



SEMESTRÁLNÍ PRÁCE Z PŘEDMĚTU KIV/MBKZ

# MULTIPLATFORMNÍ APLIKACE ZOBRAZUJÍCÍ PŘEDPOVĚĎ POČASÍ

STANISLAV KRÁL

A17B0260P

ZÁPADOČESKÁ UNIVERZITA V PLZNI  
FAKULTA APLIKOVANÝCH VĚD

# Obsah

<b>1</b>	<b>Analýza</b>	<b>2</b>
1.1	Vývoj multiplatformních aplikací pomocí frameworku Flutter .	2
1.2	Výběr API pro získání dat předpovědi počasí . . . . .	3
1.2.1	Služba OpenWeather . . . . .	3
1.2.2	Služba Weatherbit . . . . .	4
1.2.3	Služba MetaWeather . . . . .	4
<b>2</b>	<b>Uživatelské rozhraní vytvořené aplikace</b>	<b>5</b>
2.1	Úvodní obrazovka aplikace . . . . .	5
2.2	Obrazovka předpovědi počasí . . . . .	6
2.3	Dialog s nastavením aplikace . . . . .	7
<b>3</b>	<b>Popis implementace</b>	<b>8</b>
3.1	Správa stavů aplikace . . . . .	8
3.2	Použitá architektura pro správu stavů . . . . .	9
3.3	Manažement stavů procesu získávání předpovědi počasí . . . . .	9
<b>4</b>	<b>Závěr</b>	<b>11</b>

# 1 Analýza

## 1.1 Vývoj multiplatformních aplikací pomocí frameworku Flutter

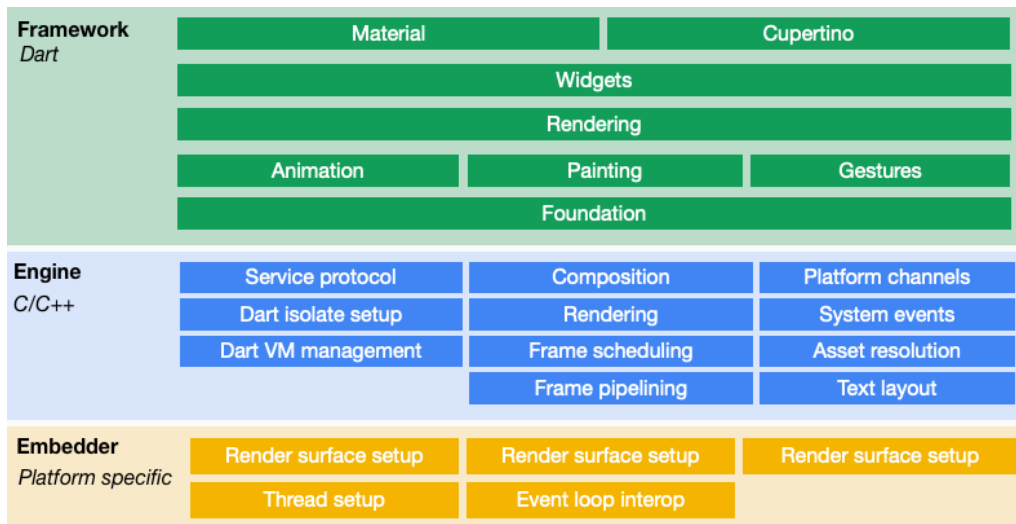
Flutter je open-source framework na jehož vývoji se podílí převážně firma Google. Cílem Flutteru je nabídnout vývojářům možnost vyvíjet výkonné aplikace, které působí nativně na všech platformách. Flutter umožňuje sdílet veškerý kód mezi všemi platformami. Toho je docíleno tak, že knihovna nepoužívá 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 zvláště, 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 funkcionality 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 včas 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 <sup>1</sup> 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 poskytovaná tímto API jsou dostupná ve formátu JSON, XML nebo HTML. Tato služba nabízí krátkodobou i dlou-

<sup>1</sup>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 <sup>2</sup>. Toto API je pečlivě popsáno a dokonce nabízí k dispozici dokumentaci pomocí služby **Swagger**.

### 1.2.3 Služba MetaWeather

Služba MetaWeather <sup>3</sup> 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.

---

<sup>2</sup>Weatherbit - <https://www.weatherbit.io/api>

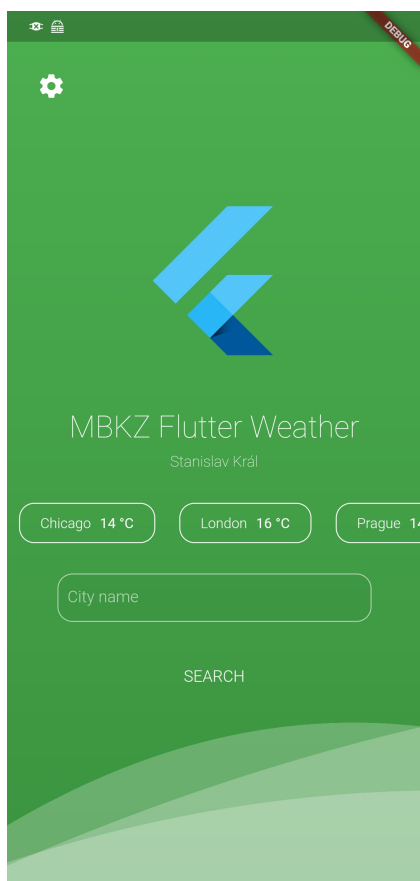
<sup>3</sup>MetaWeather - <https://www.metaweather.com/>

## 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 předpověď 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.



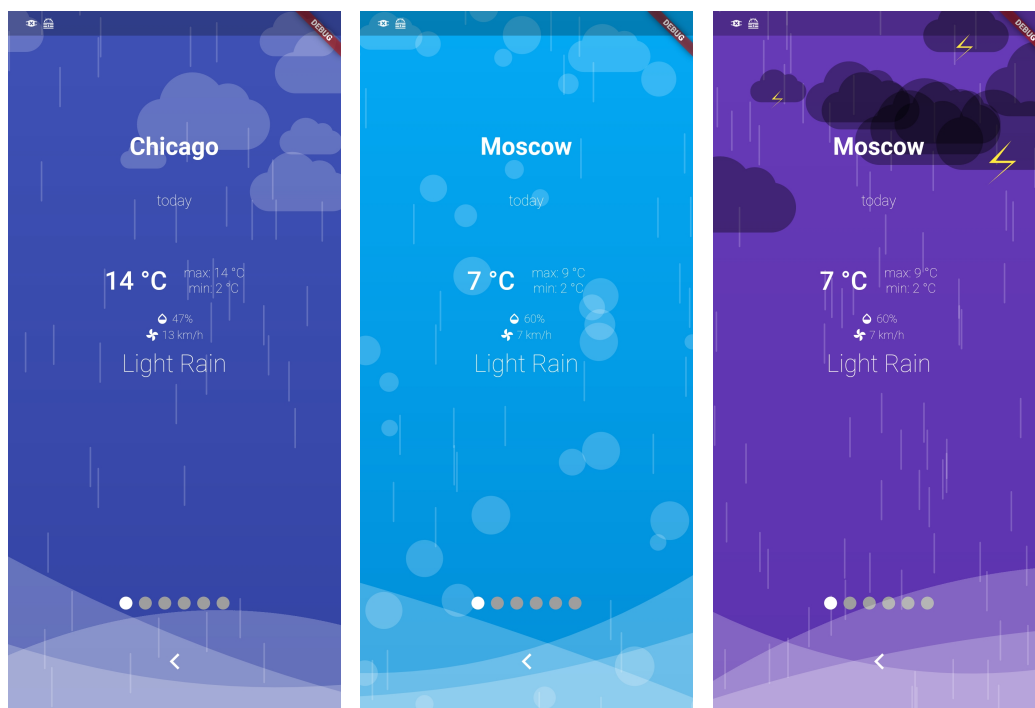
Obrázek 2: Úvodní obrazovka aplikace

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 obrazovce se nachází název města, ke kterému je aktuálně zobrazována předpověď 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.

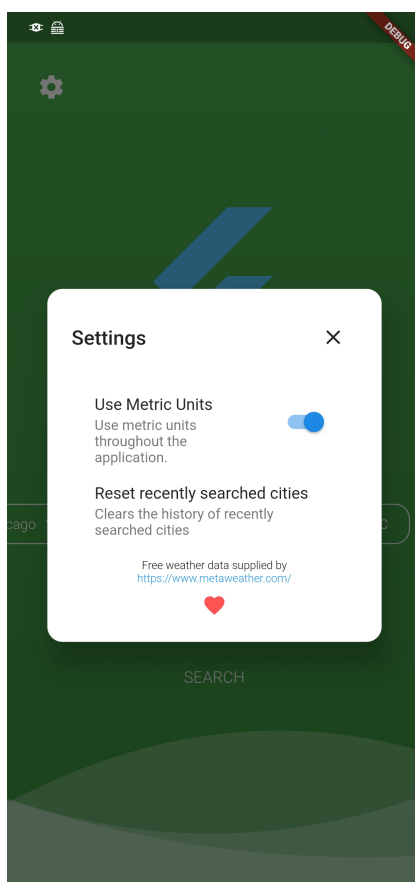


Obrázek 3: Ukázka různých typů počasí, jež aplikace vizualizuje

## 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 implementance

### 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 pojem se při využití tohoto frameworku používá proto, že narozdíl od vývoje nativních aplikací se pro tvorbu uživatelského rozhraní v knihovně Flutter využívá spíše deklarativního paradigmatu, kdy se jasně deklarují jednotlivé stavy, ve kterých se aplikace může nacházet. Základní myšlenku tvorby uživatelského rozhraní v tomto frameworku představuje následující rovnost:

$$uživatelské\_rozhraní = f(stav\_aplikace)$$

V praxi to znamená, že v celém kódu Flutter aplikace například nelze nalézt žádné imperativní volání nastavení textu komponenty pro zobrazení textu na obrazovce, kdy 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:

---

```
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;
  void _incrementCounter() {
    // setState method forces the UI to redraw by
    // saying the state of the screen has changed
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(context) {
    return Scaffold(
      body: Center(
        child: Text(
          'Counter value: ' + _counter.toString(),
        ),
      ),
    );
  }
}
```

```

    ),
    floatingActionButton: FloatingActionButton(
      onPressed: _incrementCounter),
  );
}
}

```

---

Ukázka kódu 1: Jednoduchá demonstrace aktualizace uživatelského rozhraní ve frameworku Flutter

Avšak při vývoji složitější 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 manažementu stavů aplikace. V praxi se zásadně používají komunitou vytvořené knihovny, mezi které například patří **provider**, **MobX**, **redux** nebo **flutter\_bloc**. Jeli-kož aplikace pro rozpoznávání registračních značek vozidel je netriviální aplikací, tak pro správu stavů složitějších obrazovek a komponent byla zvolena knihovna **flutter\_bloc**, jež implementuje architekturu **BloC**.

### 3.2 Použitá architektura pro správu stavů

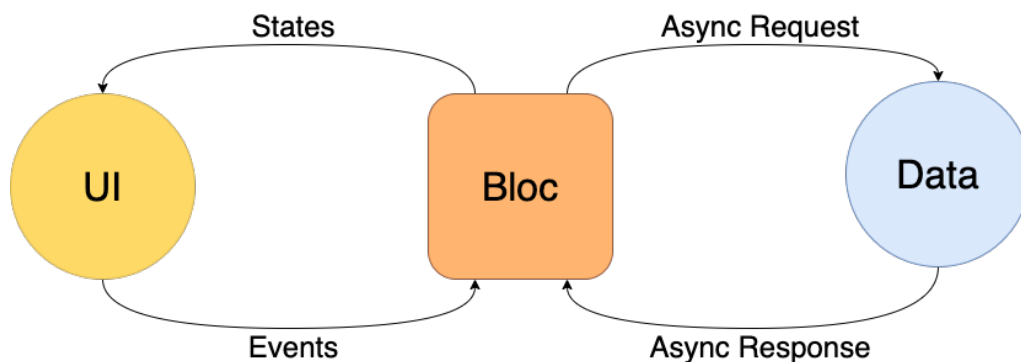
Ve vytvořené aplikaci je pro správu stavů použita architektura BloC (Business Logic Components), jež rozděluje kód aplikací do tří hlavních částí:

- **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 vrstva** - uživatelské rozhraní aplikace je vykreslováno na základě získaného stavu z byznysové logiky

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

### 3.3 Manažement stavů procesu získávání předpovědi počasí

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



Obrázek 5: Diagram architektury BloC

- **FetchWeather** - událost, která je vyvolána při vyhledání předpovědi počasí dle názvu města. Obsluha této události se nejdříve dotáže 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**
- **FetchWeatherById** - 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.

## 4 Závěr

V rámci této semestrální práce byl vytvořena dvojice programů - server a klient napodující deskovou hru Kris Kros. Důkladný a pečlivý návrh protokolu se osvědčil a v průběhu vývoje neprošel žádnými změnami. Avšak co se několikrát měnilo, byl způsob čtení zpráv, kdy nakonec jsou zprávy čteny spolehlivým automatem. Pokud bych automat navrhl hned na začátku, mohl jsem si ušetřit spoustu času laděním a upravováním parseru zpráv. Aplikace i server jsou však nyní funkční a hra je hratelná. Pokud je spojení s některým hráčem dočasně nedostupné nebo je klient násilně vypnut, tak po autorizaci je hráčovo stav korektně obnoven.