Semestrální práce z předmětu KIV/MBKZ

Multiplatformní aplikace zobrazující předpověď počasí

Stanislav Král

A17B0260P

Západočeská univerzita v Plzni
Fakulta aplikovaných věd

Obsah

1 Analýza

1.1 Vývoj multiplatformních aplikaci pomocí frameworku Flutter

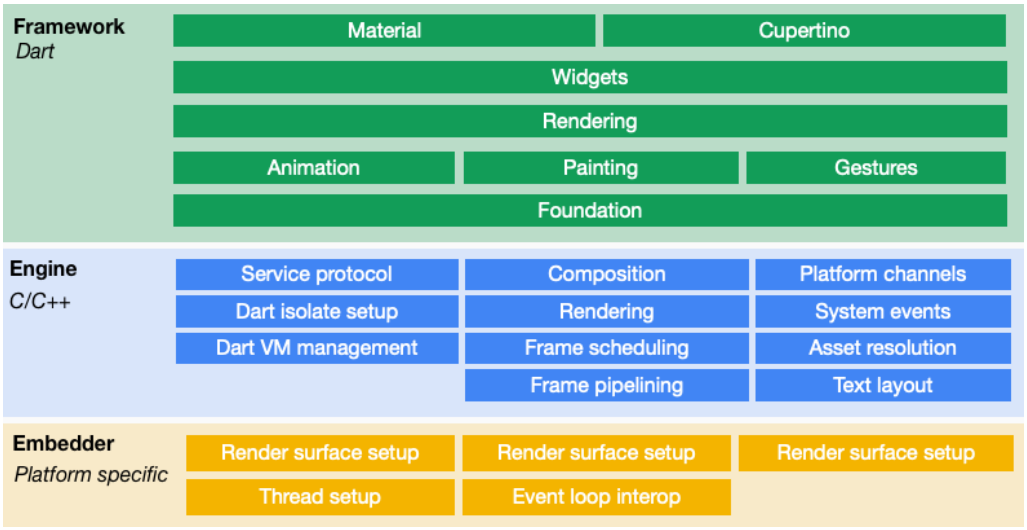
Flutter je open-source framework na jehož vývoji se podílí převážně firma Google. Cílem Flutteru je nabídnout vývojářům možnost vyvíjet výkonné aplikace, které působí nativně na všech platformách. Flutter umožňuje sdílet veškerý kód mezi všemi platformami. Toho je docíleno tak, že knihovna nepoužívá žádné nativní komponenty uživatelského rozhraní dané platformy. Okno aplikace totiž pouze slouží jako plátno a všechny komponenty si Flutter vykresluje sám. K vykreslování používá open-source knihovnu Skia, která je napsaná v jazyce C++, čímž dosahuje plynulosti ve zobrazování uživatelského rozhraní. Pro psaní Flutter aplikací se používá programovací jazyk Dart.

Flutter respektuje rozdíly v chování uživatelských rozhraní mezi mobilními operačními systémy čímž pomáhá aplikacím psaným v tomto frameworku působit tak, jako kdyby byly nativní a cílené právě pro danou platformu. Mezi hlavní rozdíly mezi platformami, jež tato knihovna implementuje na jednotlivých platformách zvlášť, patří například **scrollování**, ikony nebo typografie.

Při nasazení na Android platformu je C a C++ kód enginu je kompilován pomocí Android NDK, zatímco Dart kód se s využitím kompilace typu ahead-of-time kompiluje do nativních ARM a x86 knihoven. Tyto knihovny jsou poté použity ve vygenerované nativní Android aplikaci, která slouží jako hostitel, a z celého projektu je následně vytvořen APK balík. Při vyvíjení Flutter aplikace je použit virtuální stroj, který umožňuje upravovat zdrojový kód aplikace bez nutnosti ji restartovat. Tato funkcionalita má největší význam při tvorbě uživatelského rozhraní, kdy se jednotlivé změny v designu aplikace ihned projeví na zařízení. Podobně probíhá i nasazení aplikace na iOS platformu, kdy kód enginu je překládán pomocí LLVM. Kompilace do nativních knihoven znamená, že narozdíl od ostatních frameworků pro tvorbu multiplatformních aplikací, jsou Flutter aplikace zcela nativní.

Prvky uživatelského rozhraní, které Flutter nabízí, se snaží co nejvěrněji napodobit ty nativní. Prvky napodobující prvky platformy Android se nachází v balíku `Material` a prvky platformy iOS v balíku `Cupertino`. Tým vývojářů Flutteru bere ohledy na aktualizace mobilních operačních systémů a změny v uživatelských rozhraní včasně implementuje do svého frameworku.

Flutter dále umožňuje psát nativní kód specifický pro danou platformu pomocí konstrukce zvané `platform channel`, která funguje na principu asynchronního předávání zpráv. Část Flutter aplikace pošle hostitelské nativní



Obrázek 1: Diagram architektury Flutter

aplikaci zprávu, která asynchronně na tuto zprávu odpoví. Toto umožňuje přístup k nativnímu API dané platformy.

Na konci roku 2019, v každoročním shrnutí služby GitHub, je programovací jazyk Dart, který se v dnešní době používá nejvíce právě ve Flutter aplikacích, označen jako jazyk s nejrychleji rostoucím počtem vývojářů, jež ho používají pro vývoj aplikací.

1.2 Výběr API pro získání dat předpovědi počasí

Nutným předpokladem pro efektivní vývoj mobilní aplikace, která zobrazuje předpověď počasí, je správný výběr služby, jež zpřístupňuje API umožňující získávání dat obsahující předpověď počasí dle názvu města a tato data jsou v přehledně strukturovaná.

1.2.1 Služba OpenWeather

Služba OpenWeather [1] nabízí pro registrované uživatele tarifní plán zdarma, v rámci kterého lze přistupovat k API této služby, jež poskytuje dle názvu města data obsahující předpověď počasí pro danou oblast. Pro použití tohoto API je však nutné si nechat vygenerovat API klíč, který musí být použit při každém volání tohoto API. Data poskytována tímto API jsou dostupná ve formátu JSON, XML nebo HTML. Tato služba nabízí krátkodobou i dlou-

[1] OpenWeather - https://openweathermap.org/api

hodobou předpověď počasí a plán zdarma je omezen 1000 přístupy k API denně.

1.2.2 Služba Weatherbit

Tato služba, jež poskytuje v rámci svého zdarma tarifního plánu API poskytující předpověď počasí, umožňuje registrovaným uživatelům až 500 přístupů k předpovědím zdarma [2]. Toto API je pečlivě popsané a dokonce nabízí k dispozici dokumentaci pomocí služby **Swagger**.

1.2.3 Služba MetaWeather

Služba MetaWeather [3] slouží jako agregátor dat předpovědi počasí ze služeb BBC, **Forecast.io**, **HAMweather** nebo například **OpenWeather**, čímž slibuje větší přesnost svých předpovědí. API, které tyto agregované předpovědi poskytuje, je dostupné zdarma a bez jakýchkoliv omezení. Uživatel tohoto API je však povinen ve své práci uvést odkaz na webovou stránku této služby a neměl by toto API používat ke komerčním účelům.

Poskytované API je velmi jednoduché a umožňuje vyhledávat konkrétní předpovědi pro dané lokace pomocí názvu měst. Nevýhodou této služby je to, že počet měst, pro které poskytuje předpověď počasí, je velmi malý. Například pro Českou republiku poskytuje předpověď pouze pro Prahu.

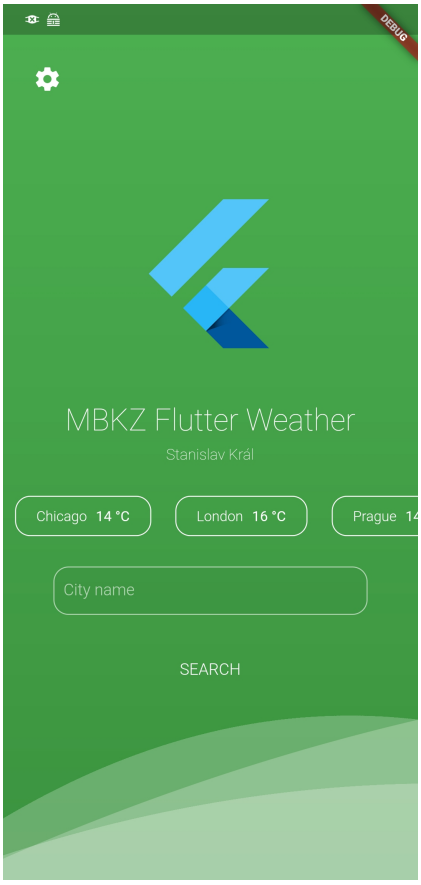
[2] Weatherbit - https://www.weatherbit.io/api
[3] MetaWeather - https://www.metaweather.com/

2 Uživatelské rozhraní vytvořené aplikace

Aplikace implementuje vlastní design uživatelského rozhraní, který je identický na obou hlavních mobilních platformách, a tak se přímo neřídí doporučeným designem iOS nebo Android aplikací.

2.1 Úvodní obrazovka aplikace

Na úvodní obrazovce aplikace je zobrazen název aplikace a logo frameworku Flutter. Pod těmito komponentami je umístěn seznam měst, ke kterým si uživatel již v minulosti vyhledal předpověd počasí. Pro tato města je automaticky získána aktuální předpověď počasí. Následně se pod tímto seznamem nachází vstupní pole, které slouží pro zadání názvu města, pro něhož si uživatel přeje zjistit předpověď počasí. Tlačítkem, jež je umístěno ve spodní části obrazovky, se spustí vyhledávání předpovědi počasí pro dané město.



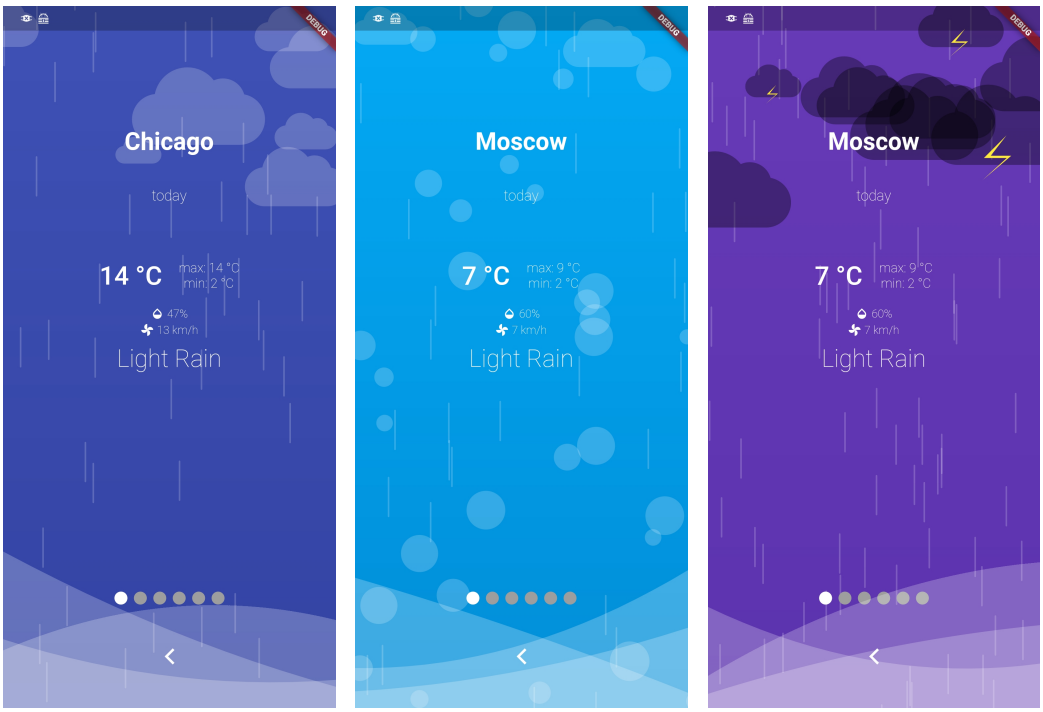
Obrázek 2: Úvodní obrazovka aplikace

Pokud uživatel klikne na nějakou položku ze seznamu již vyhledaných měst, tak se otevře detailní předpověď pro vybrané město. Dlouhým stisknutím prstu na vybranou položku je tato položka z tohoto seznamu odebrána.

2.2 Obrazovka předpovědi počasí

Na této obrazce se nachází název města, ke kterému je aktuálně zobrazována předpověd počasí. Pod názvem města je zobrazené datum, ke kterému se předpověď vztahuje. Předpověď popisuje očekávané počasí za jeden konkrétní den. Uprostřed obrazovky je zobrazena předpovídaná průměrná teplota, minimální teplota a maximální teplota za daný den. Pod teplotami jsou zobrazeny informace o vlhkosti a rychlosti větru.

Horizontálním táhnutím prstu po obrazovce lze přepínat mezi předpověďmi na nadcházející dny (celkem 5 předpovědí). Každý hlavní typ počasí, které v daný den panuje, je reprezentován animovaným pozadím této obrazovky, jež se při změně typu mění a vizualizuje tak například když má v ten den pršet, sněžit nebo pokud má být jasná obloha.

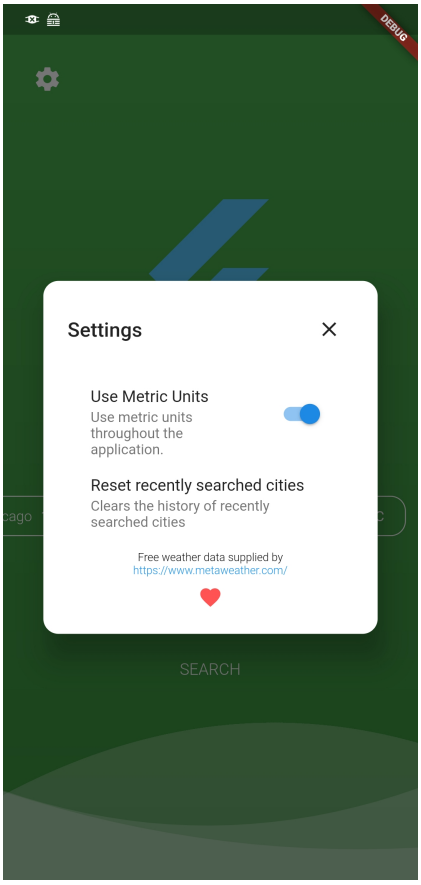


Obrázek 3: Ukázka různých typů počasí, jež aplikace vizualizuje

2.3 Dialog s nastavením aplikace

V tomto dialogu, jež se otevře po kliknutí na ikonu ozubeného kola umístěného na v levém horním rohu úvodní obrazovky, má uživatel možnost pomocí přepínače přepnout nastavení, jestli se mají v aplikaci používat metrické nebo imperiální jednotky. Tato preference je uložena napříč jednotlivými spuštěními aplikace.

Pod přepínačem metrických a imperiálních jednotek se nachází tlačítko, jež slouží pro vymazání historie jich vyhledaných měst.



Obrázek 4: Dialog s nastavením aplikace

Kliknutím na odkaz použité služby pro získávání předpovědi počasí je ve výchozím internetovém prohlížeči zařízení otevřena domovská stránka této služby.

3 Popis implemtance

3.1 Správa stavů aplikace

Při vývoji mobilních aplikací ve frameworku Flutter se často setkáváme s pojmem **state management**, jež lze volně přeložit do češtiny jako **správa stavů aplikace**. Tento po jem se při využití tohoto frameworku používá proto, že na rozdíl od vývoje nativních aplikací se pro tvorbu uživatelského rozhraní v knihovně Flutter využívá spíše deklarativního paradig matu, kdy se jasně deklarují jednotlivé stavy, ve kterých se aplikace může nacházet. Základní myšlenku tvorby uživatelského rozhraní v tomto frameworku představuje následující rovnost:

$$user\_interface = f(application\_state)$$

V praxi to znamená, že v celém kódu Flutter aplikace například nelze nalézt žádné imperativní volání nastavení textu komponenty pro zobrazení textu na obrazovce, zatím co se například na platformě Android v nativním vývoji pro tuto operaci používá metoda `setText`. Každá část aplikace může mít vlastní stav, který se v kódu implementuje tak, že se vytvoří potomek třídy `State` a nadefinují se jednotlivé atributy tohoto stavu. Při aktualizaci stavu se tato celá část aplikace vykreslí znovu. Tento koncept lze nejlépe pochopit na následující ukázce kódu:

```
1  class _MyHomePageState extends State<MyHomePage> {
2    int _counter = 0;
3    void _incrementCounter() {
4      // setState method forces the UI to redraw by
5      // saying the state of the screen has changed
6      setState(() {
7        _counter++;
8      });
9    }
10   @override
11   Widget build(context) {
12     return Scaffold(
13       body: Center(
14         child: Text(
15           'Counter value: ' + _counter.toString(),
16         ),
17       ),
18       floatingActionButton: FloatingActionButton(
19           onPressed: _incrementCounter),);
20   }
21 }
```

Ukázka kódu 1: Jednoduchá demonstrace aktualizace uživatelského rozhraní

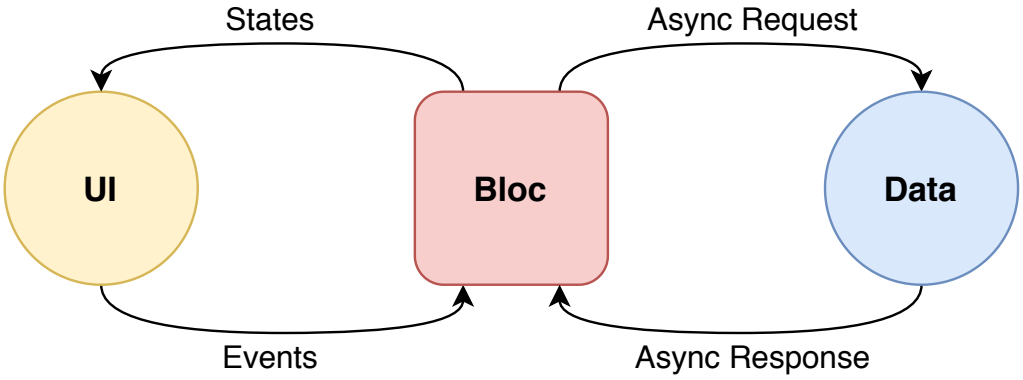
ve frameworku Flutter.

Avšak při vývoji složitejší aplikace brzy zjistíme, že při definici velké množiny stavů se bez sofistikovanější správy stavů kód aplikace brzy stane nepřehledným. Přístup ukázaný v ukázce 1 slouží jen k demonstraci konceptu aktualizace uživatelského rozhraní a nepředstavuje vhodné řešení managementu stavů aplikace. V praxi se zásadně používají komunitou vytvořené knihovny, mezi které patří například `provider`, `MobX`, `redux` nebo `flutter_bloc`. Pro správu stavů obrazovek a komponent byla zvolena knihovna `flutter_bloc`, jež implementuje architekturu **BloC**.

3.2 Použitá architektura pro správu stavů

Ve vytvořené aplikaci je pro správu stavů použita architektura BloC (Bussiness Logic Components), jež rozděluje kód aplikací do tří hlavních částí:

- **data** – datová část je zodpovědná za uložení a manipulaci s daty aplikace;
- **byznysová logika** – úkolem této vrstvy je reagovat na události z prezentační vrstvy novými stavy aplikace;
- **prezentační vrsva** – uživatelské rozhrání aplikace je vykreslováno na základě získaného stavu z byznysové logiky.



Obrázek 5: Diagram architektury BloC

Obsluha jednotlivých událostí je asynchronní a prezentační vrstvě může být během jedné obsluhy události předáno několik stavů.

3.2.1 Management stavů procesu získávání předpovědi počasí

Pro management stavů procesu získávání předpovědi počasí je vytvořena komponenta byznysové logiky `WeatherBloc`, jež reaguje na následující události:

- **FetchWeather** – událost, která je vyvolána při vyhledání předpovědi počasí dle názvu města. Obsluha této události se nejdříve dotází MetaWeather API na ID lokace dle názvu města. Pokud je dle názvu města nalezena nějaká lokace, tak tato obsluha dále již konkrétní předpověď pro nalezenou lokaci. Před vyhledáním předpovědi je vrácen stav `WeatherLoading`. Po úspěšném získání předpovědi je vrácen stav `WeatherLoaded`. V případě neúspěchu získávání je vrácen stav `WeatherError`
- **FetchWeatherByLocationId** – obsluha této události je velmi podobná obsluze události `FetchWeather` s tím rozdílem, že již je známé ID lokace ke které chceme získat předpověď, tudíž můžeme pomocí API vyhledat rovnou předpověď pro danou lokaci. Tato událost je vyvolávána položkami v historii již vyhledaných lokací.

3.2.2 Management stavů procesu získávání předpovědi počasí

Stavy animovaného pozadí, které se aktualizují při změně předpovědi počasí, jsou spravovány logikou `WeatherBackgroundBloc`, jež reaguje na následující události:

- **WeatherBackgroundChanged** – událost, která je vyvolána při změně dne, pro který si chce uživatel zobrazit předpověď počasí. Pokud je mezi přepínanými dny odlišné počasí (např. slunečno a deštivo), je vrácen stav `WeatherBackgroundState` obsahující nový stav počasí. Vrácení tohoto stavu je provedeno až po krátké časové prodlevě, aby byla změna pozadí plynulá.

3.2.3 Management stavů procesu uložení historie vyhledávání

Ukládání lokací, pro které uživatel vyhledal předpověď počasí, je řešeno v rámci logiky `WeatherHistoryBloc`, jež reaguje na následující události:

- **AddRecentlySearchedCity** – událost, která je vyvolána poté, co uživatel vyhledá předpověď počasí pro nějakou lokaci. ID této lokace je uloženo do preferencí aplikace. Po uložení tohoto ID do preferencí je vrácen stav `WeatherHistoryLoaded`, ve kterém jsou uvedeny všechny lokace, jež uživatel v minulosti vyhledal.

- **LoadRecentlySearchedCities** – při zobrazení úvodní obrazovky aplikace je vyvolána tato událost, jejíž obsluha načte seznam lokací, pro které již uživatel v minulosti vyhledal předpověď počasí. V případě, že není historie vyhledávání prázdná, je vrácen stav `WeatherHistoryLoaded`, jinak je vrácen stav `WeatherHistoryEmpty`.
- **ClearRecentlySearchedCities** – obsluha této události smaže historii vyhledaných lokací z preferencí aplikace a vrátí stav `WeatherHistoryEmpty`.
- **ClearRecentlySearchedCity** – pokud si uživatel přeje odebrat dříve vyhledanou lokaci z historie, je vyvolána tato událost, jejíž obsluha smaže vybranou lokaci z preferencí a následně vrátí stav `WeatherHistoryLoaded` nebo `WeatherHistoryEmpty` podle toho, zdali v historii zůstala nějaký záznam.

3.2.4 Management stavů nastavení aplikace

Logika nastavení aplikace je definována v komponentě `SettingsBloc` a obsluhuje následující události:

- **UnitsToggled** – událost, která je vyvolána, když uživatel mění, jaké jednotky se mají v aplikaci používat. Tato obsluha přepíná mezi zobrazováním metrických a imperiálních jednotek. Toto nastavení se ukládá do preferencí aplikace. Je vrácen aktualizovaný stav `SettingsState`.

3.3 Popis modulů aplikace

3.3.1 Modul „blocs“

Obsahuje komponenty logiky, které byly popsány v předchozí sekci.

3.3.2 Modul „models“

Obsahuje třídy, které v aplikaci definují strukturu modelů počasí a lokace.

3.3.3 Modul „pages“

Obsahuje definici obrazovek uživatelského rozhraní:

- `initial_page.dart` – soubor obsahující definici uživatelského rozhraní úvodní obrazovky aplikace;
- `weather_page.dart` – soubor obsahující definici uživatelského rozhraní detailu předpovědi počasí pro danou lokaci.

3.3.4 Modul „physics“

Obsahuje třídy definující simulaci pohybu jednotlivých částicí, které jsou zobrazovány v animovaném pozadí předpovědi počasí.

3.3.5 Modul „repositories“

Obsahuje prostředky pro získání předpovědí počasí ze služby MetaWeather. Také je zde definováno perzistentní lokální úložiště, které implementuje třída `PreferencesClient` ukládající data do preferencí aplikace.

3.3.6 Modul „utils“

Definuje funkce pro používané při formátování výpisu teploty a převodu mezi jednotkami.

3.3.7 Modul „widgets“

Obsahuje definici znovupoužitelných widgetů uživatelského rozhraní, které se používají napříč celou aplikací. Je zde například definován widget představující tlačítko, který při zobrazení načte předpověď počasí pro danou lokaci.

3.3.8 Modul „painters“

Modul, jež se nachází v modulu `widgets`, ve kterém jsou definovány widgety, které slouží k vykreslování jednotlivých částic animovaného pozadí, jenž vizualizuje počasí. Vykreslovány jsou sněhové vločky, kapky deště či pohybující se mraky.

4 Závěr

V rámci této semestrální práce byla pomocí open-source frameworku Flutter vytvořena multiplatformní aplikace, jež využívá bezplatné služby MetaWeather k zobrazování předpovědi počasí. Předpovědi lze vyhledat podle názvu lokace. Pro již vyhledané lokace je při startu aplikace zobrazná aktuální předpověď počasí. Aplikace implementuje univerzální design, který je použitelný na obou platformách Android i iOS , přičemz implementuje plynulé animace a vizualizuje stavy počasí. Testovaná byla na mobilních zařízeních Oneplus 6 a Xiaomi Mi A1.