verzemi vzhledu jednotlivých stránek a schéma barev, a poté jsem na základě bodování dospěl k finální verzi.

Pod názvem této kapitoly si primárně představuji uživatelskou zkušenost, tento význam z očí programátora nese spoustu faktorů a aspektů, na které musí dbát. Jedním z nich je konzistence, který nemůžeme vidět pro stránku registrace na obrázku níže, kde lze vidět pistáciová

Obrázek 26 – Návrh vstupních stránek do aplikace

barva, na prototypu níže je vzhledem k využitým gradientům v pozadí sice správná barva pozadí, ale výška barevného přechody by neměla být rozdílná jako v předešlých obrazovkách.

7.2.1 Standardy při tvorbě aplikací

Standardy UI / UX jsou soubor pokynů, které definují, jak by uživatelské rozhraní mělo vypadat a fungovat. Jsou důležité pro zajištění toho, aby uživatelé měli při interakci s produktem konzistentní a předvídatelné zkušenosti. Existuje mnoho organizací, které vyvíjejí standardy a pokyny, které pomohou každému vytvořit web na základě zásad přístupnosti, internacionalizace, soukromí a zabezpečení, jako je Google, World Wide Web Consortium (W3C). [10]

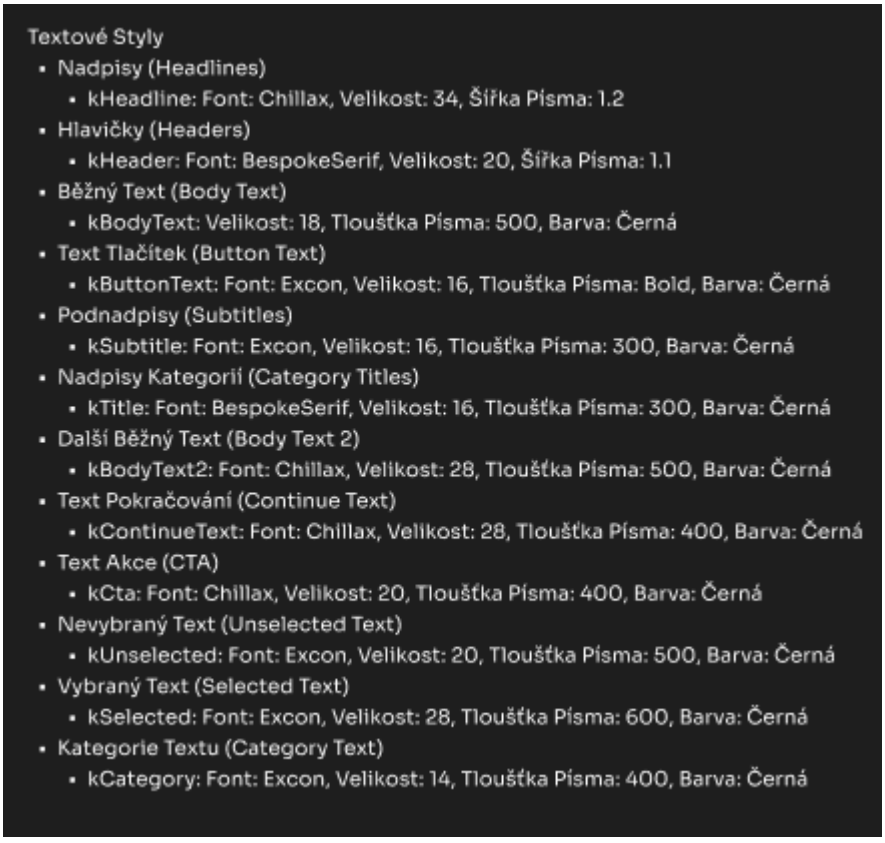
Konzistence je klíčová pro uživatelskou zkušenost, protože pomáhá uživatelům předvídat, co se stane, když provedou určitou akci. Konzistentní UI zahrnuje používání stejných vzorů, prvků a interakcí napříč celou aplikací. To zajišťuje, že uživatelé se nemusí znovu učit, jak používat různé části aplikace. Tím, že si vypíšeme kuchařku s jednotlivými barvami a styly jsme schopni v kódu implantovat globální nastavení vzhledu z jednoho místa, a zamezíme tak duplicitnímu využití. Pro aplikaci Meteoratlas jsem zvolil rozdělení do dvou barevných palet, jak lze vidět na obrázku níže.

Obrázek 27 – Rozvržení barevného schéma do aplikace

Samotnou implementaci v kódu realizuji v kapitole věnované využitých technologií.

Každý uživatel má vlastní preference, což je důvod, proč jsem přidal právě tuto funkcionalitu výběru mezi vícero nastavení vzhledu aplikace. Různé situace a prostředí vyžadují odlišný vizuální styl. Například v noci může být preferováno tmavé téma, zatímco během dne může být vhodnější světlé téma, taktéž stojí za úvahu v kontextu meteorologické aplikace zobrazit pozadí dle situace počasí.

Podobným principem jsem si definoval i typografii textu pro aplikaci, přičemž jsem využil tří fontů – Bespoke Serif, Excon a Chillax, a pro jednotlivé případy užití definoval velikost, tloušťku, barvu a v případech nadpisů i odsazení mezi písmeny.



Obrázek 29 – Rozvržení typografie v prostředí Figma

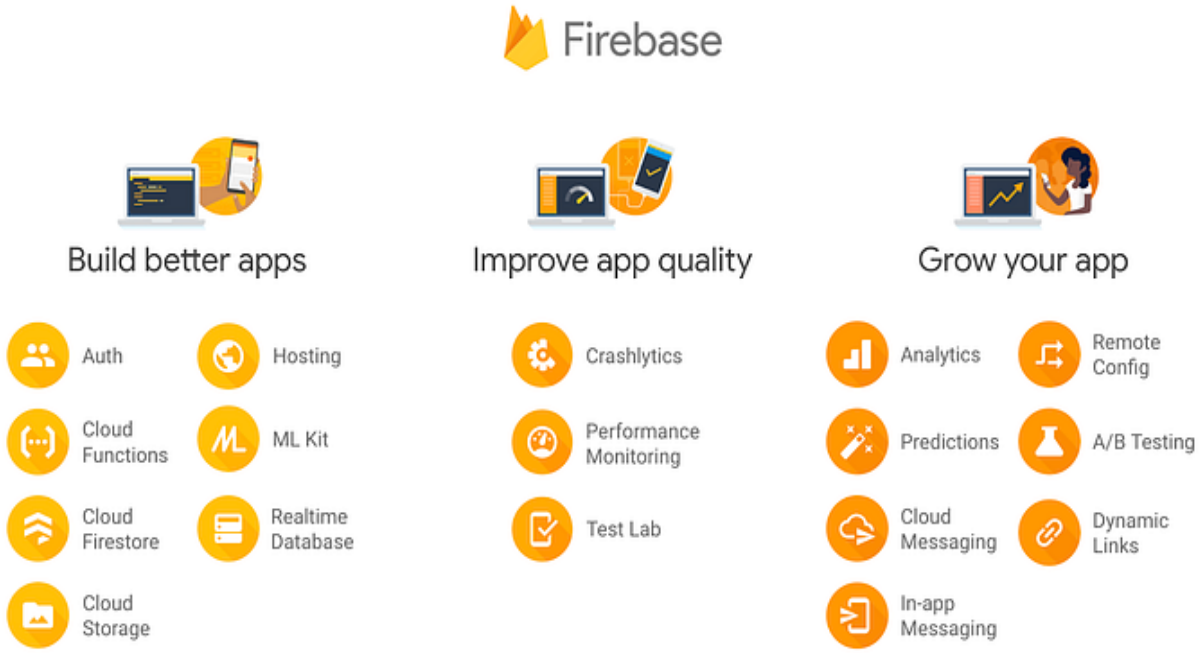
8 VYUŽITÉ TECHNOLOGIE A NÁSTROJE

Kromě samotného nástroje vývoj Flutter jsem použil dostupné knihovny a platformu pro správu a usnadnění některých funkcí. V těchto kapitolách se věnuji platformě Firebase, která nabízí spoustu hotových funkcí integrováním SDK do aplikace.

8.1 Platforma Firebase

Firebase je multiplatformní prostředí pro vývoj aplikací, ať už mobilních pro iOS nebo Android, tak pro web. Aplikace obsahuje několik nástrojů pro udržení a rozšíření standartních aplikací na novou úroveň. Platforma je zdarma, avšak najdou se zde prémiové funkce, které vyžadují placený účet. V dalších kapitolách se budu věnovat jednotlivým funkcím a nástrojům využitých v mé diplomové práci, včetně implementace a konfigurování.

Firebase v kontextu aplikace vzdálený server, díky kterému můžeme například odesílat push notifikace nebo autorizovat uživatele. [34]



Obrázek 30 – Zobrazení poskytovaných funkcí do aplikace. Zdroj [34]

Pro nasazení platformy do aplikace je potřeba splnit pár kroků, které uvedu níže.

Prvním krokem je vytvoření projektu ve Firebase Console na webové stránce Firebase Console. Zde můžete zadat název projektu a vybrat zemi nebo oblast, ve které chceme projekt umístit. Konfigurace dle systematičnosti probíhá takto:

1. Vytvoření a přidání aplikace do Firebase platformy. Již dříve na začátku kapitoly jsem doporučil následování příručky vývojářů z oficiálních stránek, které jsou barvitě a dostatečně popsány.

2. Přidání konfiguračních souborů do aplikace pro iOS a Android, konfigurace obsahuje informace o API vygenerovaném klíči pro náš účet, identifikátory pro databázi nebo pro Google Analytics.

3. Přidat tyto soubory do verzovacího souboru *gitignore*, aby nebyl účet publikováním API klíčů zneužit.

```
static const FirebaseOptions web = FirebaseOptions(
  apiKey: 'api-akey',
  appId: '1:akey:web:akey',
  messagingSenderId: 'akey',
  projectId: 'meteorat',
  authDomain: 'meteorat.firebaseapp.com',
  databaseURL:
      'https://meteorat-default-rtdb.europe-west1.firebasedatabase.app',
  storageBucket: 'meteorat.appspot.com',
  measurementId: 'G-CH1TZ0MEKH',
);
```

Obrázek 31 – Soukromé konfigurační parametry pro komunikaci s Firebase

4. Inicializace funkcí do aplikace, viditelných na obrázku níže. Jsou zde nativní metody při inicializování Firebase na klientské úrovni, přidání knihovny pro získávání chyb uživatelů nesoucí název Crashlytics, nebo vytvoření instance pro odesílání notifikací.

```
await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  FlutterError.onError = (errorDetails) {
    FirebaseCrashlytics.instance.recordFlutterFatalError(errorDetails);
  };
  PlatformDispatcher.instance.onError = (error, stack) {
    FirebaseCrashlytics.instance.recordError(error, stack, fatal: true);
    return true;
  };
  await FirebaseMessaging.instance.setAutoInitEnabled(true);
  await notificationRepository.setupBackgroundMessage();
  await notificationRepository.setupForegroundMessage();

  FirebaseMessaging.onBackgroundMessage(_firebaseMessagingBackgroundHandler);
```

Obrázek 32 – Inicializace služby Firebase

Výhodou integrace této služby je spousta benefitů v podobě funkcionality, ale především to, že každý uživatel již má staženou knihovnu v klientské aplikace a komunikace tak probíhá na nativní úrovni. Díky tomuto architektonickému návrhu není vhodné sdílet citlivé údaje, jako jsou API klíče v konfiguračních souborech. V dalších kapitolách bych se věnoval konkrétním funkcím, které platforma nabízí.

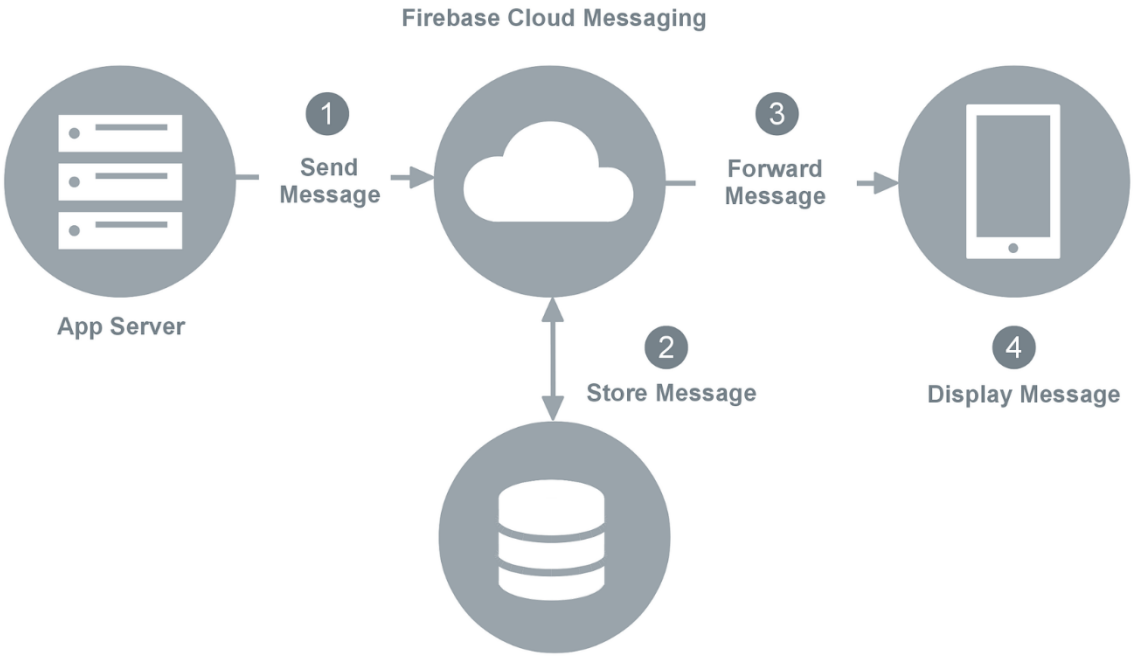
8.2 Firebase Cloud Messaging

FCM umožňuje vývojářům softwaru odesílat push notifikace pro jejich aplikace koncovým uživatelům prostřednictvím rozhraní pro programování aplikací (API). Pomocí FCM můžeme klientské aplikaci oznámit nový e-mail nebo jiná data je k dispozici pro synchronizaci. Pro případy použití, jako je rychlé zasílání zpráv, může zpráva přenést užitečné zatížení až 4000 bytů do klientské aplikace. [34]

Implementace FCM zahrnuje dvě hlavní komponenty pro odesílání a přijímání:

- Backend prostředí pro přenesení logiky z klientské aplikace do pozadí, jako jsou cloudové funkce pro Firebase nebo aplikační server na kterém lze stavět, cílit a odesílat zprávy.
- Klientská aplikace Apple, Android nebo web (JavaScript), která přijímá zprávy prostřednictvím odpovídající přepravní služby specifické pro platformu.

Server tedy distribuuje zprávy do Firebase platformy, která následně data přeposílá do klientských aplikací, ve kterých na základě lokace uživatele zobrazí notifikace o blížící se výstraze v lokaci. Princip komunikace jednotlivých koncových zařízení a prostředníka Firebase lze vidět na obrázku níže.



Obrázek 33 – Architektura komunikace API [34]

Pomocí cílení zpráv je FCM schopen doručovat zprávy aplikacím třemi způsoby, a to do jednotlivých zařízení, do skupin zařízení nebo do zařízení předplatných témat. Vývojáři mají možnost vytvářet zprávy ve skladateli oznámení, který může odesílat cílené zprávy konkrétním segmentům uživatelů. Tyto zprávy jsou plně integrovány do Firebase Analytics, která sleduje zapojení a převod uživatelů.

Kód níže je implementace notifikací pro obě mobilní platformy. Singleton návrhový vzor je implementován pomocí statické proměnné _instance a privátního konstruktoru _internal(), který zabraňuje přímému vytvoření nové instance třídy. Instance je přístupná pomocí tovární metody factory NotificationRepository(), která vrací existující instanci nebo vytváří novou, pokud ještě neexistuje. Níže vysvětlím princip jednotlivých funkcí, jejich definice čerpám z oficiální dokumentace [34], implementovaných na obrázku níže.

- Metoda *setupForegroundMessage*()
    - Metoda nastavuje zachycení notifikací při běhu aplikace. Získává také FCM token a při zachycení notifikace vypisuje příslušná data a zobrazuje notifikaci v uživatelském rozhraní.
- Metoda *setupBackgroundMessage*()
    - Metoda nastavuje zachycení notifikací v pozadí.
- Metoda *_firebaseMessagingBackgroundHandler*()
    - Tato metoda zpracovává notifikace zachycené v pozadí.
- Metoda *handleNotificationPermission*()
    - Metoda zpracovává oprávnění pro zobrazování notifikací.
- Metoda *_showNotificationInUI*()
    - Metoda zobrazuje notifikaci v uživatelském rozhraní pomocí balíčku flutter_local_notifications.
- Statická metoda *showTextNotification*()
    - Statická metoda slouží k zobrazení textové notifikace s určitými parametry. Používá se pro zobrazování specifických notifikací v různých částech aplikace.
- @pragma('vm:entry-point')
    - Tato direktiva specifikuje bod vstupu pro JIT kompilaci a slouží k označení metody *_firebaseMessagingBackgroundHandler*() jako vstupního bodu pro pozadí. Tato metoda bude spuštěna, když bude aplikace v pozadí a zachytí se nová notifikace.

```
Future<void> setupBackgroundMessage() async {
  FirebaseMessaging.onBackgroundMessage(_firebaseMessagingBackgroundHandler);
}

@pragma('vm:entry-point')
Future<void> _firebaseMessagingBackgroundHandler(
    RemoteMessage message) async {
  print("Handling a background message: ${message.messageId}");
  _showNotificationInUI(
    title: message.notification!.title ?? '',
    body: message.notification!.body ?? '',
  );
}

Future<void> handleNotificationPermission() async {
  NotificationSettings settings = await _messaging.requestPermission(
    alert: true,
    announcement: false,
    badge: true,
    carPlay: false,
    criticalAlert: false,
    provisional: false,
    sound: true,
  );

  print('User granted permission: ${settings.authorizationStatus}');
}

Future<void> _showNotificationInUI(
    {required String title, required String body}) async {
  const AndroidNotificationDetails androidPlatformChannelSpecifics =
      AndroidNotificationDetails('your_channel_id', 'your_channel_name',
          icon: '@mipmap/ic_launcher',
          importance: Importance.max,
          priority: Priority.high);

  const NotificationDetails platformChannelSpecifics =
      NotificationDetails(android: androidPlatformChannelSpecifics);

  await flutterLocalNotificationsPlugin.show(
    0,
    title,
    body,
    platformChannelSpecifics,
    payload: 'item x',
  );
}
```

Obrázek 34 – Implementace příjmu notifikace na straně klienta

8.3 Firebase Authorization Service

Rád bych se více věnoval této třídě, která plní funkci přihlašování díky integraci Firebase balíčku do projektu. Již dříve jsem se v kapitole 8.1 věnoval samotnému balíčku, jeho implementaci a jeho funkcím. Přihlašování je důležitým aspektem aplikace, které vyžadují autorizaci uživatele a definování mantinelů, co uživatel může a nemůže dělat. Dle kódu níže je třída koncipována dle vzoru singleton, neboli existuje pouze jedna instance během životního cyklu aplikace. Zvolil jsem tuto možnost, jelikož mi přišla architektonicky smysluplnější – při špatném použití zde hrozí problém s pamětí, nicméně v případě využívání autorizace nechceme vytvářet nové instance, třebaže ani omylem. Tato hodnota je stejná při jakémkoliv zavolání konstruktoru, jelikož se použije interní hodnota *internal*.

Samotné metody už pouze pracují s instancí Firebase Auth Service, která umožňuje komunikovat se serverem pro ukládání nových dat nebo porovnání dat pro přihlášení. Jak si lze všimnout z implementace kódu, metody jsou asynchronní, což je ve webovém prostředí rovno premise – příslibu, že data budou navráceny, jakmile bude získána informace od cíleného serveru. Stejným principem funguje komunikace i s Firebase, díky konfiguraci, kterou jsem se zabýval v kapitole 8.1. a spravuje je stejný balíček. [34]

Zde je přehled klíčových prvků autorizace, které využívám u sebe v aplikaci, viditelných na obrázku níže.

- Enum AuthStatus
  Výčet definuje různé stavy ověření, které mohou nastat při pokusu o přihlášení.
- Třída AuthExceptionHandler
  Třída obsahuje metody pro zpracování výjimek při ověřování.
- Metoda handleAuthException()
  Metoda přijímá výjimku typu FirebaseAuthException a na základě kódu výjimky určuje odpovídající stav ověření. Výsledek je vrácen jako hodnota výčtu AuthStatus.

```
enum AuthStatus {
  successful,
  wrongPassword,
  emailAlreadyExists,
  invalidEmail,
  weakPassword,
  unknown,
}

class AuthExceptionHandler {
  static handleAuthException(FirebaseAuthException e) {
    AuthStatus status;
    switch (e.code) {
      case "invalid-email":
        status = AuthStatus.invalidEmail;
        break;
      case "wrong-password":
        status = AuthStatus.wrongPassword;
        break;
      case "weak-password":
        status = AuthStatus.weakPassword;
        break;
      case "email-already-in-use":
        status = AuthStatus.emailAlreadyExists;
        break;
      default:
        status = AuthStatus.unknown;
    }
    return status;
  }
```

Obrázek 35 – Enum obsahující stavy hlášek autorizace

Vzhledem k využitému stavu řízení, jímž je BLoC, jehož cílem je odseparovat logiku, kontrolér a vzhled. Je klíčové uvést důležité části plnící funkční celek – autorizace. Data jsou abstraktně odděleny ve třídě AuthRepository, která komunikuje s rozhraním Firebase. Následující kód ukazuje implementaci, která zprostředkovává autentizaci uživatelů pomocí Firebase Auth a ukládání dat o uživatelích do Firestore. Rozšířením je enum AuthStatus a rozšíření AuthErrorExtention pro výpis návratové error hodnoty. Kombinací získáme z datové vrstvy konkrétní výsledek funkce.

Obrázek 36 – Implementace třídy pro autorizaci uživatele

```dart
import 'package:firebase_auth/firebase_auth.dart';

class AuthService {
  final FirebaseAuth _auth = FirebaseAuth.instance;

  // Singleton instance
  static final AuthService _instance = AuthService._internal();

  factory AuthService() {
    return _instance;
  }

  AuthService._internal();

  // Registrace uživatele
  Future<User?> registerWithEmailAndPassword(
      String email, String password) async {
    try {
      UserCredential result = await _auth.createUserWithEmailAndPassword(
        email: email,
        password: password,
      );
      User? user = result.user;
      return user;
    } catch (e) {
      print(e.toString());
      return null;
    }
  }

  // Přihlášení uživatele
  Future<User?> signInWithEmailAndPassword(
      String email, String password) async {
    try {
      UserCredential result = await _auth.signInWithEmailAndPassword(
        email: email,
        password: password,
      );
      User? user = result.user;
      return user;
    } catch (e) {
      print(e.toString());
      return null;
    }
  }

  // Odhlášení uživatele
  Future<void> signOut() async {
    await _auth.signOut();
  }

  // Získání aktuálně přihlášeného uživatele
  User? getCurrentUser() {
    return _auth.currentUser;
  }
}
```

Tato třída je odvozena z třídy Bloc <AuthEvent, AuthState>, což je obecný vzor pro zpracování událostí (AuthEvent) a produkci stavů (AuthState). Na obrázku níže zobrazuji konkrétní vytvoření kontroléru Bloc pro autorizaci uživatele, který dle návrhové vzoru reaguje na eventy a vrací nový stav aplikace, na který ve *view* reagujeme.

```
class AuthBloc extends Bloc<AuthEvent, AuthState> {
  final AuthRepository _authRepository = AuthRepository();

  AuthBloc() : super(AuthState()) {
    on<SignInEvent>(_mapSignInToState);
    on<SignUpEvent>(_mapSignUpToState);
    on<LogOut>(_mapLogOutState);
  }

  Future<void> _mapSignInToState(
      SignInEvent event, Emitter<AuthState> emit) async {
    emit(AuthLoadingState());

    User? user = await _authRepository.signInWithEmailAndPassword(
      event.email,
      event.password,
    );

    if (user != null) {
      emit(AuthAuthenticatedState(user: user));
    } else {
      emit(AuthErrorState(error: "Failed to sign in"));
    }
  }
```

Obrázek 37 – Řízení stavů autorizování uživatele

Každá událost (AuthEvent) je mapována na odpovídající stav (AuthState) pomocí metody on. Vracíme AuthAuthenticatedState v případě úspěšného přihlášení nebo AuthErrorState v případě chyby. Při odhlášení se mapuje událost LogOut na stav AuthState s nulovým stavem, totiž abych mohl následně reagovat na změny ve view aplikace.

8.4 Třída pro nastavení vzhledu aplikace

Theme je v kontextu aplikací známým pojmem, lze si jej představit od vzhledu tlačítka až po komplexní řešení barevné palety. Barevná paleta je koncipovaná dle nativního rozdělení tmavého a světlého režimu, a sentimentálně zbarvené do meteorologického tématu. Výsledkem je vytvoření daylightTheme, která obsahuje předdefinované styly s barvami, kterými jsem se zabýval v kapitole vytváření grafického návrhu.

```
static final daylightTheme = buildBaseTheme(
  primaryColor: AppColors.lightAqua,
  scaffoldBackgroundColor: Colors.white,
  backgroundColor: AppColors.green,
  appBarColor: AppColors.green,
  iconColor: Colors.black,
  titleTextStyle: const TextStyle(
    color: Colors.black87,
    fontSize: 16.0,
    fontWeight: FontWeight.bold,
  ),
  textTheme: lightTextTheme,
  brightness: Brightness.light,
  colorScheme: const ColorScheme.light(
      primary: AppColors.lightAqua,
      secondary: Colors.deepOrange,
      tertiary: AppColors.lightAqua,
      onTertiary: Colors.black),
);
```

Obrázek 38 – Nastavení schématu barev v aplikaci

8.5 Cache dat

Cache – neboli mezi paměť, sloužící k urychlení procesu práci s daty tím, že výsledky zkusíme uložit pro pozdější použití. Ve Flutter tuto funkcionalitu implementuje mixin CacheMixin, který má dvě metody pro práci s cache pamětí. Vstupními parametry jsou klíče, jelikož jsou hodnoty ukládané strukturovaně jako klíč a hodnota.

```dart
mixin CacheMixin<T> {
  Future<T?> loadFromCache(String key) async {
    final cachedData = await HydratedBloc.storage.read(key);
    if (cachedData != null) {
      return json.decode(cachedData) as T;
    }
    return null;
  }

  Future<void> saveToCache(String key, T data) async {
    final encodedData = json.encode(data);
    await HydratedBloc.storage.write(key, encodedData);
  }
}
```

Obrázek 39 – Implementace ukládání do mezi paměti zařízení

8.6 Lokální uložiště Shared Preferences

Tento mechanismus umožňuje ukládat data ve formě klíč-hodnota a poskytuje jednoduchý způsob, jak ukládat a načítat nastavení aplikace, uživatelské preference nebo jiné drobné údaje. V aplikaci se nám tato funkce hodí při ukládání informací o nastavení jazyka nebo vzhledu aplikace, protože jsou to jednoduché datové struktury, jak lze vidět na obrázku níže. Stejným principem data získáváme nebo mažeme, chronologicky nejdříve získáme klíč a pokusíme se dotázat, zda klíč existuje v uložišti, a dále zpracovávat výstupy. Pro vývojáře je dostupný z komunitního katalogu rozšíření. [35]

Pokud bychom chtěli ukládat komplexnější datové struktury, nedoporučuje se použití tohoto balíčku z důvodu zápisu mapy, kdy existuje ke klíči právě jedna hodnota. Modely by bylo třeba objektově přenést do formátu json. Komplikace nastává při kolizi přepisování hodnot nebo při ukládání důvěrných dat. Toto uložiště vytváří soubor obsahující tyto data, které nejsou bez implicitních úprav zabezpečené – jsou viditelné jako otevřený text.

```dart
class LocalStorage {
  static const String _keyPrefix =
      'meteoratlas_'; // Prefix pro klíče, abychom se vyhnuli kolizím

  // Metoda pro uložení hodnoty do lokálního úložiště
  static Future<void> saveValue(String key, dynamic value) async {
    final preferences = await SharedPreferences.getInstance();
    key = _getFullKey(key);
    if (value is int) {
      preferences.setInt(key, value);
    } else if (value is double) {
      preferences.setDouble(key, value);
    } else if (value is bool) {
      preferences.setBool(key, value);
    } else if (value is String) {
      preferences.setString(key, value);
    } else if (value is List<String>) {
      preferences.setStringList(key, value);
    } else {
      throw Exception('Unsupported value type');
    }
  }
```

Obrázek 40 – Implementace lokálního uložiště

9 OBSAH A KLÍČOVÉ ČÁSTI APLIKACE

Aplikace obsahuje sedm základních stránek, jejichž obsah a význam vysvětlím v následujících kapitolách. Jednotlivé stránkwy řídí nástroj pro řízení stavu v aplikaci, kterému jsem se věnoval již v teoretické části. Benefitem abstrakce business logiky od UI vrstvy a modelu je především lepší čitelnost kódu a údržba, jelikož je vše rozděleno do vlastních tříd dle BLoC patternu.

9.1 Splash screen

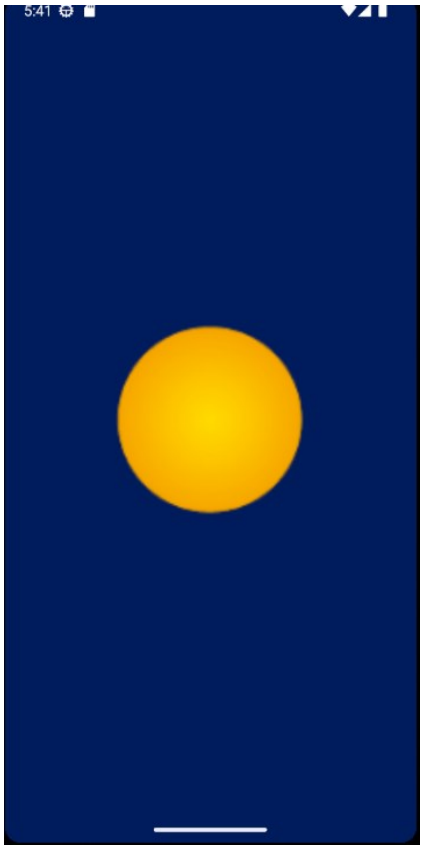
Každá aplikace se po otevření, pokud není na pozadí otevřená, načítá. Tento načítací čas aplikace využívá k načítání základních knihoven a komponent, které jsou nezbytné pro běh aplikace. To může zahrnovat načítání frameworků, externích knihoven, a dalších závislostí.

Během tohoto načítání se děje část procesů na pozadí nutných pro spuštění aplikace, stažení závislostí a balíčků pro inicializaci aplikace:

- Flutter framework se inicializuje. To zahrnuje načítání základních knihoven, inicializaci runtime prostředí a nastavení prostředí pro běh aplikace.
- Načítání konfigurací a zdrojů, jako jsou textové řetězce, obrázky, písma a další prostředky, které jsou nezbytné pro běh aplikace.
- Nastavení lokálního uložiště, nastavení jazyka a regionu, nastavení oprávnění a další konfigurace na úrovni uživatele.
- Aplikace inicializuje služby a pluginy, které mohou být použity v aplikaci, jako je správa stavu, navigace, HTTP komunikace, lokalizace a další.
- Před zobrazením Splash screen se může provádět kontrola stavu aplikace, jako je ověření uživatelských oprávnění, kontrola dostupnosti internetového připojení a další, a vývojář na ně může vytvořit patřičnou akci.
- Nakonec se připravuje samotný Splash obrazovka, která se zobrazí uživateli při spuštění aplikace.

```
flutter_native_splash:
  color: "#071b5d"
  image: assets/splash.png
  fill: true
  android: true
  ios: true
  color_dark: "#000000"
  android_12:
    image: assets/sun12.png
```

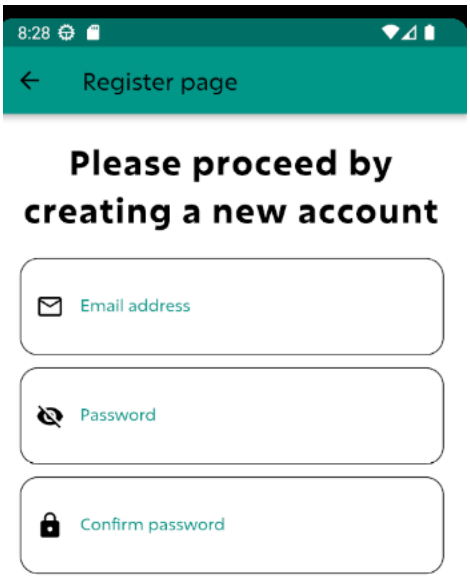
Obrázek 14 – Konfigurace nativního zobrazení



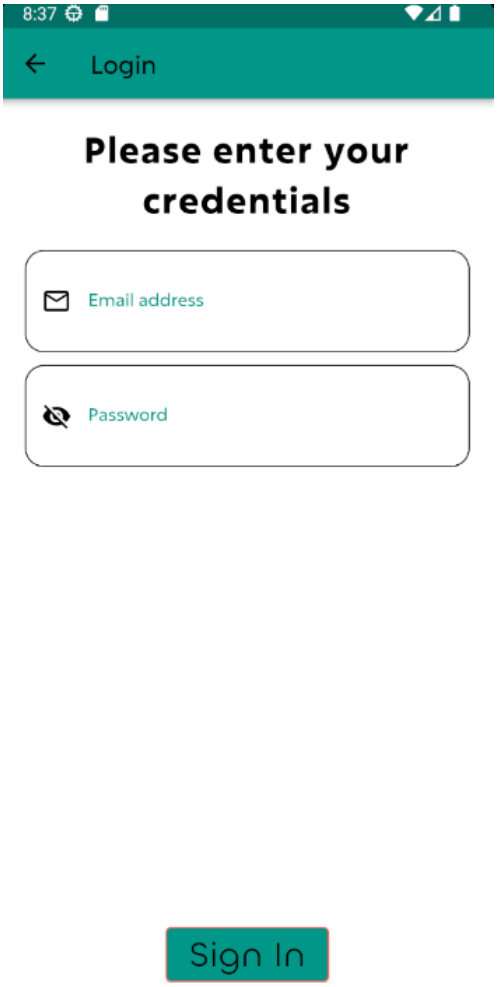
Obrázek 41 – Splash screen

9.2 Přihlášení a registrace

Při vytváření stránky pro registraci a přihlášení jsem vzal v potaz znovu-použitelnost komponent, kdy si jednoduše pomocí boolean hodnoty komponentu změním za dvou místech, a to v nadpisu a potvrzení hesla. Funkcionalitě se věnuji v kapitole věnované platformě Firebase, uvedu zde však ukázku integrace do validačního formuláře, konkrétně metodu authenticate, viditelnou na obrázku č. 44.



Obrázek 43 – Stránka registrace



Obrázek 42 – Stránka přihlášení

```
  void _authenticate(BuildContext context) {
    if (_formKey.currentState!.validate()) {
      final email = _emailController.text;
      final password = _passwordController.text;

      if (widget.isRegister) {
        context.read<AuthBloc>().add(SignUpEvent(
              email: email,
              password: password,
              checkPassword:
                  _confirmPasswordController.text == _passwordController.text,
            ));
      } else {
        // Přihlášení
        context.read<AuthBloc>().add(SignInEvent(
              email: email,
              password: password,
            ));
      }
    }
  }
}
```

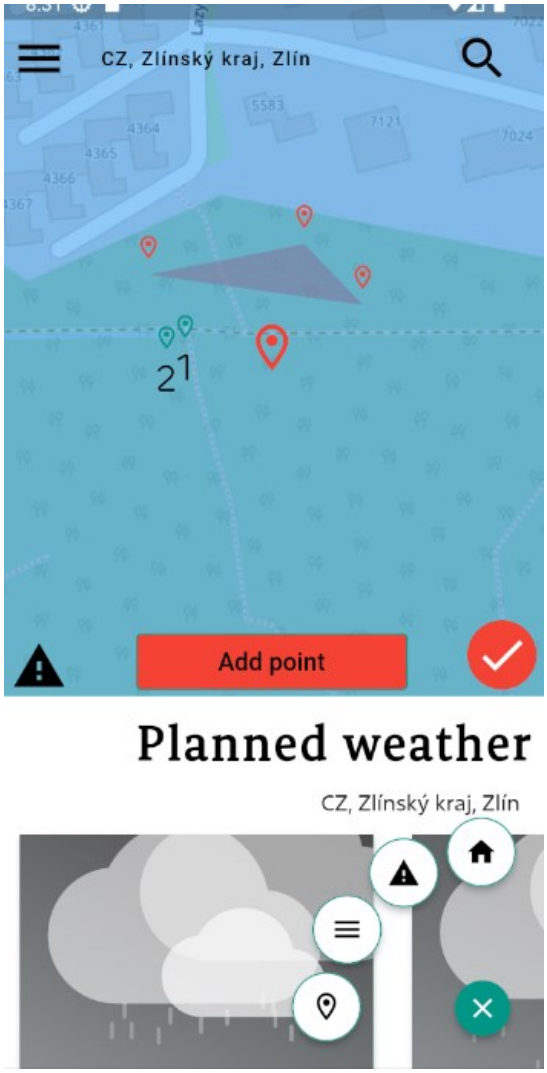
Obrázek 44 - Metoda pro odeslání údajů do kontroléru

9.3 Domovská stránka

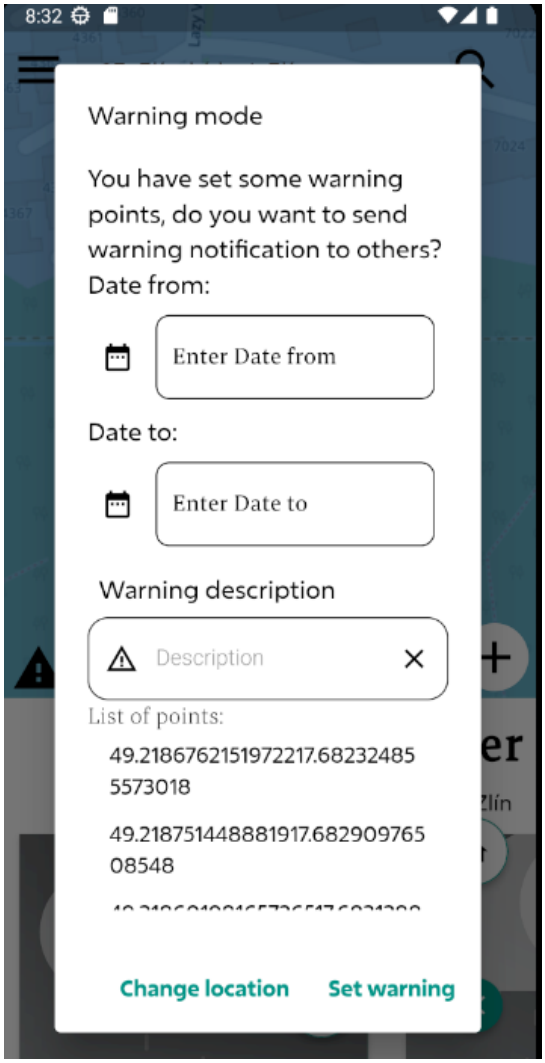
Domovská stránka je pomyslný střed aplikace, jelikož nabízí obsah a navigace napříč celou aplikací. Aplikace je lokalizována do dvou jazyků, při získání slovníků dalších cizích jazyků je možné lokalizace rozšířit. Domovská stránka zobrazuje sekce mapy, předpovědi počasí a výstrah, dva navigační prvky rozdělené diagonálně pro lepší uživatelský zážitek – vpravo dole a vlevo nahoře, viz. obrázek č.7 níže.

Díky využití řízení stavů pomocí eventů, stránka reaguje na změny a vykresluje patřičné komponenty. V praxi se například při stavu zakázaného sdílení lokace nezobrazí mapa, jelikož nebude automaticky získána, ale uživatel bude vyzván, ale zadal alespoň nějakou lokaci pro získání informací o počasí. Při tvorbě aplikace je klíčové znát uživatele, pro které je tvořena a vyvíjena, což jsou v mém případě studenti. Aplikace by měla umožnit zadávat různé lokace a nebýt fixován pouze na svou, ačkoliv je na tom založena funkcionalita push notifikací – odeslání zprávy v reálném čase o nově zadané výstraze v blízké oblasti uživatele.

Tlačítkem dole můžeme téměř ihned zjistit počasí v dané oblasti s daty, které se mění na základě změny aktuální lokace. Po potvrzení místa se vytvoří červená značka, která nám vizuálně zobrazuje předešlý zvolený bod. Důvodem implementace je například získání detailu počasí z výstrahy zobrazené na mapě, nebo samoúčelné brouzdání po mapě a získávání nových poznatků o měření. Ikona napravo s vykřičníkem vyznačuje právě zmíněné výstrahy na mapě v podobě barevných polygonů, jak jde vidět na obrázku níže.



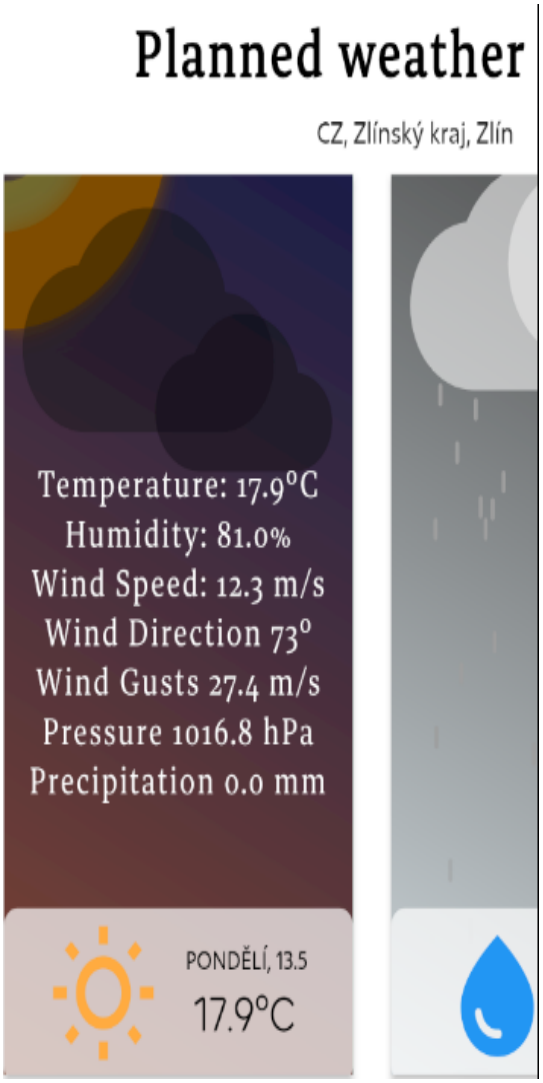
Obrázek 45 - Homepage



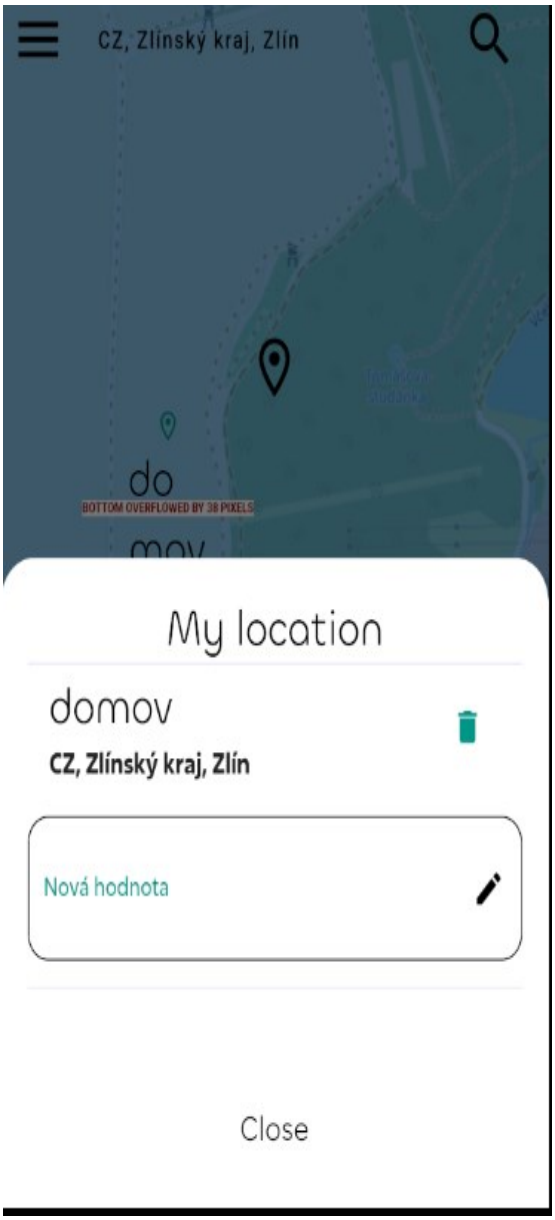
Obrázek 46 – Zadání výstrahy

Samotný list výstrah viditelný na obrázku č.46 vlevo dole obsahuje body s identifikovanými hrozbami počasí, například vysoká teplota, silný vítr nebo významné změny počasí. Po stisknutí tlačítka se mapa animuje na místo výstrahy.

Další sekcí na hlavní stránce je sekce předpovědi počasí pro každou hodinu a na týden dopředu. Důležitým prvkem jsou hodnoty karty po rozkliknutí, kde lze najít detailnější údaje, viditelné na obrázku níže.



Obrázek 47 – Detailní informace karty počasí

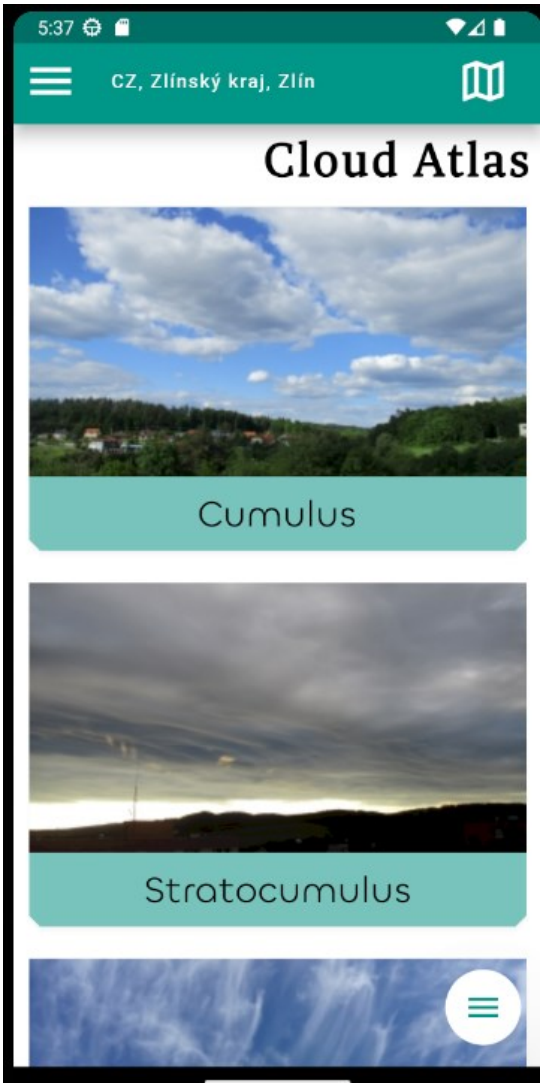


Obrázek 48 – Ukládání lokací s možností editace

9.4 Atlas mraků

Tato stránka je poskytována daty z vlastní databáze. Data obsahují informaci o adrese obrázku a dalších informacích, jako je popis mraku nebo latinský název.

Komponenta Atlas Page zobrazuje celý list mraků. Stránka se pak skládá z hlavičky stránky, která zobrazuje název atlasu, získaný z dat ze serveru na základě nastaveného jazyka uživatele. Používá se zde Bloc pro správu stavu aplikace, kterému jsem věnoval kapitolu teoretické části. Pokud jsou data načtena (AtlasLoadedState), zobrazí se seznam mraků, jinak se zobrazuje indikátor načítání (CircularProgressIndicator). Umožňuje rolovatelný obsah stránky.

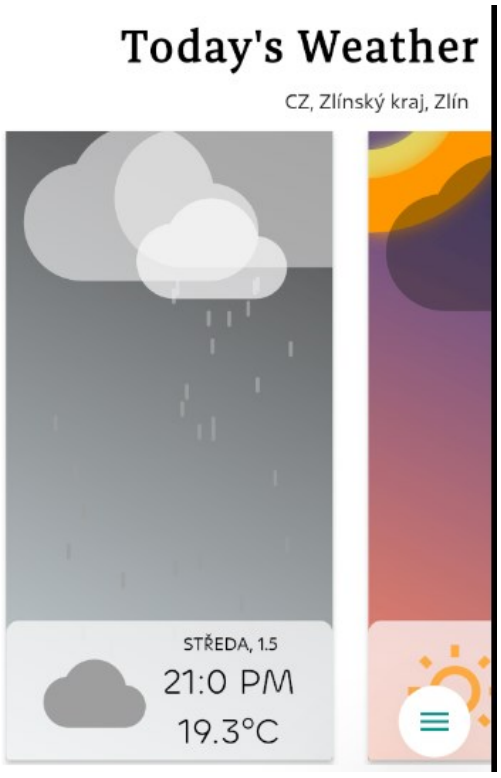


Obrázek 50 – přehled kategorií mraků

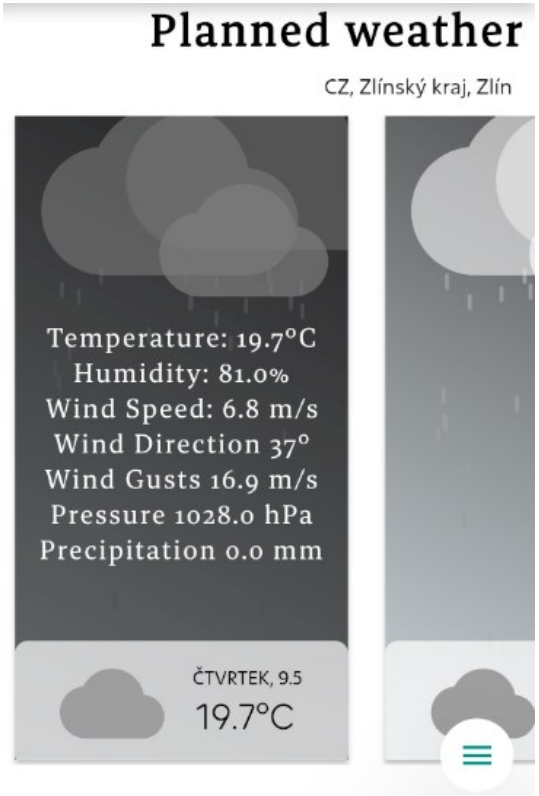
Obrázek 49 – Detail mraku

9.5 Předpověď počasí

Je určena k zobrazování informací o počasí, které zahrnují předpověď, aktuální podmínky, teploty a další relevantní informace. Karta počasí je stavová komponenta, což znamená, že její vzhled se může měnit na základě uživatelských interakcí nebo změn ve stavu aplikace. Komponenta má několik důležitých vlastností, pokud ji uživatel stiskne, zobrazí více informací, které lze vidět na obrázku níže. Komponenta zobrazuje různé textové informace, jako je teplota, vlhkost, rychlost větru a další relevantní údaje. Používá různé styly písma a velikosti pro zdůraznění důležitých informací. Tyto sekce obsahujíc tři stavy, budto data existují a načtou se, načítají se – v tom případě se zobrazí kolečko načítání, a stav nenačtených dat, kdy sekce nejde vidět vůbec, protože došlo k nějaké chybě.



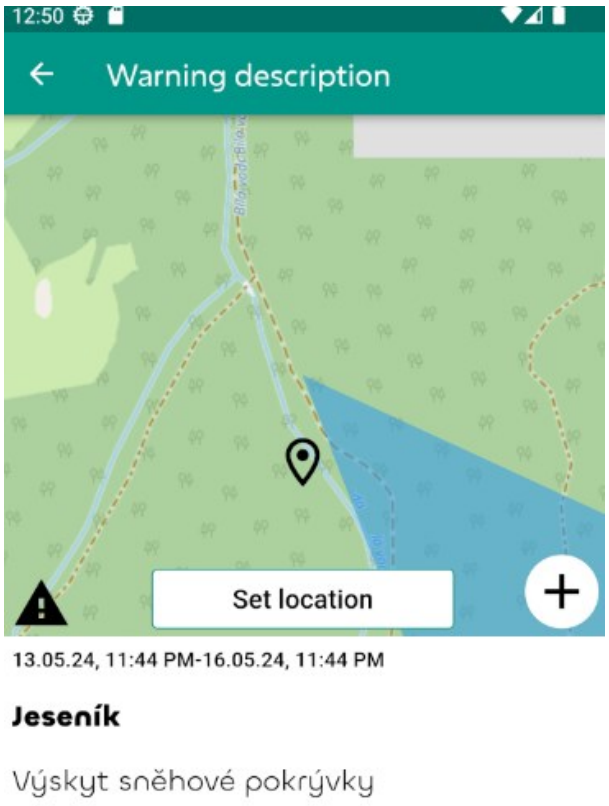
Obrázek 51 – Předpověď počasí na týden



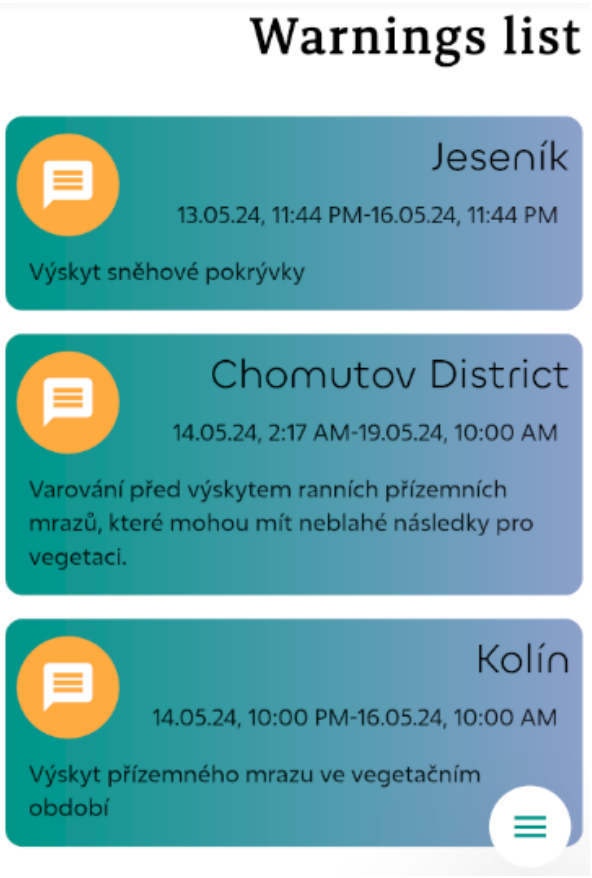
Obrázek 52 – Předpověď počasí – detail

9.6 Výstrahy

Stránka výstrah obsahuje list výstrah získaných z backend serveru. Tyto výstrahy jsou ale zadávané jak na straně serveru – backend, tak i z klientské aplikace, jednak pro testovací účely, jednak pro edukativní účely pro jiné pedagogy nebo uživatele, Z pedagogického hlediska by bylo možné průřezovými tématy použít aplikaci pro matematiku, fyziku, programování mobilních aplikací. Reálným příkladem uplatnění může být například odeslání výstrahy o události začátku deště, ze které je možné zjistit rychlost odeslání, rychlost větru v korelaci s časem a místem – pokud je vzdálenost od místa zadání 600 metrů a rychlost je 10 m/s, za jak dlouho se dostaví přeháňka ke mně, a podobně. Vzhled stránky je koncipován pro získání historie všech výstrah s přesměrováním na detailní list, obsahující všechny charakteristické znaky k výstraze a legendu s vysvětlivkami.



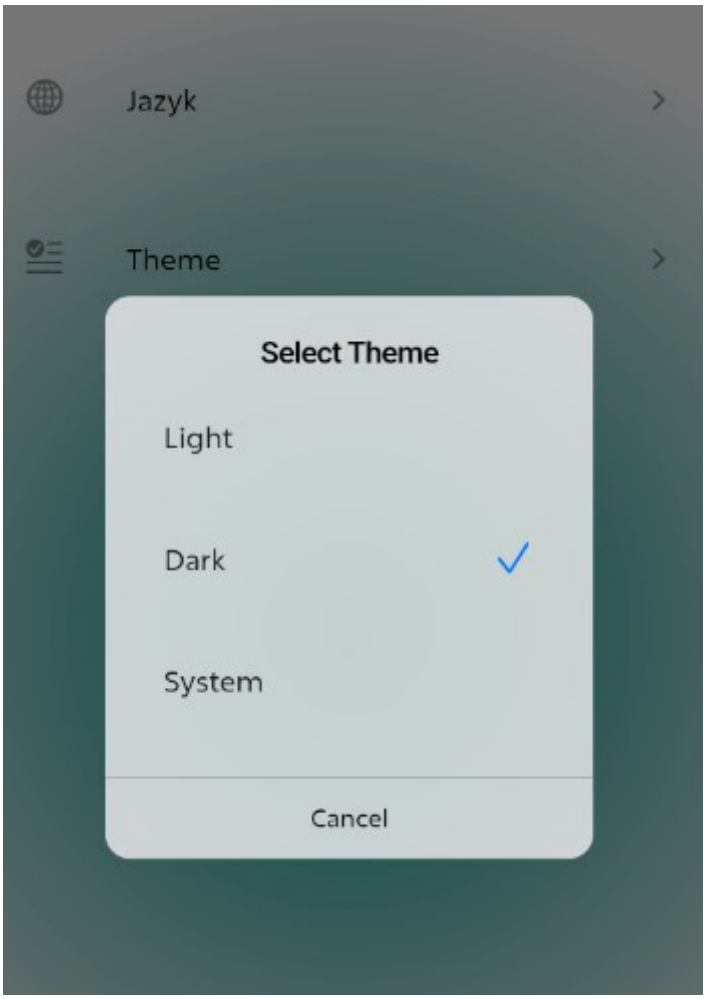
Obrázek 54 – Detail výstrahy



Obrázek 53 – List výstrah

9.7 Stránka nastavení

Aplikace nabízí dohromady tři nastavitelné palety barev, které doplňují aplikaci o kontrast a autentičnost. Nastavení lokalizace je možné přes nastavení do angličtiny, nebo češtiny. Stav se uchovává i po zavření aplikace, jelikož se hodnota uchová do paměti a do smazání nebo přepsání tam existuje.



Obrázek 55 – Nastavení vzhledu a lokalizace v aplikaci

9.8 Testování aplikace

Důležitost testů nelze podcenit v kontextu kvalitního vývoje softwaru. Správně navržená a implementovaná testovací strategie může výrazně zlepšit kvalitu a spolehlivost aplikace, snížit počet chyb a minimalizovat riziko selhání aplikace v produkčním prostředí. Zároveň umožňuje pružnější a rychlejší vývoj, protože identifikuje problémy a nedostatky již v raných fázích vývoje, což umožňuje jejich okamžité opravy. Testováním dosáhneme determinovaného stavu aplikace a můžeme jej také automatizovat a spouštět testy při vývoji.

Jedním z hlavních typů testování aplikací jsou integrační testy, které mají za cíl ověřit interoperabilitu a správné fungování různých komponent aplikace jako celku. Integrační testy jsou zvláště důležité v prostředích s architekturou mikro služeb nebo distribuovanými systémy, kde různé části aplikace musí efektivně spolupracovat. Simulace reálných scénářů a testování interakcí mezi komponentami jsou klíčovými prvky integračních testů. Máme zde testy i pro naše komponenty – widget testy, které testují vzhled a rozložení UI prvků na obrazovce, abychom změnou regresně nezměnili nebo nepoškodili fungování.

9.8.1 Widget testy

Dalším důležitým typem testování jsou widget testy, které se zaměřují na ověření správného fungování uživatelského rozhraní aplikace. Widget testy simuluji interakce uživatele s jednotlivými částmi rozhraní, jako jsou tlačítka, textová pole nebo seznamy, a ověřují, zda se chovají přesně podle očekávání a jsou uživatelsky přívětivé. Toho dosáhneme v kontextu Flutter pomocí balíčku od vývojářů flutter_test, která obsahuje následující nástroje dle dokumentace. [16]

- WidgetTester umožňuje vytváření a interakci s widgety v testovacím prostředí.
- Funkce testWidgets() automaticky vytvoří nový test WidgetTester pro každý testovací případ a použije se místo normální funkce test().
- Třídy Finder umožňují vyhledávání widgetů v testovacím prostředí. K tomu nám slouží klíč, pomocí kterého můžeme v testovacím prostředí simulovat reálné. Konstanty Matcher specifické pro widget pomáhají ověřit, zda Finder najde widget nebo více widgetů v testovacím prostředí.

9.8.2 BloC testy

V neposlední řadě jsou BloC testy, které testují správné fungování obchodní logiky aplikace. Business Logic Component (BloC) je často používaný architektonický vzor, který odděluje obchodní logiku od uživatelského rozhraní. Tyto testy kontrolují, zda se obchodní logika chová podle očekávání a vrací správné výsledky v různých situacích. Popis k těmto testům a jejich využití je popsán v oficiální dokumentaci vývojářů [13], ze které jsem primárně pro své účely čerpal.

counter_bloc_test.dart

```
blocTest(
  'emits [1] when CounterIncrementPressed is added',
  build: () => counterBloc,
  act: (bloc) => bloc.add(CounterIncrementPressed()),
  expect: () => [1],
);

blocTest(
  'emits [-1] when CounterDecrementPressed is added',
  build: () => counterBloc,
  act: (bloc) => bloc.add(CounterDecrementPressed()),
  expect: () => [-1],
);
```

Obrázek 56 – Ukázka Bloc testu a definování očekáváné hodnoty. Zdroj [13]

9.8.3 Unit testy

Tyto testy jsou užitečné pro testování malých funkčních celků, což mohou být funkce, metody, kde chceme determinovaně získat očekávaný výsledek v porovnání s reálným, a validovat tak konzistenci v chování napříč aplikací. Cílem těchto testů je zajišťování stability a spolehlivosti kódu, zejména když roste a vyvíjí se s novými funkcemi nebo změnami. [38] Unit testing zahrnuje testování nejmenších testovatelných částí kódové základny, obvykle funkcí, metod nebo tříd, aby se ověřilo, že se chovají podle očekávání.

Benefity plynou již ze samotného účelu, nicméně níže vypíšu pár bodů, proč je esenciální testovat aplikaci při produkčním provozu.

- Pomáhají udržovat kvalitu kódu
- Detekce defektů – bugů v časných stádiích vývoje
- Usnadnění přidání do agilních vývojových procesů
- Poskytování dokumentace k funkčnosti – ověření, zda test splňuje účel

Testovat můžeme pomocí pár metod, a to manuálně nebo automatizovaně.

Automatizované testování se používá k automatizaci testovacích případů – scénáři, které mohou nastat, v praxi se v testování je Flutter používá emulátor, který automatizovaně dle testů dokáže zaznamenávat, přehrávat a porovnávat očekávané výsledky. Manuální oproti tomu vyžaduje čas a kapacitu, avšak je jednoduší.

Při testování white-box zná tester vnitřní strukturu softwaru včetně kódu a může jej otestovat podle návrhu a požadavků. Naproti tomu black-box testeři nemají znalosti o vnitřních strukturách nebo kódu softwaru, neví, co a jak dopodrobna funguje, což nabízí širší škálu všech možných případů, které mohou nastat. Na obrázku níže zobrazuji testování dvou stavů ku dvěma hodnotám. První je stav, kdy nepřidáme žádnou hodnotu a žádnou neočekáváme, to označujeme klíčovým slovem expect. Přidáním hodnoty do blocu nyní očekáváme hodnotu 1, a proto ji musíme uvést. Pokud jsou podmínky splněny a očekávané hodnoty byly navráceny, testy úspěšně projdou.

```
group('CounterBloc', () {
  blocTest(
    'emits [] when nothing is added',
    build: () => CounterBloc(),
    expect: () => [],
  );

  blocTest(
    'emits [1] when CounterIncrementPressed is added',
    build: () => CounterBloc(),
    act: (bloc) => bloc.add(CounterIncrementPressed()),
    expect: () => [1],
  );
});
```

Obrázek 57 – Testování stavů

10 DISKUSE VÝSLEDKŮ

V pedagogice je evaluace důležitý proces, který je esenciální pro mentální vývoj jedince, sloužící jako odrazový můstek pro další rozvoj. V případě mé práce jsem zvolil jako nástroj pro evaluaci plošné testování aplikace žáky SŠ, na které praktikuji.

V kontextu sběru výsledků může být korelace s agilním vývojem aplikací klíčovým faktorem pro úspěšné zdokonalení meteorologické edukativní aplikace. Agilní přístup k vývoji umožňuje pružnost, rychlou reakci na zpětnou vazbu a postupné zlepšování výsledného produktu. Zavedení agilních metod a principů může podpořit efektivní iterativní proces vývoje, přizpůsobení se měnícím potřebám uživatelů a zajištění vyšší spokojenosti s koncovým produktem. Výzkum je evaluační nástroj pro sběr zpětné vazby respondenta, což jsou v mém případě žáci, kteří by potencionálně aplikaci používali. Vytvořený dotazník obsahuje takové otázky, které se dotazují na klíčové faktory fungování aplikace a uživatelský dojem. Na základě odpovědí jsem schopen vytyčené nedostatky vypilovat a agilně reagovat na změny koncového klienta. Pro zajištění chronologické nátury agilních testů, které probíhají ve sprintech, budu během tvorby aplikace v iteracích sprintu sbírat zpětnou vazbu ve formě dotazníku pro každou verzi aplikace.

10.1 Tvorba dotazníku

Z požadavků zmíněných v kapitole dříve jsem zkonstruoval dotazník pro sběr názorů a poznatků k mé vyvinuté aplikaci, určený pro žáky středních škol. Vzhledem k mé praxi na střední škole jakožto učitel jsem poprosil pár žáků o otestování aplikace a sepsání dotazníku, abych byl schopen reagovat na potřeby uživatelů.

Dotazník je tvořen otevřenými otázkami, odpovědi jsou hodnoceny stupnicí 1- nespokojený až 5 – spokojený. Poslední otázka je otevřeného typu, aby respondent mohl přidat textovou zpětnou vazbu kromě škály bodové. Jednotlivé otázky jsou rozděleny dle kategorií níže. Dotazník je přiložený na konci diplomové práce jako Příloha 5 – **Dotazník**. Dotazník zahrnuje otázky v oblastech funkčnosti, ovládání, obsahu aplikace a přesnosti dat. V následující kapitole se věnuji analyzování a vykreslování odpovědí do grafů s následnou evaluací.

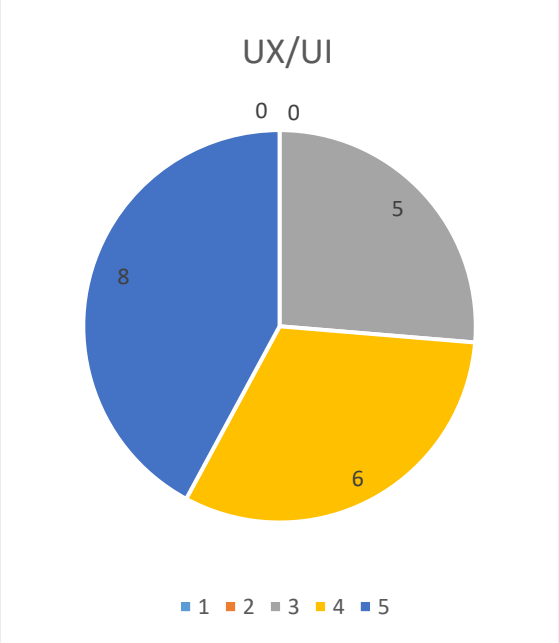
10.2 Sběr výsledků respondentů

Dotazník jsem podal žákům střední školy, kde praktikuji a kterým jsem aplikaci dal na otestování. Mimo tyto žáky jsem oslovil studenty Meteoklubu, aby si mohli aplikaci vyzkoušet.

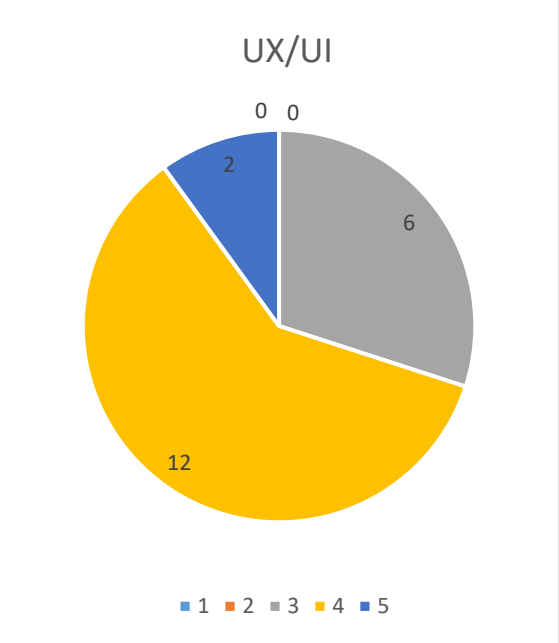
Dohromady jsem oslovil přes dvacet studentů, sesbíral výsledky, které níže dle oblasti otázky budu demonstrovat.

10.2.1 Otázka UX/UI

- Jak byste ohodnotili přehlednost aplikace?
- Jak byste ohodnotili používání aplikace z hlediska ovládání?



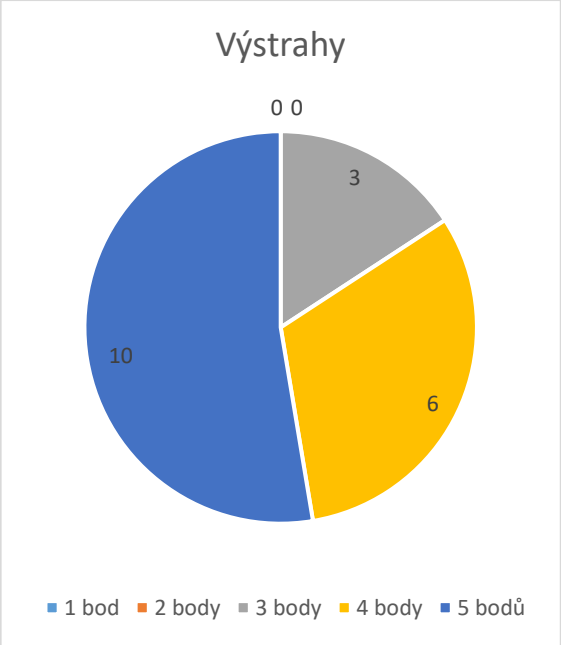
Graf odpovědí na otázku: Jak byste ohodnotili přehlednost aplikace?



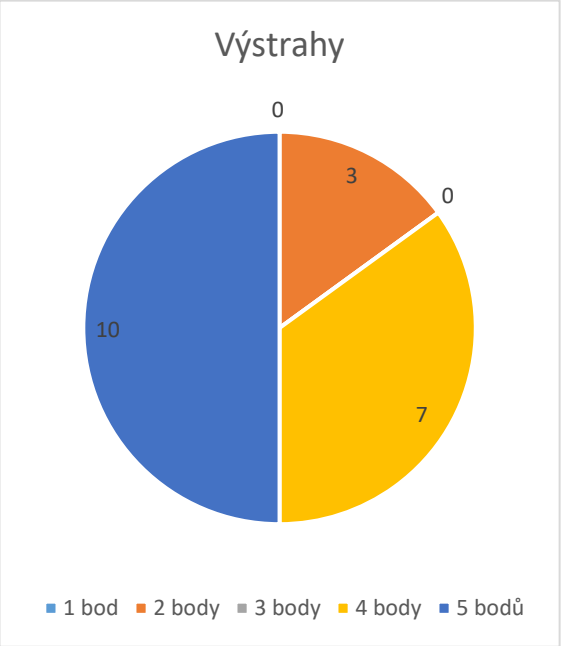
Graf odpovědí na otázku: Jak byste ohodnotili používání aplikace z hlediska ovládání?

10.2.2 Výstrahy

- Jaké je vaše hodnocení upozornění na extrémní podmínky (např. bouřky, vichřice)?
- Jsou tato upozornění dostatečně včasné a přesné?



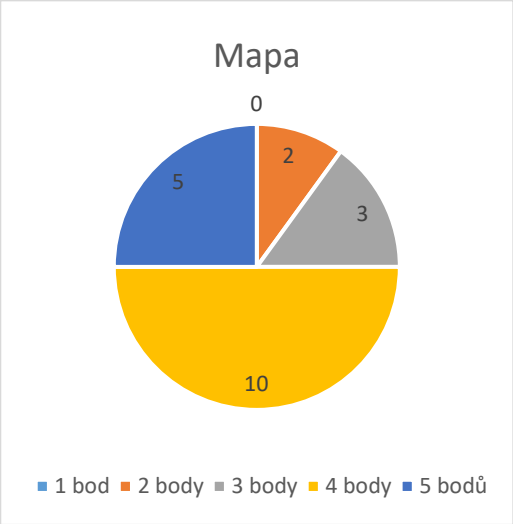
Graf odpovědí na otázku: Jaké je vaše hodnocení upozornění na extrémní podmínky (např. bouřky, vichřice)?



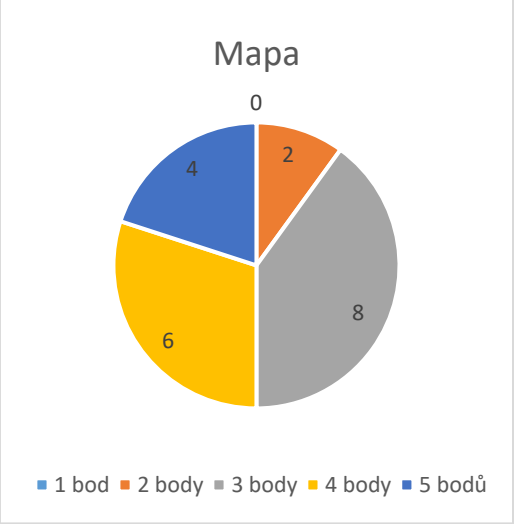
Graf odpovědí na otázku: Jsou tato upozornění dostatečně včasné a přesné?

10.2.2 Mapa

- Poskytují vizualizace dostatek informací o aktuálním počasí a meteorologických podmínkách?
- Jsou zde všechny funkce, které potřebuješ? Pokud ne, napiš, jaké bys chtěl přidat.



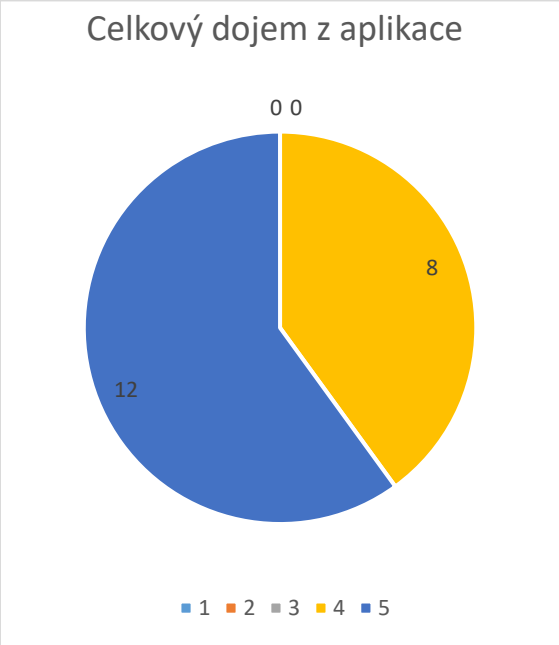
Graf odpovědí na otázku: interaktivní mapy snadno ovladatelné a přehledné?



Graf odpovědí na otázku: Jsou zde všechny funkce, které potřebuješ? Pokud ne, napiš, jaké bys chtěl přidat

10.2.2 Celkový dojem z aplikace

- Jaký je váš celkový dojem z aplikace pro meteorologii?
- Máte nějaké další připomínky nebo nápady na zlepšení naší aplikace? (Textová odpověď rozepsána v následující kapitole)



Graf odpovědí na otázku: Jak byste ohodnotili používání aplikace?

10.2.2 Připomínky

V této závěrečné kapitole rozeberu připomínky, které byly součástí dotazníku a které mají informativní hodnotu. Z počtu dvaceti respondentů jsem až na pár otázek získal přehled, jak aplikace působí na studenty z hlediska ovládání, funkcí a nápadu. Nejlépe hodnocenou funkcí aplikace jsou atlas mraků, z připomínek jsem získal zpětnou vazbu v podobě dotazu na přidávání vlastních mraků do databáze, což je naprosto validní podnět, za který jsem rád. Vzhledem k bezpečnosti a náročnosti implementace této funkcionality nezbyl čas, a proto jsem tuto připomínka dal do seznamu věcí na dodělání – backlog.

Další připomínkou byla konstruktivní kritika na nedostatečnou informovat o funkcích, kdy není zřejmé, co vše uživatel může v aplikaci dělat. Toto je opět naprosto validní připomínka, kterou jsem vyřešil přidáním úvodních stránek o funkcích aplikace, aby uživatel při prvním stáhnutí aplikace získal přehled, čeho všeho je aplikace schopná. Určitě je zde více prostoru pro rozšíření intuitivního ovládání, nicméně v reflexi na odpovědi kategorie UX/UI nebyly negativní, což je pro mě skvělou zprávou. Někteří studenti v připomínkách vyplnili zpětnou vazbu v podobě textu, v kontextu všech odpovědí z dotazníku těchto respondentů reagovali na všechny otázka známkou 5 – spokojen, tudíž nebylo co vytknout, což je pro opět pozitivní reakcí na mou aplikaci. Dále zde bylo pár připomínek, které byly vzneseny uživateli platformy iOS, na které aplikace nemá nastavenou ikonu a pozadí při načítání obrazovky, na což jsem v průběhu vývoje nemyslel a poučil jsem se tak těmito připomínkami.

Celkově vzato jsou odpovědi pozitivní, uživatelé by ocenili více barevných kombinací pozadí, někteří by aplikaci zjednodušili a některé pouze osvětlili funkce použití aplikace, což jsou připomínky, které mi napomohli zlepšit kvalitu aplikace a po kterých tento projekt završuji.

ZÁVĚR

Aplikace MeteorAtlas je projekt, který mě bavil realizovat, třebaže v něm sám vidím potenciál a základnu pro vědění, které lze v budoucnu rozšířit. Vnímám tento projekt jako kapesní pomocník pro případy, kdy se potřebuji informovat o počasí nebo varovat ostatní v případě, že hrozí výstraha. Zároveň tuto aplikaci použiji k výuce na střední škole, kde nyní praktikuji, pro výuku mobilních aplikací, kdy mohu kombinovat výuku meteorologie a informačních technologií, což mě baví a chtěl bych se věnovat i nadále. Aplikaci lze udělat robustnější a přidat více funkcí nebo dat, se kterými bude pracovat. Nabízí se třeba rozšíření o astronomii, což by mohlo upoutat zájem skupiny příznivců. Tato diplomová práce je komplexní prací, výstupem je aplikace do výuky, projektové zadání do průřezových témat výuky a samotná práce obsahuje teoretické základy potřebné pro vývoj projektu. Práce je určena pro pedagogy a studenty, kteří se zajímají o meteorologii a cílem je poskytnout pomůcku do výuky, pomocí které mohou studenti bádat o meteorologických jevech, o mracích díky atlasu mraků, využívat implementované funkce nebo vyvinout vlastní aplikaci. Vývoj mobilních aplikací je v dnešní době stále aktuální téma, dennodenně mobilní zařízení používá drtivá většina lidí a tráví zde spoustu času. V teoretické části čtenáře seznamuji s druhy vývoje software, které jsou esenciální při vývoji software v projektu a se kterými se dnes můžeme ve firmách setkat. Jako klíčové zároveň považuji sběr požadavků aplikace a prvotní analýzu, kterou je třeba před zaháním provést. Následně se věnuji rozdělení nativních a hybridních aplikací pro získání vhledu do toho, jak fungují a jaký je mezi nimi rozdíl. Já jsem zvolil pro svou aplikací vývoj multiplatformní, který je optimálnější při jednočlenném týmu, protože není potřeba psát dvě kódové základny – pro iOS a Android. V praktické části popisuji platformu Firebase a její funkce, které u sebe v aplikaci využívám a které mi ušetřili spoustu práce při vývoji díky snadné integraci. Posléze v kapitolách rozepisuji funkce, které jsem implantoval pro zvýšení uživatelského zážitku, ať už se jedná o změnu vzhledu, lokalizace nebo ukládání oblíbených lokací, a následnému ukládání do mezi paměti. Později v praktické části zobrazuji mimo implementaci kódu i obrázky z emulátoru pro přehled, jak aplikace vypadá ve skutečnosti. Zajímavostí může být například srovnání grafického návrhu aplikace s finálním, protože aplikace během půl roku prošla spousty iterací obsahujících implementaci a opravu chyb, dalo by se zde říct, že byla využita metodika „kóduj a oprav“, kterou popisuji v teoretické části práce v kapitole metodik vývoje software. Mimo realizaci jsem uvedl i testování aplikace a sběr názorů od studentů pomocí dotazníku pro získání cenné zpětné vazby, díky které mohu aplikaci dále optimalizovat. Doufám, že tato práce bude přínosem pro studenty,

pedagogy a širší veřejnost, pro stávající nebo budoucí nadšence meteorologie a programátory, kteří chtějí nebo už vyvíjejí mobilní aplikace.

SEZNAM POUŽITÉ LITERATURY

[1] DEWEY, J. The School and Society. 2nd ed. Chicago: The University of Chicago Press.1915. [cit. 2024-05-07].

[2] KALHOUS, Zdeněk a Otto OBST. Školní didaktika. Vyd. 2. Praha: Portál, 2009. ISBN 978-80-7367-571-4.

[3] MOJŽÍŠEK, L. Vyučovací metody. Praha: SPN, 1975.

[4] Ministerstvo vnitra České republiky. Online. Ministerstvo vnitra České republiky. Dostupné z: https://aplikace.mvcr.cz/sbirka-zakonu/SearchResult.aspx?q=561/2004&typeLaw=zakon&what=Cislo_zakona_smlouvy. [cit. 2024-05-08].

[5] MANAGEMENTMANIA. Vodopádový model (Waterfall model). Online. ManagementMania.com. [cit. 2024-05-07]. Dostupné z: https://managementmania.com/cs/vodopadovy-model-waterfall-model.

[6] MAŇÁK, Josef a Vlastimil ŠVEC. Výukové metody. Brno: Paido, 2003. ISBN 80-731-5039-5.

[7] BENDL, Stanislav, Anna KUCHARSKÁ a E. MARADOVÁ. Kapitoly ze školní pedagogiky a školní psychologie: skripta pro studenty vykonávající pedagogickou praxi. V Praze: Univerzita Karlova, Pedagogická fakulta, 2008. ISBN 978-80-7290-366-5.

[8] PECINA, Pavel a Lucie ZORMANOVÁ. *Metody a formy aktivní práce žáků v teorii a praxi*. Brno: Masarykova univerzita, 2009. ISBN 978-80-210-4834-8.

[9] Android Developers. Security guidelines. Android Developers [online]. [cit. 2024-2-25]. Dostupné na internete: https://developer.android.com/privacy-and-security/security-tips

[10] Web Standards. Online. W3C. Dostupné z: https://www.w3.org/standards/. [cit. 2024-05-07].

[11] App Developers: Start with Security. Online. Federal Trade Commission. Dostupné z: https://www.ftc.gov/business-guidance/resources/app-developers-start-security. [cit. 2024-05-06].

[12] Asynchronous programming: futures, async, await. Online. Dart. Dostupné z: /codelabs/async-await/. [cit. 2024-04-01].

[13] Architecture. Online. Bloc. Dostupné z: https://bloclibrary.dev/architecture/. [cit. 2024-05-01].

[14] VALIŠOVÁ, Alena a Hana KASÍKOVÁ. Pedagogika pro učitele. Vyd. 1. Praha: Grada, 2007. Pedagogika (Grada). ISBN 978-802-4717-340.

[15] Hallows, Danielle Wardinsky, "An Eye to the Sky: Describing Characteristics of Weather App Users Through Q Method" (2022). Theses and Dissertations. 9441. [cit. 2024-03-23].

[16] FLUTTER. Flutter - Dart API docs [online]. 2024 [cit. 2024-03-23]. Dostupné z: https://api.flutter.dev/

[17] The history of Snake: How the Nokia game defined a new era for the mobile industry. Online. Nenalezený vydavatel. Dostupné z: https://www.itsnicethat.com/features/taneli-armanto-the-history-of-snake-design-legacies-230221. [cit. 2024-03-28].

[18] MOBSF. Search code, repositories, users, issues, pull requests.. Online. GitHub. Dostupné z: https://github.com/MobSF/Mobile-Security-Framework-MobSF. [cit. 2024-05-07].

[19] Hansen Hsu. The Deep History of Your Apps: Steve Jobs, NeXTSTEP, and Early Object-Oriented Programming. Online. CHM. Dostupné z: https://computerhistory.org/blog/the-deep-history-of-your-apps-steve-jobs-nextstep-and-early-object-oriented-programming/. [cit. 2024-03-28].

[20] *Security*. Online. Flutter. Dostupné z: https://docs.flutter.dev/security. [cit. 2024-03-31].

[21] Beck, K. a Beedle, M. a Bennekum, A. a Cockburn, A. a Cunningham, W. a Fowler, M. a Grenning, J. a Highsmith, J. a Hunt, A. a Jeffries, R. a Kern, J. a Marick, B. a Martin, R. a Mellor, S. a Schwaber, K. a Sutherland, J. a Thomas, D.: Manifest of Agile development.

[22] *Git*. Online. Dostupné z: https://git-scm.com/. [cit. 2024-03-28].

[23] Permissions on Android. Online. Android Developers. Dostupné z: https://developer.android.com/guide/topics/permissions/overview. [cit. 2024-03-31].

[24] App permissions in Android 8: The complete guide. Online. Kaspersky official blog. Dostupné z: https://www.kaspersky.com/blog/android-8-permissions-guide/23981/. [cit. 2024-03-31].

[25] Key to Successful Unit Testing – How Developers Test Their Own Code? Online. Software Testing Help – FREE IT Courses and Business Software/Service Reviews. Dostupné z: https://www.softwaretestinghelp.com/unit-testing/. [cit. 2024-03-31].

[26] History of Android. Online. GeeksforGeeks. Dostupné z: https://www.geeksforgeeks.org/history-of-android/. [cit. 2024-05-01].

[27] Overview. Online. Stack Overflow. Dostupné z: https://survey.stackoverflow.co/2023/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2023. [cit. 2024-04-01].

[28] NETKOW, Matt a NETKOW, Matt. Ionic vs Flutter: Best Platform for Hybrid App Development. Online. Flutter Alternative. Dostupné z: https://ionic.io/resources/articles/ionic-vs-flutter-comparison-guide. [cit. 2024-04-01].

[29] Kadlec, V.: Agilní programování, Computer Press, 2004, 80-251-0342-0

[30] SCHWABER, K., SUTHERLAND, J. The Scrum Guide— The Definitive Guide to Scrum: The Rules of the Game. [online]. 2013 [cit. 20-5-2024]. Dostupné z www: https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf

[31] MYSLÍN, J. Scrum: průvodce agilním vývojem softwaru. Brno: Computer Press, 2016. ISBN 978-80-251-467. [cit. 2024-05-01].

[32] Architecture overview. Online. Android Open Source Project. Dostupné z: https://source.android.com/docs/core/architecture. [cit. 2024-05-01].

[33] App Review Guidelines. Online. Apple Developer. Dostupné z: https://developer.apple.com/app-store/review/guidelines/. [cit. 2024-05-01].

[34] FCM Architectural Overview. Online. Firebase. Dostupné z: https://firebase.google.com/docs/cloud-messaging/fcm-architecture. [cit. 2024-05-07].

[35] Flutter Favorites. Online. Dart packages. [cit. 2024-05-07]. Dostupné z: https://pub.dev

[36] How you design, align , and build matters. Do it together with Figma. Online. Figma. Dostupné z: https://www.figma.com/. [cit. 2024-03-31].

[37] KODYTEK, Samuel. Lekce 1 - Úvod do metodologie vývoje softwaru. Online. Úvod do metodologie vývoje softwaru. Dostupné z: https://www.it-network.cz/uvod-do-metodologie-vyvoje-softwaru. [cit. 2024-05-08].

[38] An introduction to widget testing. Online. Flutter. Dostupné z: https://docs.flutter.dev/cookbook/testing/widget/introduction. [cit. 2024-03-31].

[39] Talsec. Flutter Security Sdk. Online. Dostupné z: https://www.talsec.app/flutter-security. [cit. 2024-05-08].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

RASP – Runtime Application Self-Protection

SDK – Software development kit

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

Příloha 1 – Projektová výuka Meteorologie

Téma: Meteorologie – (předpovědi nadcházejícího počasí, odhadování oblačnosti z atlasu mraků, diagnostikování nebezpečných jevů)

Cíl projektu

Tvorba vlastních předpovědí podle numerických modelů (v létě speciálně zaměřené na predikci bouřek a nebezpečných doprovodných jevů). Tyto data žáci porovnají s daty v aplikaci a následně vytvoří výstup v podobě prezentace. Kromě porovnávání dat z matematických modelů vůči aplikaci využívající data z meteostanice verifikujte reálná data s předpovědními ze školní meteostanice.

Zadání projektu

Studenti budou pracovat ve skupinách na analýze meteorologických modelů a sbírání dat. Vyberte si libovolné dny v minulosti a vytvořte předpověď počasí na tyto dny na základě získaných hodnot z matematických modelů. Porovnejte tyto hodnoty mezi sebou a následně výsledky porovnejte s reálnými hodnotami.

Očekávané výstupy

- **Prezentace**: Detailní prezentace s naměřenými výsledky meteorologických dat z meteostanice, vybraných matematických modelů a reálných dat
- **Předpověď**: Výstup v podobě porovnání předpovědi počasí vytvořené žáky s reálnými daty
- **Zpráva**: Komplexní zpráva zahrnující popis, cíle a vyhodnocení projektu.

Kritéria hodnocení

1. Analýza dat: Kvalita a přesnost analýzy meteorologických dat.
2. Měření: Správnost měření a porovnání dat mezi sebou
3. Vědecká metodika: Dodržování vědeckých principů při výzkumu a analýze.
4. Prezentace: Vzhled a průběh prezentace včetně vizuálního zpracování.
5. Spolupráce: Spolupráce ve skupině, rozdělení práce úměrně.

Příloha 2 - Projektová výuka Informační technologie

Téma: Informační Technologie – (vývoj vlastní nebo stávající aplikace pomocí Git platformy, databáze a databázové systémy, komunikace API zařízení, vytvoření nových funkcí aplikace, optimalizace kódové základny, testování aplikace)

Zadání projektu

Studenti budou pracovat na vývoji mobilní aplikace s interaktivními funkcemi.

Navrhnou uživatelské rozhraní (UI) a vytvoří grafický design aplikace.

Projekt bude obsahovat prototyp a prezentaci aplikace.

Cíl projektu

Cílem projektu je vývoj mobilní aplikace, která bude obsahovat interaktivní funkce pro výuku a grafický design. Studenti se naučí základy vývoje aplikací, programování a grafického návrhu. Zvolit a popsat zvolenou metodu vývoje software v případě tvorby vlastní aplikace a otestovat ji.

Očekávané výstupy

- Mobilní aplikace: Funkční prototyp mobilní aplikace s interaktivními prvky.
- Grafický návrh: Návrh uživatelského rozhraní a grafického designu.
- Prezentace: Prezentace vývoje aplikace a vysvětlení technologických řešení.
- Kód: Použijte GitHub pro verzování vašeho kódu, přiložte odkaz na váš repositář do prezentace

Kritéria hodnocení

1. Funkčnost: Míra funkčnosti a bezchybného provozu aplikace.
2. UX/UI: Kvalita a estetika grafického návrhu.
3. Programování: Kvalita kódu a použité technologie.
4. Prezentace: Jasnost a efektivnost prezentace aplikace.
5. Spolupráce: Účinnost spolupráce ve skupině a koordinace prací.

Příloha 3 – Projektová výuka Matematiky

Téma: Matematika

Zadání projektu

Zvolte si numerický modely z dostupných zdrojů pro předpověď nacházejícího počasí a jevů, které mohou nastat. Vysvětlete, jak funguje zpracování modelů předpovědi počasí a jaké modely jste využili. Nakonec vytvořte předpověď a porovnejte ji s reálnými daty, které nastaly. Měření zpracujte a popište, jak přesné bylo měření.

Projekt bude zahrnovat výstup s prezentací výsledků a závěrů.

Cíl projektu

Cílem projektu je najít a analyzovat numerické modely pro predikci bouřek a nebezpečnými doprovodnými jevy.

Očekávané výstupy

- Prezentace: Prezentace obsahující popis numerických modelů, zpracování a průběh tvorby předpovědi
- Zpráva: Dokumentace zpracování

Kritéria hodnocení

1. Přesnost: Přesnost matematických výpočtů a modelů
2. Porovnání s Reálnými Daty: Schopnost porovnat modely s reálnými daty
3. Prezentace: Jasnost a srozumitelnost prezentace výsledků
4. Spolupráce: Účinnost spolupráce ve skupině
5. Dokumentace: Kompletnost a srozumitelnost zprávy

Příloha 4 - Projektová výuka Multimédií

Téma: Multimédia / Marketing

Zadání projektu

Práce s multimédii pro tvorbu obsahu aktivit Meteoklubu na sociální sítě. Vytvoření alespoň pěti příspěvků a následnému použití vhodného analytického nástroje pro měření dosahu příspěvků

Cíl projektu

Rozšířit základnu nadšenců do meteorologie za pomocí využití sociálních sítě, komunikováním multimediálních souborů a výtvorů. Primárním cílem je rozšířit uživatelskou základnu Meteoklubu, jejich aktivitách

.

Očekávané Výstupy

- Prezentace nebo videonahrávka historických událostí spojených s nebezpečnými jevy
- Report o výsledcích dosahu na sociálních sítích

Kritéria Hodnocení

1) Měření úspěšnosti: Jaký je dosah projektu?
2) Kvalita zpracování: Jak detailně je zpracován materiál?
3) Prezentace: Jasnost a srozumitelnost prezentace výsledků
4) Spolupráce: Účinnost spolupráce ve skupině
5) Cíl: Cíl projektu splněn (ano/ne)

Příloha 5 – Dotazník

Děkuji za naleznutí času pro vyplnění mého dotazníku určeného pro oznámkování aplikace Meteoratlas. Dotazník je anonymní, nemusíte vyplňovat osobní údaje, pokud sami nechcete. Známkování je formou bodů a to 1 – nespokojený až 5 – spokojený. Poslední je otevřená otázka pro připomínky.

UX/UI

- Otázka č.1 - Jak byste ohodnotili přehlednost rozhraní aplikace?
- Otázka č.2 - Jak byste ohodnotili používání aplikace?

Předpověď počasí na následující dny

- Otázka č.3 - Jsou podle vás předpovědi dostatečně detailní a informativní?
- Otázka č.4 - Jsou zde všechny potřebná data, které potřebujete?

Mapa

- Otázka č.5 - Jsou interaktivní mapy snadno ovladatelné a přehledné?
- Otázka č.6 - Poskytují vizualizace dostatek informací o aktuálním počasí a meteorologických podmínkách?
- Otázka č.7 - Jsou zde všechny funkce, které potřebuješ? Pokud ne, napiš, jaké bys chtěl přidat.

Atlas mraků – vzdělávací obsah o meteorologii

- Otázka č.8 - Jak byste zhodnotili kvalitu a zajímavost vzdělávacího obsahu o meteorologii?
- Otázka č.9 - Považujete tento obsah za užitečný a přínosný?

Upozornění na extrémní podmínky

- Otázka č.10 - Jaké je vaše hodnocení upozornění na extrémní podmínky (např. bouřky, vichřice)?
- Otázka č.11 - Jsou tato upozornění dostatečně včasné a přesné?

Celkový dojem z aplikace

- Otázka č.12 - Jaký je váš celkový dojem z aplikace pro meteorologii?
- Otázka č.13 - Máte nějaké další připomínky nebo nápady na zlepšení naší aplikace?