

Komentářové konvence v kódu

Definuje sadu konvencí pro komentáře v kódu, které pomáhají vývojářům rychle identifikovat různé typy poznámek a úkolů.

// TODO: Co je potřeba dodělat

Označ místo, kde je potřeba něco dodělat nebo implementovat.

```
// TODO: Přidat validaci vstupních dat
```

// FIXME: Oprava chyby

Označ místo, kde je chyba, kterou je nutné opravit.

```
// FIXME: Metoda vrací špatný výsledek při nulovém vstupu
```

// NOTE: Poznámka nebo vysvětlení

Použij pro poznámky nebo vysvětlení, proč je něco udělané určitým způsobem.

```
// NOTE: Používáme synchronní volání kvůli kompatibilitě s legacy systémem
```

// HACK: Rychlé (neideální) řešení

Označ workaround nebo neideální řešení, které funguje.

```
// HACK: Obcházíme bug v knihovně pomocí této kontroly
```

// REVIEW: Kód ke kontrole

Použij, když si nejsi jistý a chceš kód později zkонтrolovat nebo prodiskutovat.

// REVIEW: Je tento algoritmus dostatečně efektivní pro velké množství dat?

🎯 // OPTIMIZE: Prostor pro zlepšení

Označ místo, které lze časem optimalizovat.

// OPTIMIZE: Cyklus by šel paralelizovat pro vyšší výkon

Vývojové vzory – Kompletní přehled & použití

 Praktické rady pro opakovaně použitelné návrhové vzory v softwarovém vývoji.

Co jsou vývojové vzory?

- ▶  Definice & význam
-

Klasifikace návrhových vzorů

- ▶  Přehled kategorií
-

Ukázky hlavních vzorů

- ▶  Creational vzory
 - ▶  Structural vzory
 - ▶  Behavioral vzory
-

Ukázky implementace

- ▶  Singleton
 - ▶  Factory Method
 - ▶  Adapter
 - ▶  Observer
-

Další zdroje

-  Dokument vývojových vzorů (PDF)
-  Design Patterns – Refactoring Guru ↗
-  Přehled vzorů v JavaScriptu ↗



Vývojové metodiky & Konvence pojmenování

🚀 Praktické rady pro vývoj softwaru, metodiky řízení projektů, rychlé prototypování a pojmenování v kódu.



Vývojové metodiky

- ▶ ⚡ Agilní metodika (Scrum)
 - ▶ 💧 Vodopádová metodika
 - ▶ 📜 Kanban
-



Rychlé prototypování

- ▶ 🧩 Postup prototypování
-



Pojmenování BEM & Konvence v kódu

- ▶ 🧩 BEM metodika
- ▶ 📝 Konvence pojmenování v kódu



Výběr platformy pro vývoj aplikací

- Praktické rady pro výběr správné platformy a frameworku podle typu projektu.
-

Webové aplikace

- Přehled populárních frameworků
-

Mobilní aplikace

- Frameworky pro mobilní vývoj
-

Počítačové aplikace

- Frameworky pro desktop
-

Databázový vývoj

- Přehled databázových technologií
-

Herní vývoj

- Frameworky pro vývoj her
-

CI & CD

- Nástroje pro automatizaci
-

Testování

- Frameworky pro automatizované testy



.NET – Modifikátory, Runtimes & PInvoke

🚀 Praktické rady pro správu přístupu, multiplatformní nasazení, uvolnění zdrojů a volání nativních DLL v .NET.

🔒 Modifikátory přístupu

- ▶ 🔑 Přehled modifikátorů

📁 Složka **runtimes** & multiplatformní nasazení

- ▶ 💻 K čemu slouží složka `runtimes`?
- ▶ 📦 Typy nasazení

🧹 Uvolnění zdrojů

- ▶ 🗑️ Řízené vs. neřízené zdroje
- ▶ 💬 Destruktor & Dispose

🌐 Volání funkcí z externích DLL (PInvoke)

- ▶ 🔗 Jak volat nativní kód?
- ▶ ✖️ Speciální případ: `__Internal`
- ▶ 🍏 PInvoke v Unity (AppleAuth příklad)

.NET – Interface (Rozhraní)

 Praktické rady pro použití rozhraní v .NET, rozdíl mezi mělkým a hlubokým klonováním, ukázky implementace.

Co je Interface?

- ▶  Základní principy rozhraní

ICloneable – Klonování objektů

- ▶  Mělká kopie
- ▶  Hluboká kopie

.NET – Kolekce & Datové typy

 Praktické rady pro práci s kolekcemi v .NET, jejich vlastnosti, příklady použití a tipy pro efektivní správu dat.

FIFO/LIFO kolekce

- ▶  Queue (Fronta)
 - ▶  PriorityQueue
 - ▶  Stack (Zásobník)
-

Seznamy

- ▶  List
 - ▶  LinkedList
-

Slovníky

- ▶  Dictionary
 - ▶  SortedDictionary
-

Kolekce bez duplicit

- ▶  HashSet
 - ▶  Hashtable
-

Tuple kolekce

- ▶  Tuple
 - ▶  ValueTuple
-

Pozorovatelné kolekce

- ▶  ObservableCollection
-

Kolekce pouze pro čtení

- ▶  [ReadOnlyCollection](#)
 - ▶  [ReadOnlyDictionary](#)
-

Neměnné kolekce

- ▶  [ImmutableArray](#)
 - ▶  [ImmutableList](#)
 - ▶  [ImmutableDictionary](#)
 - ▶  [ImmutableHashSet](#)
 - ▶  [ImmutableSortedSet](#)
 - ▶  [ImmutableQueue](#)
 - ▶  [ImmutableStack](#)
-

Paměťové kolekce

- ▶  [Memory](#)
 - ▶  [Span](#)
-

Slabé reference

- ▶  [WeakReference](#)
-

Kolekce pro více vláken

- ▶  [ConcurrentQueue](#)
- ▶  [ConcurrentStack](#)
- ▶  [ConcurrentDictionary](#)
- ▶  [ConcurrentBag](#)
- ▶  [BlockingCollection](#)

.NET – Atributy, Datové anotace & FileHelpers

► Praktické rady pro práci s atributy v .NET, validaci dat pomocí anotací a zpracování souborů s FileHelpers.

Co jsou atributy?

-  Základní principy atributů

Datové anotace

-  Nejčastější datové anotace
-  Příklad použití datových anotací
-  Vlastní datová anotace

FileHelpers – Zpracování souborů

Important

Nepodporuje:

- Záznamy s proměnnou délkou (každý záznam musí mít stejný počet polí)
- Změnu formátu za běhu (formát musí být stejný po celou dobu běhu programu)

-  Nejčastější atributy FileHelpers
-  Příklad použití FileHelpers
-  Vlastní konvertor



.NET – Enum (Výčtové typy)

🚀 Praktické rady pro použití výčtových typů v .NET, volbu velikosti a tipy pro efektivní správu hodnot.

✳️ Co je Enum?

- ▶ 🔎 [Základní principy Enum](#)
-

📁 Typy Enum podle velikosti

- ▶ 📦 [Velikost a rozsah Enum](#)
-

💻 Příklad použití Enum

- ▶ 📝 [Ukázka deklarace a použití](#)



.NET – Metody & Parametry

► 🚀 Praktické rady pro práci s metodami v .NET, typy parametrů, delegáty, asynchronní a paralelní zpracování.



Základní pojmy

- 🔎 Parametr vs. Argument
-



Předání hodnoty vs. reference

- 📦 Hodnota
 - 🔗 Reference
-



Druhy metod

- 📖 Přehled typů metod
-



Ukázky deklarace metod

- 📋 Indexátor
 - ⚡ Statická metoda
 - 🧑 Instanční metoda
 - ↲ Virtuální metoda
 - 🔄 Abstraktní metoda
 - 🔄 Přetížené metody
 - 📝 Výchozí hodnoty parametrů
 - 1 2 3 4 Params
 - 🔗 Ref/Out/In parametry
 - ✨ Rozšířené metody
 - ⚡ Asynchronní metoda
-



Ukazatel na metody & Delegáti

- 🔗 Delegáti

- ⚡ Generické delegáty
-

Asynchronní & Paralelní metody

- ⚡ Asynchronní volání
- 📁 Paralelní zpracování (TPL)



.NET – Implicitní & Explicitní operátory

🚀 Praktické rady pro převody typů v .NET, rozdíly mezi implicitními a explicitními operátory, ukázky použití.

💡 Co jsou implicitní a explicitní operátory?

- ▶ 🔎 Základní principy převodů
-

⬅️ Implicitní operátor

- ▶ ⚡ Automatický převod
-

🛡️ Explicitní operátor

- ▶ 🔒 Převod s použitím castu

.NET – Vytvoření REST API v ASP.NET Core

 Praktické rady pro založení, strukturu, konfiguraci a rozšíření vlastního API v C# s ASP.NET Core.

Vytvoření projektu

- ▶  Krok za krokem
-

Struktura projektu

- ▶  Přehled složek a souborů
-

Přidání kontroleru

- ▶  Ukázka kontroleru
-

Konfigurace závislostí

- ▶  Nastavení v `Startup.cs`
-

Vstupní bod aplikace

- ▶  `Program.cs`
-

Konfigurace v `appsettings.json`

- ▶  Ukázka konfigurace
-

Příklad služby s konfigurací

- ▶  Třída služby

.NET – XML: Serializace, CDATA, Namespace & Konvence

 Praktické rady pro práci s XML v .NET, nahradu znaků, CDATA, serializaci/deserializaci objektů, namespace a konvence.

Náhrada znaků v XML

- ▶  Základní a speciální znaky
-

CDATA sekce

- ▶  Co je CDATA?
-

Serializace & Deserializace objektů

- ▶  Jak převést objekt na XML a zpět?
-

Namespace v XML

- ▶  Jak fungují jmenné prostory?
-

Konvence serializace XML

- ▶  Přizpůsobení serializace tříd a vlastností

.NET – NUnit (Testovací framework)

 Praktické rady pro psaní unit testů v .NET pomocí NUnit, práce s více asserty a odkazy na video prezentaci.

Co je NUnit?

- ▶  [Základní principy NUnit](#)
-

Multiple Asserts

- ▶  [Jak fungují Multiple Asserts?](#)
-

Video prezentace

- ▶  [Trendy v unit testování a mockování](#)



WPF – Moderní UI & Tipy

► 🚀 Praktické rady pro tvorbu desktopových aplikací ve WPF, stylování, datové vazby, validaci, animace a responzivní design.

✳️ Co je WPF?

- 🔎 Základní principy WPF
-

🖱️ Základní ovládací prvky

- ⏺ Button
 - 🖌️ TextBox
 - ✅ CheckBox
 - ▾ ComboBox
 - ⏹ RadioButton
 - 📂 Slider
-

🛠️ Vlastní ovládací prvky

- 💡 Jak vytvořit vlastní prvek?
-

🎨 Styls & Šablony

- 🖌️ Definování stylu
 - ✨ Použití stylu
 - 💡 Šablony (ControlTemplates)
-

🧭 Prefixy v XAML

- 📚 Přehled prefixů
-

📐 Responzivní design

- ▶  Layout Panely
 - ▶  Dynamické velikosti
 - ▶  Sledování změny velikosti
 - ▶  ViewBox
-

Triggery

- ▶  Dynamické změny stylu
-

Data Binding (Vazba dat)

- ▶  ViewModel + Binding
-

Validace

- ▶  IDataErrorInfo
 - ▶  INotifyDataErrorInfo
-

Animace

- ▶  Příklad animace

Flutter

Flutter je framework pro vývoj mobilních aplikací pro **Android** a **iOS**. Používá programovací jazyk **Dart**.

Instalace

Důležité:

Flutter používá **Git** pro správu závislostí, proto je nutné mít nainstalovaný **Git**.

Pro vývoj pro Android je potřeba mít nainstalovaný **Android Studio**.

Windows

1. Stáhněte Flutter SDK z [oficiálních stránek](#).
2. Rozbalte ZIP do složky, např. `C:\src\flutter`.

Poznámka: Cesta nesmí obsahovat mezery ani speciální znaky.

3. Přidejte cestu k `flutter\bin` do proměnné prostředí **PATH**.
4. Ověřte instalaci:

```
flutter doctor
```

5. Vypněte analyzování:

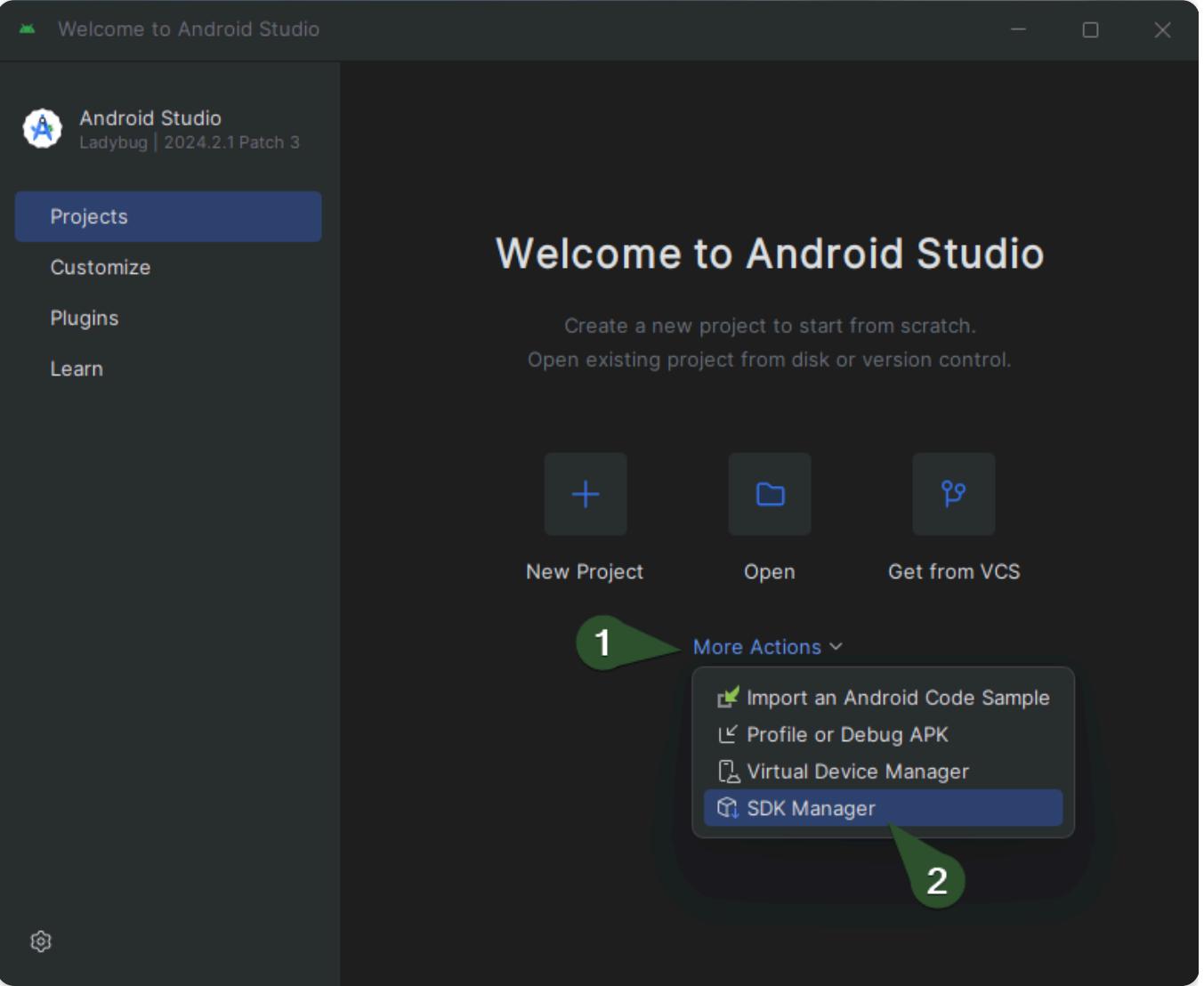
```
flutter config --no-analytics
```

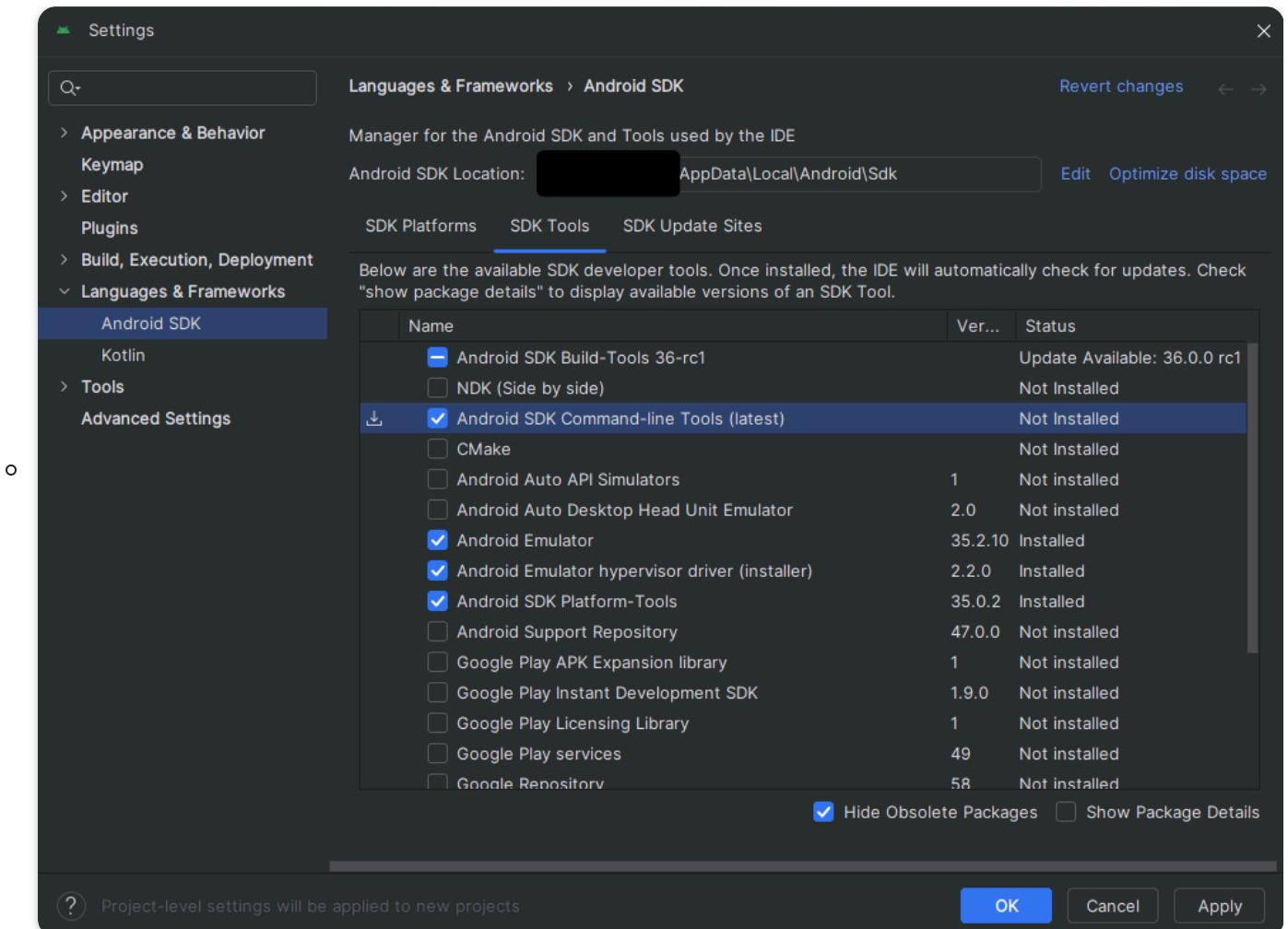
6. Pro kontrolu nastavení:

```
flutter config
```

Android toolchain

1. Ujistěte se, že je nainstalován **Android Studio**.
2. Nastavte Android toolchain podle obrázků:





Vývoj webových aplikací

Pokud chcete použít jiný prohlížeč než Google Chrome:

```
flutter config --no-web-browser
```

Spusťte aplikaci:

```
flutter run -d web-server
```

Otevřete ve vlastním prohlížeči adresu <http://localhost:PORT/>.

Vytvoření nového projektu

1. Vytvořte projekt:

```
flutter create project_name
```

2. Spusťte aplikaci:

```
cd project_name  
flutter run
```

Záloha, obnova a ukládání dat

Umístění aplikačních dat

- **shared_preferences.json:** %APPDATA%\com.example\xxx_app
 - %APPDATA% – uživatelská data (C:\Users\<uživatel>\AppData\Roaming)
 - com.example – identifikátor aplikace
 - xxx_app – název aplikace
 - shared_preferences.json – uložená data
-

Záloha závislostí

Zálohujte složku .pub-cache:

- **Windows:** C:\Users\<uživatelské_jméno>\AppData\Local\Pub\Cache
 - **macOS/Linux:** /Users/<uživatelské_jméno>/ .pub-cache
 - **hosted** – balíčky z repozitářů ([pub.dev](#))
 - **hosted-hashes** – hash soubory pro ověření integrity
 - **temp** – dočasné soubory při stahování balíčků
-

Obnova závislostí

Obnovte obsah složky .pub-cache do původního umístění.

Lokalizace (interní knihovna)

1. Přidejte do `pubspec.yaml`:

```
dependencies:  
  flutter:  
    sdk: flutter  
  flutter_localizations:  
    sdk: flutter  
  cupertino_icons: ^1.0.8  
  flutter_svg: ^2.0.16
```

2. Vytvořte lokalizační soubory:

- `lib/l10n/intl_en.arb`:

```
{  
  "@@locale": "en",  
  "hello": "Hello",  
  "welcome": "Welcome"  
}
```

- `lib/l10n/intl_cs.arb`:

```
{  
  "@@locale": "cs",  
  "hello": "Ahoj",  
  "welcome": "Vítejte"  
}
```

3. Přidejte `intl_utils` do `pubspec.yaml`:

```
dependencies:  
  flutter:  
    sdk: flutter  
  flutter_localizations:  
    sdk: flutter  
  intl_utils: ^2.5.0  
  cupertino_icons: ^1.0.8  
  flutter_svg: ^2.0.16
```

4. Vygenerujte lokalizační soubory:

```
dart pub get  
dart run intl_utils:generate
```

5. Použití lokalizace v aplikaci:

```
import 'package:flutter/material.dart';  
import 'package:flutter_localizations/flutter_localizations.dart';  
import 'generated/l10n.dart';  
  
void main() {  
    runApp(MyApp());  
}  
  
class MyApp extends StatelessWidget {  
    @override  
    Widget build(BuildContext context) {  
        return MaterialApp(  
            localizationsDelegates: [  
                S.delegate,  
                GlobalMaterialLocalizations.delegate,  
                GlobalWidgetsLocalizations.delegate,  
                GlobalCupertinoLocalizations.delegate,  
            ],  
            supportedLocales: S.delegate.supportedLocales,  
            home: MainPage(),  
        );  
    }  
}  
  
class MainPage extends StatelessWidget {  
    @override  
    Widget build(BuildContext context) {  
        return Scaffold(  
            appBar: AppBar(title: Text(S.of(context).hello)),  
            body: Center(child: Text(S.of(context).welcome)),  
        );  
    }  
}
```

Základní znalosti

Co je Riverpod?

- **Riverpod** je knihovna pro správu stavu ve Flutteru.
- Pomáhá jednoduše sdílet data (např. uživatele, nastavení, seznamy) mezi různými částmi aplikace.
- Místo toho, abys data tahal ručně, použiješ tzv. **provider** a data si "vytahneš" kdekoliv v aplikaci.

Příklad:

Chci vědět, kdo je přihlášený uživatel. Vytvořím provider a pak ho použiju v jakémkoliv widgetu.

1. Co je **StatelessWidget**?

- **Neumí si nic pamatovat.**
- Jen zobrazí UI podle vstupních dat.
- Nepoužívá žádný vnitřní stav, žádné změny, žádné reakce na události.

Použití:

Když potřebuješ jen vykreslit něco jednoduchého, co se nemění.

2. Co je **StatefulWidget**?

- **Umí si pamatovat stav.**
- Může reagovat na změny, ukládat si data, spouštět kód při startu (`initState`) a při zavření (`dispose`).
- Hodí se, když potřebuješ např. načítat data, čekat na odpověď, reagovat na kliknutí, animace apod.

Použití:

Když stránka potřebuje něco sledovat, měnit, nebo reagovat na události.

3. Co je **ConsumerWidget**?

- **Speciální typ StatelessWidgetu pro Riverpod.**
- Umožňuje jednoduše číst data z providerů (Riverpod).
- Nemá vlastní stav, ale umí reagovat na změny v datech z provideru.

Použití:

Když chceš zobrazit data z Riverpod provideru, ale nepotřebuješ vlastní stav.

Kdy použít co?

- **StatelessWidget** – jednoduché zobrazení, žádný stav, žádné změny.
- **StatefulWidget** – složitější logika, potřeba pamatovat si stav, reagovat na změny, používat `initState`/`dispose`.
- **ConsumerWidget** – jednoduché zobrazení dat z Riverpod provideru, bez vlastního stavu.

Proč není všude **ConsumerWidget**?

- **ConsumerWidget** neumí mít vlastní stav ani životní cyklus (`initState`, `dispose`).
- Pokud potřebuješ vlastní stav nebo reagovat na start/zavření stránky, použij **StatefulWidget**.
- Pokud jen čteš data z provideru a nepotřebuješ stav, použij **ConsumerWidget**.

Jednoduché příklady

```
// 1. StatelessWidget - jen vykreslí text
class JednoduchaStranka extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Text('Ahoj světe!');
  }
}

// 2. ConsumerWidget - čte data z provideru (Riverpod)
final userProvider = Provider((ref) => 'Pepa');

class HomePage extends ConsumerWidget {
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final user = ref.watch(userProvider);
    return Text('Ahoj, $user');
  }
}

// 3. StatefulWidget - umí si pamatovat stav a reagovat na změny
class PurchaseCodePage extends StatefulWidget {
  @override
  State<PurchaseCodePage> createState() => PurchaseCodePageState();
}
```

```
class PurchaseCodePageState extends State<PurchaseCodePage> {
    bool _loading = false;

    @override
    void initState() {
        super.initState();
        // Tady začíná sledování nákupu
    }

    @override
    void dispose() {
        // Tady končí sledování nákupu
        super.dispose();
    }

    @override
    Widget build(BuildContext context) {
        return _loading ? CircularProgressIndicator() : Text('Koupit aplikaci');
    }
}
```

Shrnutí v jedné větě

- **StatelessWidget** = nic si nepamatuje, jen vykreslí.
- **StatefulWidget** = umí si pamatovat, reaguje na změny, složitější logika.
- **ConsumerWidget** = jednoduché zobrazení dat z Riverpod provideru.
- **Riverpod** = knihovna na sdílení a správu dat v aplikaci.

Flutter – Přehled příkazů

|  Kategorie |  Příkaz |  Popis |
|---|--|---|
|  Verze a kanály | <code>flutter --version</code> | Zobrazí verzi Flutter SDK, Dart SDK a aktuální kanál. |
| | <code>flutter upgrade</code> | Aktualizuje Flutter SDK. |
| | <code>flutter downgrade</code> | Vrátí Flutter SDK na předchozí verzi. |
| | <code>flutter channel</code> | Zobrazí dostupné kanály (<code>stable</code> , <code>beta</code> , <code>dev</code> , <code>master</code>). |
| | <code>flutter channel stable</code> | Přepne na stabilní kanál. |
| | <code>flutter channel beta</code> | Přepne na beta kanál. |
|  Diagnostika | <code>flutter doctor</code> | Zkontroluje nastavení Flutteru. |
| | <code>flutter doctor -v</code> | Detailní výstup diagnostiky. |
|  Závislosti | <code>flutter pub get</code> | Stáhne závislosti z <code>pubspec.yaml</code> . |
| | <code>flutter pub upgrade</code> | Aktualizuje závislosti. |
| | <code>flutter pub cache repair</code> | Opraví cache závislostí. |
|  Projekt | <code>flutter create project_name</code> | Vytvoří nový projekt. |
| | <code>flutter run</code> | Spustí aplikaci. |
| | <code>flutter build apk</code> | Vytvoří produkční APK. |
| | <code>flutter build ios</code> | Vytvoří build pro iOS (vyžaduje macOS a Xcode). |
| | <code>flutter clean</code> | Vyčistí build cache. |
| | <code>flutter test</code> | Spustí testy. |
|  Zařízení | <code>flutter devices</code> | Zobrazí dostupná zařízení. |
| | <code>flutter emulators</code> | Zobrazí dostupné emulátory. |

|  Kategorie |  Příkaz |  Popis |
|---|--|---|
| | <code>flutter emulators --launch emulator_id</code> | Spustí emulátor podle ID. |
| | <code>flutter install</code> | Nainstaluje aplikaci na zařízení. |
|  Analýza a opravy | <code>dart analyze</code> | Spustí analýzu kódu. |
| | <code>dart fix --apply</code> | Aplikuje doporučené opravy. |
|  Logy | <code>flutter logs</code> | Zobrazí logy aplikace. |

Tip:

Pro více informací ke konkrétnímu příkazu použij `--help` (např. `flutter run --help`).

Pokrytí kódu (Code Coverage)

Bez nahlédnutí do zdrojového kódu

1. Nainstalujte balíček:

```
npm install -g @lcov-viewer/cli
```

2. Vytvořte `package.json`:

```
npm init -y
```

3. Přidejte skript:

```
{
  "name": "xxx_app",
  "version": "1.0.0",
  "description": "A new Flutter project.",
  "main": "index.js",
  "directories": {
    "lib": "lib",
    "test": "test"
  },
  "scripts": {
    "test-report": "flutter test --coverage && lcov-viewer lcov -o ./coverage/report ./coverage/lcov.info"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

4. Spusťte:

```
npm run test-report
```

S nahlédnutím do zdrojového kódu

1. Stáhněte [genhtml](#).
2. Umístěte soubor do kořenové složky projektu.
3. Ujistěte se, že máte nainstalovaný `Git Bash` (obsahuje perl).

- Ověřte instalaci perl:

```
where perl
```

4. Spusťte v Git Bash:

```
perl genhtml coverage/lcov.info -o coverage/html
```

 Automatické zmenšení textu bez doplňků

Použijte kombinaci `Expanded` a `FittedBox` pro automatické přizpůsobení velikosti textu:

```
Expanded(  
  child: FittedBox(  
    fit: BoxFit.scaleDown,  
    child: Text('${widget.article.price * chosenQuantity} DH',  
      style: const TextStyle(fontSize: 25, fontWeight: FontWeight.w700),  
    ),  
  ),  
)
```

 Vypnutí pravidla `no_logic_in_create_state`

Jak vypnout linter pravidlo:

1. Otevřete soubor `analysis_options.yaml` v kořenovém adresáři projektu.
2. Přidejte následující konfiguraci:

```
linter:  
  rules:  
    no_logic_in_create_state: false
```

3. Uložte soubor a restartujte IDE.

 Chyba: Building with plugins requires symlink support

Pokud se zobrazí tato chyba na Windows, je potřeba povolit `Developer Mode`:

1. Stiskněte `Win + R`
2. Zadejte `ms-settings:developers` a potvrďte
3. Povolte `Developer Mode` (Režim pro vývojáře)

Unity – Základy & Tipy pro rychlý start

 Praktické rady pro efektivní práci v Unity, rychlé prototypování a výběr správného typu projektu.

Rychlejší spuštění hry

- ▶  Jak zrychlit vstup do Play módu?
-

Výběr typu hry: 2D vs 3D

- ▶  Video: Jak vybrat 2D nebo 3D projekt?
-

Rychlé prototypování

- ▶  Video: Jak rychle prototypovat v Unity?
-

Klíčová slova v Unity

- ▶  Video: Důležitá klíčová slova v Unity
-

Vývojové vzory v Unity

- ▶  Video: Nejčastější vývojové vzory



Unity 2D – Tilemap, Sprity & Tipy

► Praktické rady pro práci s 2D grafikou v Unity, nastavení Tilemap, velikosti obrázků, animace a řešení běžných problémů.

✳️ Tilemap vs Sprite Renderer

Use cases for Tilemaps

| | When to use it | Example |
|-------------------------------------|---|---------|
| Tilemap Renderer, Chunk mode | <ul style="list-style-type: none">• Grid-based levels• No sorting required | |
| Tilemap Renderer, Individual | <ul style="list-style-type: none">• Grid-based levels• Sorting needed | |
| Sprite Renderer | <ul style="list-style-type: none">• Characters or elements that can't conform to the grid | |



Tilemap

- 🎨 Vykreslení a nastavení barvy dlaždic
- 🔎 Pravidla pro Tilemap



Velikost obrázku

- 📎 Definice velikostí
- 🛡️ Nastavení velikosti (PPU)



Animace obrázku

- 🦴 Hloubka (Depth) u kostí
 - 🦴 Univerzální Rigging
-

🛠️ Řešení chyb při vykreslování sprite

- | ● Na kameře musí být komponenta [Pixel Perfect Camera](#) pro 2D!
- ❤️ Černé čáry
- ✨ Problíkávání

UMotion – Klíčování animací v Unity

 Praktické tipy pro efektivní práci s UMotion při ukládání změn animací.

Uložení změn v UMotion

- ▶  Jak klíčovat změny?
-

Postup klíčování

- ▶  Krok za krokem

Unity – Kamera & Tipy

 Praktické rady pro nastavení kamery v Unity, rozdíly mezi ortografickou a perspektivní kamerou, a proč používat Pixel Perfect Camera v 2D hrách.

Pixel Perfect Camera

- ▶  [Proč ji použít v 2D?](#)
-

Ortografická Kamera

- ▶  [Vlastnosti & použití](#)
-

Perspektivní Kamera

- ▶  [Vlastnosti & použití](#)
-

Novinky & Video

- ▶  [Co je nového v Unity kamerách?](#)

Unity – Navigační systém & Tipy

 Praktické rady pro nastavení navigace, pohyb postav a využití NavMesh v Unity.

Co je Navigační systém?

- ▶  Základní principy
-

Nastavení NavMesh v Unity

- ▶  Jak vytvořit NavMesh?
-

Pohyb postavy pomocí NavMesh Agent

- ▶  Jak nastavit pohyb?
-

Novinky & Video

- ▶  Co je nového v navigaci?



Unity – ScriptableObject & Tipy

► Praktické rady pro použití ScriptableObject v Unity, jejich výhody, omezení a moderní patterny.

Co je ScriptableObject?

- Základní principy
-

Vytvoření ScriptableObject

- Jak vytvořit ScriptableObject?
-

Ukládání & Obnovení dat

- Omezení ScriptableObject
-

Singleton pattern se ScriptableObject

- Jak na singleton ScriptableObject?



Unity – UI systém & Tipy

🚀 Praktické rady pro práci s UI v Unity, včetně nastavení tlačítek, detekce kliknutí a užitečných vlastností komponent.

✳️ Co je UI systém v Unity?

- ▶ 🔎 [Základní principy UI](#)
-

🟡 Tlačítko (Button)

- ▶ 🖱️ [Vlastnosti tlačítka](#)
-

🟣 Rozsah detekce kliknutí podle průhlednosti

- ▶ 💧 `alphaHitTestMinimumThreshold`



Unity – UI Toolkit & Tipy

🚀 Praktické rady pro práci s UI Toolkit v Unity, jeho výhody, základní principy a moderní patterny.

🌟 Co je UI Toolkit?

- ▶ 🔎 Základní principy UI Toolkit
-

🛠️ Jak začít s UI Toolkit?

- ▶ 📦 Základní kroky
-

🖼️ UXML & USS – Příklad

- ▶ 💾 Ukázka kódu
-

🆕 Novinky & Video

- ▶ 🎥 Co je nového v UI Toolkit?

Unity – Universal Render Pipeline (URP) & Tipy

 Praktické rady pro nastavení URP, globální konfigurace, Volume efekty a optimalizaci renderování v Unity.

Co je Universal Render Pipeline (URP)?

- ▶  [Základní principy URP](#)
-

Default Volume Profile

- ▶  [Výchozí efekty Volume](#)
-

UniversalRenderPipelineGlobalSettings

- ▶  [Globální nastavení URP](#)
-

URP Render Pipeline Asset

- ▶  [Hlavní konfigurace URP](#)
-

URP Renderer Data

- ▶  [Specifická nastavení rendereru](#)



.NET CLI (Command Line Interface)

Pro práci je nutné mít nainstalovaný .NET SDK a .NET Runtime

Umístění balíčků a nástrojů

| Operační systém | Cesta k nástrojům | Zjištění cesty ke spustitelnému souboru |
|-----------------|----------------------------|---|
| Windows | %USERPROFILE%\dotnet\tools | where dotnet |
| macOS / Linux | ~/.dotnet/tools | which dotnet |

Správa nástrojů (.NET Tools)

| Akce | Globálně | Lokálně |
|------------------------|---|--|
| Seznam nástrojů | <code>dotnet tool list -g</code> | <code>dotnet tool list</code> |
| Instalace | <code>dotnet tool install -g</code> <code><název_balíčku></code> | <code>dotnet tool install</code> <code><název_balíčku></code> |
| Zastaralé | <code>dotnet tool list -g --outdated</code> | <code>dotnet tool list --outdated</code> |
| Aktualizace | <code>dotnet tool update -g</code> <code><název_balíčku></code> | <code>dotnet tool update</code> <code><název_balíčku></code> |
| Odinstalace | <code>dotnet tool uninstall -g</code> <code><název_balíčku></code> | <code>dotnet tool uninstall</code> <code><název_balíčku></code> |

Záloha a obnova globálních nástrojů

Záloha

- Získejte seznam nainstalovaných nástrojů:

```
dotnet tool list -g
```

2. Zaznamenejte názvy a verze pro pozdější obnovu.
3. Zálohujte adresář s nástroji:
 - Windows: `%USERPROFILE%\dotnet\tools`
 - macOS / Linux: `~/dotnet/tools`

Obnova

1. Zkopírujte zálohovaný adresář zpět na původní místo.
2. Restartujte terminál.
3. Ověřte instalaci:

```
dotnet tool list -g
```



NuGet Packages

🔧 Pro správu balíčků je potřeba mít nainstalovaný **NuGet CLI** nebo používat integrované nástroje v IDE.

Správa balíčků

- Způsoby správy balíčků

Globální složka balíčku

- Umístění globální složky

Doplnil jsem příklad a tabulky příkazů pro práci s NuGet balíčky v .NET. Vložte následující úsek do souboru `programming/packages/nugetPackage.md` na vhodné místo (např. pod sekci "Správa balíčků").

Příklady použití balíčku

```
dotnet add package SixLabors.ImageSharp.Drawing --version 2.1.7
```

Přehled základních příkazů

Správa balíčků

| Příkaz | Popis |
|---|--|
| <code>dotnet add package <název> --version <verze></code> | Přidá nebo aktualizuje konkrétní NuGet balíček na zadанou verzi v projektu. |
| <code>dotnet remove package <název></code> | Odebere balíček z projektu. |
| <code>dotnet restore</code> | Obnoví všechny závislosti projektu podle souboru <code>csproj</code> nebo <code>packages.config</code> . |

Kontrola a aktualizace

| Příkaz | Popis |
|--|--|
| <code>dotnet outdated</code> | Zobrazí seznam zastaralých NuGet balíčků v projektu a navrhne novější verze. |
| <code>dotnet outdated --upgrade</code> | Automaticky aktualizuje všechny zastaralé NuGet balíčky na nejnovější verze. |

Správa zdrojů a cache

| Příkaz | Popis |
|--|---|
| <code>dotnet nuget list source</code> | Zobrazí seznam zdrojů NuGet balíčků (repozitářů). |
| <code>dotnet nuget locals all --clear</code> | Vyčistí lokální cache NuGet balíčků (odstraní staré verze ze složky s balíčky). |

Správa npm balíčků

 Pro správu balíčků je potřeba mít nainstalovaný **Node.js** a **npm**.

Aktualizace balíčků

- ▶  Jak správně aktualizovat balíčky?
-

Globální balíčky

- ▶  Kde najdu globální balíčky?
-

Záloha globálních balíčků

- ▶  Jak zálohovat globální balíčky?
-

Obnova balíčků z offline zálohy

- ▶  Jak obnovit balíčky ze zálohy?
-

Aplikační balíčky

- ▶  conventional-changelog-cli

Python – Balíčky & Tipy

 Praktické rady pro správu Python balíčků, zálohování, offline instalaci a užitečné příkazy.

Co jsou Python balíčky?

- ▶  Základní principy
-

Záloha balíčků

- ▶  Jak zálohovat balíčky?
-

Instalace balíčků ze zálohy

- ▶  Offline instalace



Appcast Feed XML

🚀 Appcast je RSS feed ve formátu XML pro distribuci aktualizací aplikací pomocí technologie [Sparkle](#).

✳️ Hlavní komponenty

| 💡 Element | 💡 Popis |
|-----------|--|
| <rss> | Kořenový element, verze a namespace. |
| <channel> | Hlavní sekce feedu, metadata kanálu aktualizací. |
| <item> | Jednotlivý záznam pro jednu verzi aplikace. |

📦 Struktura feedu

- ▶ 🔎 [Příklad feedu](#)

🏷️ <rss> – Kořenový element

- ▶ 📖 [Detailly & příklad](#)

📡 <channel> – Metadata kanálu

- ▶ 📋 [Elementy & atributy](#)

📝 <item> – Jednotlivá aktualizace

- ▶ 📋 [Elementy & atributy](#)

✳️ Delta aktualizace

⚠️ Delta soubory obsahují pouze rozdíly mezi verzemi aplikace a šetří šířku pásma.

►  Příklad delta aktualizace



Přístup k projektům v XAMPP

Tento návod ukazuje, jak spravovat více projektů v XAMPP a pohodlně k nim přistupovat přes prohlížeč.

1. Spuštění Apache

- ▶ [Jak spustit Apache server?](#)
-

2. Vytvoření složek pro projekty

- ▶ [Jak organizovat projekty?](#)
-

3. Přístup k projektům přes prohlížeč

- ▶ [Jak zobrazit projekty v prohlížeči?](#)

Virtual Hosts v XAMPP

 Virtual Hosts umožňují přiřadit každému projektu vlastní doménu, např. `project1.local`, pro pohodlnější přístup a testování.

1. Konfigurace Apache

- ▶  Jak nastavit Virtual Hosts?
-

2. Úprava hosts souboru

- ▶  Jak přidat domény do hosts?
-

3. Restart Apache

- ▶  Jak restartovat Apache?
-

4. Přístup k projektům

- ▶  Jak přistupovat k projektům?