



Vývoj XAML (praktický cyklus)

1 Základní rozvržení

- Vytvoření složky UI a samotné Page (např. `MainPage.xaml`).
 - Vytvoření a napojení na navigaci (PageRegistry).
 - Definice hlavních **Gridů, Rows, Columns**.
 - Převzetí hlavních resourců (barvy, fonty) a překlíčování na **local resources**.
-

2 Doplnění obsahu

- Přidání:
 - TextBlocků (popisky, hlavičky).
 - Input polí (`TextBox`, `ComboBox`).
 - Tabulek (`DataGrid` nebo vlastní Grid).
 - Obrázků / ikon (SVG, FontIcon).
 - Všechny styly (barvy, fonty, velikosti) brány z **local resource dictionary**.
-

3 Příprava bindingu

- Vše, co se týká **multilanguage textů**, je bindováno.
- Veškerá **data (hodnoty, seznamy, stavové informace)**:
 - Jsou přiřazena z **konstruktoru codebehind Page**.

- Bindingy (např. `{x:Bind LabelText}`) vedou přímo do **codebehind**, pokud jde o hodnoty řízené View.
 - CodeBehind obsahuje:
 - Texty a data definovaná v konstruktoru Page.
 - Základní řízení UI (přebindování, logika viditelnosti, dynamické fonty).
 - Interaktivitu (tlačítka, eventy).
 - Resourcy mohou být přepínány dle potřeby (lokální jazyk, režim).
-

4 Čištění

- Úklid:
 - Odebrání nevyužitých názvů, properties.
 - Vyčištění XAML od zbytečných marginů, stylingu.
 - Codebehind slouží pouze jako řízení UI vrstvy, vše zbytečné odstraněno.
 - Přidání komentářů a popisků přímo v XAML.
-



Přechod do datové logiky

5 Vytvoření základního ViewModelu

- Vytvoření čistého **ViewModelu**:
 - Obsahuje pouze **data a business logiku**.
 - Neměsí žádné UI prvky ani eventy.
- Modely dle potřeby, XMLdata doplněna (nové properties, elementy).

6 Napojení dat

- Načtení dat z XML, databáze nebo služby:
 - Data se načítají do **modelů**.
 - Modely jsou předány do **ViewModelu**.
- ViewModel pouze poskytuje data pro PageView.

7 Binding do PageView

- VM je přiřazen jako **DataContext** PageView.
- Data z VM jsou bindována přímo přes XAML:
 - `{x:Bind VM.Property}` nebo `{Binding Property}`.
- Codebehind tím zůstává pouze jako **UI řadič** (přebindování, eventy, resource management).



Finální čištění a předání

8 Kontrola a refaktoring

- Kontrola XAML struktury.
- Refaktoring:
 - Odebrání dočasných proměnných.
 - Sloučení nadbytečných stylů, optimalizace resourců.
- Zajištění konzistence:

- Komentáře v angličtině.
 - SPC stylistika kódu.
 - Popis účelu VM přidán jako shrnutí na konec souboru.
-

9 Předání

- Připravená Page → Code Review.
 - Základní UX testing.
 - Přidání do dokumentace (kdo dělal, k čemu slouží, datum vytvoření).
-



Výsledný stav:

- Stránka rozdělena na Grid strukturu.
- Obsah plně bindovaný.
- Data řízená ViewModelem.
- View řídí pouze layout a přebindování.
- Kód je čistý, srozumitelný a dokumentovaný.



Strategie commitování v projektu tOrder



Obecné zásady:

- Každý commit řeší jeden konkrétní logický krok (ne míchat např. úpravu layoutu a opravu dat).
 - Komentáře commitů v angličtině.
 - Používat časování commitů podle pracovních bloků (ne až na konci dne).
 - Každá změna, která mění UI nebo data, by měla být commitována.
-



Návrh jednotlivých commit bodů:



1 Struktura a příprava projektu

- Initialize folder structure for new PageView
 - Add navigation hook for PageView
 - Setup base Grid layout for PageView
-



2 XAML vývoj (layout a UI)

- Define main layout grid and panels in PageView
- Add placeholder TextBlocks and basic input fields
- Setup local resource dictionary for colors and fonts

- Implement initial visual structure of PageView
-

3 Přidávání obsahu

- Add text fields and bind text properties from codebehind
 - Integrate icons and images into PageView
 - Apply local styles and fonts for text elements
 - Implement data tables / DataGrid in PageView
-

4 Data a binding

- Bind UI elements to codebehind properties
 - Prepare multilanguage bindings in PageView
 - Connect input controls to codebehind logic
 - (volitelné: pokud MVVM) Bind ViewModel data to PageView
-

5 Logika v CodeBehind

- Add UI-only event handlers in PageView codebehind
- Implement dynamic resizing and font scaling logic
- Manage resource switching in codebehind
- Implement button click events and UI state management

6 ViewModel a datové modely

- Create basic ViewModel structure for PageView
- Define models for XML data
- Load XML data and populate models
- Bind ViewModel data to UI

7 Čištění a optimalizace

- Refactor XAML layout and remove unused elements
- Clean up codebehind and remove temp variables
- Add comments and standardize code formatting
- Finalize ViewModel with documentation comment

8 Verze k odevzdání / revizi

- Prepare PageView and ViewModel for code review
- Final code cleanup before review
- Submit working PageView with basic data binding for review

 **Příklad dobře napsaného commitu:**

[Refactor] NamePageView: XAML layout, apply local resource styles and add multilanguage bindings.



Příklady stavových akcí:

- [Add] – přidání nové části.
- [Fix] – oprava chyby.
- [Remove] – odstranění zbytečných částí.
- [Update] – aktualizace stávajícího.
- [Optimize] – zlepšení výkonu nebo čitelnosti.
- [Test] – přidání testu nebo testovacího obsahu.



Doporučení commit frekvence:

- **Každé 1-2h** nebo při dokončení jednoho logického kroku.
- Před refaktoringem nebo větší změnou – vždy commit.
- Před předáním na code review – vždy čistý a připravený commit.
- Rozlišení průběžných, čistých a commitů verzí

🌟 SPC / tOrder – Kodovací konvence

🗺️ Dokument pro sjednocení vývoje, čitelnosti a kvality architektury v rámci projektu SPC **tOrder**

📅 Verze: 1.0 | 📁 Kontext: **SPC2.Tools**, **tOrder**, **MVVM**, **WinUI3**

📁 Struktura souboru a hlavičky

Každý **.cs** soubor musí začínat hlavičkou v jednotném formátu:

```
//=====
// $Workfile:: NázevSouboru.cs                $
// $Author::                                $
// $Revision::                                $
// $Date::                                    $
//=====
// Description: Stručný popis souboru
// Revision history:
// xx.mm.rrrr, Jméno, Popis změny
//=====
```

Každá třída je zřetelně označena:

```
//=====
// class NázevTřídy
//=====
```

Pro členění používej **#region** sekce:

```
//-----
#region Fields
//-----
```

🧠 Konvence pojmenování (Maďarská notace)



💡 Typ / Význam	Prefix	Příklad
----------------	--------	---------

<code>int</code> – čísla, indexy	<code>n</code>	<code>nCount, nRecord</code>
<code>bool</code> – logická hodnota	<code>b</code>	<code>bResult, bIsRemote</code>
<code>string</code>	<code>s</code>	<code>sName, sKey</code>
<code>object, var</code>	<code>o</code>	<code>oValue, oResult</code>
<code>float, double, decimal</code>	<code>f, d</code>	<code>dTimestamp, fScale</code>
<code>char</code>	<code>ch</code>	<code>chSeparator</code>
<code>DateTime</code>	<code>dt</code>	<code>dtStart, dtEnd</code>
<code>TimeSpan</code>	<code>ts</code>	<code>tsTimeout</code>
<code>List<T></code>	<code>lst</code>	<code>lstItems, lstPoints</code>
<code>Dictionary<K, V></code>	<code>dic</code>	<code>dicMappings</code>
<code>delegate, Func, Action</code>	<code>fn</code>	<code>fnCallback, m_fnDoX</code>
<code>StringBuilder</code>	<code>sb</code>	<code>sbFormat, sbText</code>
<code>Handle, HWND</code> apod.	<code>h</code>	<code>hWnd, hIcon</code>
<code>Pointer, IntPtr</code>	<code>p</code>	<code>pData, pBuffer</code>

Zvláštní pravidla:

- Privátní proměnné: `m_` prefix → `m_sName, m_nState`
 - Statické `readonly` proměnné: `s_` prefix → `s_fnGetDataRow`
 - Dvouznakové zkratky VELKÉ (`UI, ID`), tří a více znakové: PascalCase (`Xml, GpuMem`)
-


Formátování a styl

-  **Tabulátory** místo mezer (vynuceno `.editorconfig`)
 -  `using` direktivy jsou ve 3 skupinách:
 - `System.*`
 - `Microsoft.*`
 - Projektové (`SPC2.*`)
 - uvnitř abecedně
-

Konstrukční pravidla

Fáze	Popis
<code>Konstruktor</code>	Pouze vytvoření instance, nikdy neprovádí logiku ani validaci
<code>Init()</code>	Spouští se jednou, nastavuje výchozí stavy
<code>Load()</code> / <code>Reload()</code>	Opakovatelné, plní data, ověřuje validitu, připravuje pro binding

UI / Matematická pravidla

- Rozlišujeme **fyzický pixel** × **efektivní pixel**
 - UI layouty mají sudý počet řad → **ideálně 4**
 - Škálování: `4 → 5` (125%), `4 → 6` (150%)
 -  Mezery 3px NEPOVOLENY – minimálně 4px pro konzistenci
-



Filozofie vývoje

- Nejjednodušší funkční kód = **nejlepší kód**
 - Kvalitní vývojář:
 - ovládá své nástroje (IDE, Git, refactor, debug)
 - rozumí **architektuře** a **historickým principům**
 - respektuje vrstvy a odpovědnosti (MVVM, SRP, DI)
 - 🔍 Držíme se **Microsoft best practices** jako výchozího rámce
-




XML pravidla (LINQ to XML)

- Vždy používej explicitní operátory `(string)`, `(int?)`, `(DateTime?)` místo `.Value`
- Používej operátor `??` na defaultní hodnoty, které mohou chybět
- Zvaž vytvoření vlastních extension metod (`.AsInt()`, `.AsString()`)

Příklad:

```
var hex = (string)e.Attribute("Hex") ?? "#FF0000";  
var counter = (uint?)e.Attribute("Counter");
```

 Dokument slouží jako referenční i výukový rámec pro tým vývoje SPC/tOrder. V případě nejasností nebo návrhů na rozšíření kontaktujte správce architektury projektu.



Nastavení projektu (`.csproj`)

Všechny projekty **tOrder** musí obsahovat následující konfigurace pro zajištění jednotnosti, stability a čitelnosti kódu:

 Parametr	Popis a důvod použití
<code><AssemblyVersion></code>	Verze sestavení, používaná runtime (.NET assembly resolution)
<code><FileVersion></code>	Číslo verze souboru (zobrazeno ve vlastnostech sestavení)
<code><InformationalVersion></code>	Číslo verze určené pro uživatele (zobrazeno v aplikaci – About dialog)
<code><Nullable></code>	Povolení striktní kontroly nullable referencí (enable)
<code><LangVersion></code>	Použití nejnovější dostupné verze jazyka C# (latest)
<code><TreatWarningsAsErrors></code>	Veškerá varování kompilátoru jsou považována za chyby (true)
<code><OutputPath></code>	Sjednocená cesta výstupu build procesu (bin\\$(Configuration)\)
<code><Company></code>	Jméno společnosti (SPC Solutions)
<code><Product></code>	Název produktu (tOrder)
<code><Authors></code>	Autor projektu nebo odpovědná osoba (Alexandra Seligová)
<code><GenerateDocumentationFile></code>	Generování XML dokumentace pro IntelliSense a dokumentaci (true)

Důležité poznámky k jednotlivým nastavením:

- **Versioning (**AssemblyVersion**, **FileVersion**, **InformationalVersion**):**
 - Udržujte konzistentní verzování napříč projektem.
 - **InformationalVersion** využívejte pro zobrazení uživatelsky přátelského čísla verze.
- **Nullable Reference Types (**Nullable**):**

- Všechny nové projekty musí mít povolené nullable referenční typy, aby se minimalizovala rizika `NullReferenceException`.
- **Language Version (`LangVersion`):**
 - Zajišťuje využívání nejnovějších funkcí jazyka C#.
 - Usnadňuje dlouhodobý vývoj díky moderním funkcím jazyka.
- **Treat Warnings as Errors:**
 - Nutí vývojáře okamžitě řešit všechna varování, což dlouhodobě vede ke kvalitnějšímu kódu.
- **Unified Output Folder (`OutputPath`):**
 - Centralizuje build výsledky pro snadné nasazení a troubleshooting.
- **XML Documentation (`GenerateDocumentationFile`):**
 - Povinné pro API projekty.
 - Usnadňuje dokumentaci a poskytuje užitečné informace v IntelliSense.

tOrder – Přehled konfigurace projektu (.csproj)

Tento dokument slouží jako referenční shrnutí klíčových nastavení projektu `tOrder` v rámci SPC.

Základní konfigurace

Parametr	Hodnota	Popis
OutputType	<code>WinExe</code>	Výstupní typ aplikace, Windows Executable
TargetFramework	<code>net8.0-windows10.0.19041.0</code>	Cílová verze frameworku (.NET 8, Windows SDK)

TargetPlatformMinVersion	<code>10.0.17763.0</code>	Minimální podporovaná verze Windows
AllowUnsafeBlocks	<code>true</code>	Povolení „unsafe“ bloků kódu
DefineConstants	<code>DISABLE_XAML_GENERATED_MAIN</code>	Zakázání automaticky generovaného XAML vstupu
StartupObject	<code>tOrder.Program</code>	Hlavní vstupní bod programu
RootNamespace	<code>tOrder</code>	Hlavní namespace projektu
Nullable	<code>enable</code>	Povolení kontroly nullable referenčních typů

Packaging a distribuce

Parametr	Hodnota	Popis
ApplicationManifest	<code>app.manifest</code>	Aplikační manifest aplikace
WindowsAppSDKSelfContained	<code>true</code>	Self-contained Windows App SDK
UseWinUI	<code>true</code>	Používání UI frameworku WinUI
EnableMsixTooling	<code>true</code>	Povolení MSIX nástrojů pro snadné balení
WindowsPackageType	<code>None</code>	Bez explicitního balení do MSIX (Single Project MSIX tools)

Lokalizace

Parametr	Hodnota	Popis
DefaultLanguage	<code>en-US</code>	Výchozí jazyk aplikace

Platformy a publikování

Parametr	Hodnota	Popis
Platforms	x86;x64;ARM64	Podporované platformy
RuntimeIdentifiers	Dynamické (win-x86; win-x64; win-arm64)	Identifikátory runtime podle cílového frameworku
PublishProfile	win-\$(Platform).pubxml	Profil publikování podle cílové platformy

Dokumentace a warnings

Parametr	Hodnota	Popis
GenerateDocumentationFile	true	Generování XML dokumentace (IntelliSense, API docs)
NoWarn	1591;WMC1506;WMC0001	Ignorovaná varování

Obsah a assety

- **Assets:** obrázky, ikony, loga aplikace (SplashScreen, Logo, StoreLogo).
 - **MockData.xml:** testovací data, kopírování vždy (`CopyToOutputDirectory=Always`).
-

NuGet balíčky a závislosti

Balíček	Verze
CommunityToolkit (Common, Diagnostics, HighPerformance, MVVM)	8.4.0
CommunityToolkit.WinUI (Animations, Converters, Extensions)	8.2.250402
CommunityToolkit.WinUI.Controls (DataGrid, Markdown)	7.1.2
Microsoft.Extensions.Hosting	9.0.5

Microsoft.UI.Xaml	2.8.7
Microsoft.WindowsAppSDK	1.7.10 / 1.7.250513003
Microsoft.Windows.SDK.BuildTools	10.0.26100.4188
Microsoft.Xaml.Behaviors.WinUI.Managed	3.0.0

Nastavení publikování

Parametr	Debu g	Releas e
PublishReadyToRun	False	True
PublishTrimmed	False	True

Další konfigurace

- **Msix** Project Capability (povoleno Single-project MSIX tooling)
- **HasPackageAndPublishMenu** (zpřístupnění menu pro balení a publikaci v Solution Exploreru)

Návrh adaptivního layoutu (Scalable & Adaptive GUI)

Cílová rozlišení a aspect ratio:

Poměr stran

Příkladná rozlišení

4:3	1024×768, 1600×1200
16:9	1280×720 (HD), 1920×1080 (FHD), 3840×2160 (4K)
16:10	1280×800, 1920×1200, 2560×1600

Minimální požadavek:

- Nejnižší cílené rozlišení: **1024×768**
- Nejvyšší cílené rozlišení: **3840×2160 (4K)**

Možnosti přístupu k layoutům

1) Adaptivní/škálovatelná architektura

- Jeden layout, který se automaticky přizpůsobuje.
- Škáluje prvky v závislosti na aktuálním rozlišení a aspect ratiu.
- Obsahuje pravidla (VisualStateManager, Viewbox, Grid s * proportions).
- Velikosti jsou proporcionální a relativní, nikoliv pevně definované pixely.

Výhody:

- Nízká potřeba testovat všechny kombinace.
- Snadno přidáš nové rozlišení.
- Moderní UX/UI – plynulá adaptace.
- Vyšší dlouhodobá udržitelnost.

Nevýhody:

- Náročnější na počáteční návrh.

- Někdy se obtížně řeší detailní alignment (ikony, texty).

2 Skoková architektura (Discrete Breakpoints)

- Předem definované pevné layouty (breakpoints).
- Změna layoutu probíhá skokově podle dosažení breakpointu.
- Používají se fixní velikosti prvků pro každé rozlišení.

✓ Výhody:

- Snadnější pro přesné doladění každého jednotlivého layoutu.
- Lehčí implementace pro jednoduché případy.

✗ Nevýhody:

- Mnoho layoutů = hodně práce a testování.
- Horší udržitelnost – každá změna UI se provádí ve všech layoutech.
- Horší uživatelský zážitek (viditelné skoky).

1 2 3 4 Výpočet kombinací (pro skokový přístup)

Aspect Ratio	Počet uvedených rozlišení
4:3	2 (1024×768, 1600×1200)
16:9	3 (1280×720, 1920×1080, 4K)
16:10	3 (1280×800, 1920×1200, 2560×1600)

Celkem: 2 + 3 + 3 = 8 unikátních kombinací.

Pokud bys používala skokový přístup, musíš řešit minimálně **8 samostatných layoutů**.
Při adaptivním řešení máš jediný layout, který se sám přizpůsobuje.



Doporučený postup pro adaptivní přístup

♦ Krok 1 – Grid-based layout

- Využij „Grid“ s relativními definicemi (*, **Auto**).
- Nikdy nepoužívej fixní velikosti v pixelech (s výjimkou minimálních velikostí).

Příklad:

xml

ZkopírovatUpravit

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="2*" /> <!-- Relativní -->
    <RowDefinition Height="*" />
    <RowDefinition Height="Auto" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="3*" />
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>
</Grid>
```

♦ Krok 2 – VisualStateManager (Breakpoints)

- Jemné ladění UI pro specifické hraniční případy (např. jiný layout pro 4:3 vs. 16:9).
- Používáš pouze několik málo adaptivních breakpointů.

Příklad:

xml

ZkopírovatUpravit

```
<VisualStateManager.VisualStateGroups>
  <VisualStateGroup>
    <VisualState x:Name="WideLayout">
      <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="1200"/>
      </VisualState.StateTriggers>
    </VisualState>
    <VisualState x:Name="NarrowLayout">
      <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="0"/>
      </VisualState.StateTriggers>
    </VisualState>
  </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
```

♦ Krok 3 – Viewbox pro automatický scaling

- Užitečný pro grafiku, ikony, složité dashboardy.
- Automaticky škáluje celý obsah a udržuje proporcionalitu.

Příklad:

```
xml
ZkopírovatUpravit
<Viewbox Stretch="Uniform">
  <Grid Width="1280" Height="720">
    <!-- obsah zde -->
  </Grid>
</Viewbox>
```

♦ Krok 4 – Dynamické škálování fontů

- Velikost fontu škáluješ v závislosti na výšce nebo šířce okna.
- Využij binding na velikost View nebo řešení v codebehind.

Příklad bindingu velikosti fontu:


xml

ZkopírovatUpravit

```
<TextBlock Text="Sample"
            FontSize="{Binding ElementName=MainGrid, Path=ActualHeight,
Converter={StaticResource FontSizeConverter}}"/>
```



Shrnutí a porovnání architektur

Architektura	Adaptivní (Scalable & Adaptive) 	Skoková (Breakpoints) 
Počet layoutů	1 adaptivní	8 a více
Počet testů	Nízký	Vysoký
UX zážitek	Plynulý, moderní	Skokový, zastaralý
Udržitelnost	Vysoká	Nízká
Počáteční náročnost	Vyšší	Nižší



Doporučení pro tOrder

Pro moderní aplikaci typu **tOrder** jednoznačně doporučuji **adaptivní architekturu**, protože:

- Méně práce při dlouhodobé údržbě.
 - Příjemnější a modernější UX.
 - Menší počet layoutů ke kontrole.
 - Snadná adaptace pro nová rozlišení.
-

Optimální řešení tedy zahrnuje:

- Gridy s relativními jednotkami
- VisualStateManager s minimálními breakpointy
- Viewbox pro automatické škálování komplexního obsahu
- Dynamické škálování fontů

Dynamické vs. statické změny

Typ změny	Adaptivní přístup	Skokový přístup
Změna layoutu	Dynamicky (runtime)	Staticky (při spuštění)
Změna velikosti fontů	Dynamicky, globální scaling	Předpočítáno pro každý breakpoint
Přidání nového rozlišení	Snadné, dynamické	Práce navíc (nový breakpoint)
Změna spacingu mezi prvky	Automatické (relativní jednotky)	Manuální (fixní pro každý breakpoint)
Dynamická změna orientace	Snadno řešitelné	Nutnost nového breakpointu

Tabulka potřebných testů

Dynamický adaptivní layout

Test scénář	4:3 (1024×768, 1600×1200)	16:9 (1280×720 až 4K)	16:10 (1280×800, 2560×1600)
Scaling fontů	✓	✓	✓
Šířka prvků	✓	✓	✓
Výška prvků	✓	✓	✓
Alignment	✓	✓	✓

Prázdný prostor

(Jeden cyklus spuštění pokryje všechny scénáře dynamicky.)

Skokový layout (breakpointy)

Test scénář	4:3 (1024×768)	4:3 (1600×1200)	16:9 (1280×720)	16:9 (1920×1080)	16:9 (4K)	16:10 (1280×800)	16:10 (1920×1200)	16:10 (2560×1600)
Scaling fontů	!	!	!	!	!	!	!	!
Šířka prvků	!	!	!	!	!	!	!	!
Výška prvků	!	!	!	!	!	!	!	!
Alignment	!	!	!	!	!	!	!	!
Prázdný prostor	!	!	!	!	!	!	!	!

Debata o architektuře layoutu aplikace

tOrder

Úvod

Při řešení škálovatelného layoutu, který podporuje více rozlišení a poměrů stran (4:3, 16:9, 16:10), stojíme před dvěma hlavními možnostmi:

- **Adaptivní (jednotný) přístup**
- **Skokový (breakpointový) přístup**

? Otázky, které by se měly zohlednit

Obecné otázky

- Jak často se budou přidávat nová podporovaná rozlišení nebo aspect ratio?
- Jaké jsou požadavky na dlouhodobou udržitelnost layoutu?
- Jak důležitá je plynulost uživatelského zážitku?
- Jaký máme tým a jaký je jeho skill set pro implementaci?
- Jaké máme náklady na testování při jednotlivých přístupech?
- Jaký vliv mají zvolená řešení na výkon aplikace?

Důsledky a problematiky

- Jak řešit dynamické škálování fontů?
- Jak přistupovat k nevyužitému prostoru v různých aspect ratioch?
- Jakým způsobem definovat breakpointy u skokového přístupu?
- Jak složité bude předpočítávání rozměrů u skokového řešení?
- Jaký dopad má zvolené řešení na konzistenci UI?



Otázky k řešení – Adaptivní (dynamický) přístup

- Jak optimálně definovat globální scaling faktory?
- Jak řešit dynamické změny velikosti fontů a zachovat konzistenci napříč aplikací?
- Jak efektivně využít prázdný prostor, který vzniká u nestandardních poměrů?
- Jak zajistit přesnost alignmentu v adaptivním layoutu?
- Jak řešit performance u dynamického přepočítávání prvků runtime?

Otázky k řešení – Skokový (breakpointový) přístup


- Jak vybrat správné breakpointy pro jednotlivá rozlišení?
- Jak složité je implementovat a udržovat přesné hodnoty velikostí pixelů pro všechny kombinace?
- Jak automatizovat testování všech breakpointů?
- Jakým způsobem minimalizovat viditelnost skoků v layoutu při přechodu mezi breakpointy?
- Jak řešit změnu layoutu za běhu aplikace, pokud uživatel mění velikost okna?

Shrnutí a otevřená polemika

Adaptivní přístup poskytuje modernější UX a méně práce s dlouhodobou správou. Skokový přístup zase umožňuje přesné nastavení každého layoutu, ale za cenu zvýšených nákladů na údržbu a komplexitu testování.

Otázky k diskuzi:

- Jakou váhu má precizní kontrola každého layoutu vs. nižší náklady na údržbu?
- Je lepší uživatelská zkušenost s adaptivním layoutem dostatečně silným argumentem?
- Jak řešit efektivně problém nevyužitého prostoru u adaptivního layoutu?

 Dokument slouží jako výchozí materiál pro debatu týmu ohledně volby optimální architektury layoutu.

Responsive layout

Základní pravidla

- Neškálujeme fonty => základní rozlišení font 10

, celý layout má přirozeně reagovat na změnu velikosti do šířky a výšky

- Změna velikosti fontů a ikon se bude realizovat skokově, princip „Breaking points“
- Každé okno v tApps má mít minimální layout, pod který nepůjdeme (tApps nejsou mířené na mobily)
 - Minimalisticky požadavek při poměru 4:3 je 960 x 720 – základ pro vývoj // Správně je základ viz návrh vizuálu **1024x768**
 - Při poměru 16:9 bereme v potaz 1280 x 720

Pozn.: Veškerá škálovatelnost fontu by se měla řešit pomocí globálních stylu

Např.:

- fonty vč. základních velikostí
- velikosti ikon ovládacích prvků
- velikosti tlačítek

Atd

Oblast	(Responsive Layout s Breakpoints)	Dynamické řešení (Globální škálování, Vektorové)
Fonty	Fixní základní velikost (např. 10), škálují se skokově pomocí breakpoints	Vektorově škálované globálně, přirozeně zachován poměr (scaling)
Škálování fontů	Řešeno pomocí globálních stylů, ale změna pouze při dosažení breakpoint	Neřešíš globální fonty – vše se škáluje automaticky jednotně

Ikonky	Škálování ikon fixní / skokové (globální styl při breakpointu)	Ikony vektorové – přirozené škálování se zachováním poměru
Velikosti tlačítek	Přepínané při breakpointu, velikosti řešeny ve stylech	Dynamické škálování podle layoutu, včetně paddingu/margin
Layout - šířka/výška	Přirozená reakce layoutu na změnu šířky/výšky okna	Řešíš přirozené škálování – layout reaguje plynule
Poměr stran	Definovány minimální rozměry (960x720, 1024x768, 1280x720)	V dynamice počítáš s poměrem stran, ale není fixován – adaptivní
Breaking points	ANO – fixní body, kde dochází ke změně velikostí fontů, ikon, atd.	NE – žádné breaking points, vše se škáluje kontinuálně
Globální styly (fonts, icons)	Vyžaduje ruční definici stylů pro různé stavy/breakpointy	Nepotřebuješ – používáš jednotné vektorové škálování
Dynamika prvků kolem textu	Nejasné, spíše závislé na layoutu	Řešíš dynamicky – padding, margin a spacing škáluješ automaticky
Vhodnost pro 4K a vysoká DPI	Omezené, pouze pokud nastavíš další breaking points	Vektorové řešení bez problémů s 4K/HiDPI, zachování čitelnosti
Přístupnost (Accessibility)	Komplikované – fixní fonty vs. zvětšené fonty mohou působit nekonzistentně	Škálování přirozené, všechno ve stejném poměru – vhodnější