# Návrh architektúry komponentového balíka

Balík dvoch nezávislých komponentov (AdvancedWinUI.Logger a AdvancedWinUI.DataGrid) bude navrhnutý s dôrazom na princípy **SOLID**, **Dependency Injection** a **Clean Architecture**. SOLID zásady sú „päť prikázaní“ kvalitného softvéru a tvoria základ modulárneho, udržateľného kódu[[1]](https://learn.microsoft.com/en-us/shows/visual-studio-live-2017/w19#:~:text=The%20SOLID%20principles%20are%20the,will%20even%20check%20out%20some). Clean Architecture oddeľuje doménovú logiku (core) od infraštruktúry a UI, čo zaisťuje, že jadro zostáva nezávislé a testovateľné. Napríklad v Core definujeme rozhrania (abstrakcie) a implementácie umiestňujeme do vonkajších vrstiev, ktoré ich poskytujú cez DI[[2]](https://devblogs.microsoft.com/ise/next-level-clean-architecture-boilerplate/#:~:text=To%20achieve%20a%20clean%20architecture%2C,at%20runtime%20using%20dependency%20injection). Pri návrhu služieb dbáme na to, aby boli malé, bez globálneho stavu a ľahko testovateľné. Zabráňme priamemu vytváraniu inštancií v kóde – všetky závislosti poskytne DI kontajner[[3]](https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection-guidelines#:~:text=When%20designing%20services%20for%20dependency,injection).

* **SOLID princípy**: Každá trieda má jednu zodpovednosť, nové funkcionality pridávame rozšírením existujúcich, nie modifikáciou (OCP), triedy možno nahradiť bez zmeny klienta (LSP), rozhrania sú špecifické pre potrebu (ISP) a implementácie závisí od abstrakcie, nie naopak (DIP)[[1]](https://learn.microsoft.com/en-us/shows/visual-studio-live-2017/w19#:~:text=The%20SOLID%20principles%20are%20the,will%20even%20check%20out%20some)[[2]](https://devblogs.microsoft.com/ise/next-level-clean-architecture-boilerplate/#:~:text=To%20achieve%20a%20clean%20architecture%2C,at%20runtime%20using%20dependency%20injection).
* **DI a testovateľnosť**: Vyhýbame sa statickým globálnym stavom; všetky služby registrujeme v IServiceCollection a vkladajú sa cez konštruktory. Akákoľvek trieda s mnohými závislosťami by možno porušovala SRP a mala by byť rozdeliť na viacero tried[[3]](https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection-guidelines#:~:text=When%20designing%20services%20for%20dependency,injection).
* **Functional-OOP mix**: V C# (a .NET 8) môžeme kombinovať OOP triedy s funkcionálnymi prístupmi (immutable struktúry, statické utility). Kľúčom je zachovať neporušenosť údajov, čisté funkcie tam, kde nedochádza k vedľajším efektom, a zároveň využívať triedne abstrakcie pre komplexné logiky.

## Logger Komponent (AdvancedWinUI.Logger)

**Zameranie:** poskytne modulárny logger na báze Microsoft.Extensions.Logging. Používame abstrakciu ILogger<T> a ILoggerFactory, ale namiesto štandardných providerov implementujeme vlastný log do súboru s rotáciou.

* **Rozhranie a implementácia**: Definujeme rozhranie IAdvancedLogger (metódy Info(), Error(), atď.) v jadre komponentu. Implementácia AdvancedLoggerProvider : ILoggerProvider v infraštruktúrnej vrstve vytvára konkrétne ILogger inštancie (napr. trieda FileLogger), ktoré zapisujú logy do súboru. Podľa vzoru Clean Arch definujeme abstrakciu v Core a plnenie v infraštruktúre[[2]](https://devblogs.microsoft.com/ise/next-level-clean-architecture-boilerplate/#:~:text=To%20achieve%20a%20clean%20architecture%2C,at%20runtime%20using%20dependency%20injection).
* **Konfigurácia**: Pomocou IOptionsMonitor<LoggerOptions> umožníme dynamické nastavenie cesty k log súborom, maximálnej veľkosti súboru a počtu uchovávaných rotovaných súborov. Následne v ILoggerProvider využijeme tieto nastavenia pri vytváraní loggerov. Rovnako ako v dokumentácii Microsoft, aj tu voláme v ILogger.Log najprv IsEnabled(logLevel) pre optimalizáciu výkonu[[4]](https://learn.microsoft.com/en-us/dotnet/core/extensions/custom-logging-provider#:~:text=It%27s%20a%20good%20practice%20to,very%20fast%20in%20most%20implementations).
* **LoggerExtensions**: Pre pohodlné volanie budeme doplňovať statické rozšírenia, napr. logger.Info("message"), logger.Error(ex, "failed"), ktoré pod kapotou volajú logger.LogInformation atď. Vďaka jednej doménovej knižnici s rozhraniami ostane API čisté.
* **Rotácia a súbory**: Implementujeme rotáciu podľa veľkosti – keď súbor prekročí stanovený limit, premenujeme ho a začneme nový. Konfigurácia názvu súboru a cesty bude viazaná na LoggerOptions. Môžeme využiť podobný prístup ako Azure App Services File Logger (konfigurovanie FileSizeLimit, RetainedFileCount)[[5]](https://learn.microsoft.com/en-us/dotnet/core/extensions/logging-providers#:~:text=%7B%20options.FileName%20%3D%20%22azure,).
* **Bez UI**: Tento komponent neobsahuje žiadny UI. Využíva sa takto: knižnica je injektovaná do aplikácie, kde pomocou DI získame IAdvancedLogger a používame rozšírenia. V tíme by README uvádzal príklad registrácie: services.AddAdvancedWinUiLogger(config => { … });.

## DataGrid Komponent (AdvancedWinUI.DataGrid)

**Zameranie:** vysoko výkonná tabuľka pre WinUI3 a „headless“ (non-UI) scenáre. Kľúčové princípy sú oddelenie dátovej logiky od UI, použiteľnosť v rôznych kontextoch a množstvo features.

* **Architektúra**: Komponent bude rozdelený na dve časti: **Core** knižnica s logikou správy údajov a validácií (bez závislosti na WinUI), a **UI** projekt, ktorý obsahuje WinUI elementy a XAML. Core definuje rozhrania a modely (tabuľky, riadky, validátory). UI časť má UserControl alebo Page s DataGrid (napr. z WinUI Community Toolkit) a viaže ho na Core cez ViewModel. Takto držíme čisté oddelenie logiky od prezentácie.
* **Výkon a Virtualizácia**: Pre spracovanie miliónov riadkov je **kľúčová virtualizácia** – načítavanie dát len na požiadanie. Napríklad Syncfusion uvádza, že s povolenou virtualizáciou dokáže načítať milióny záznamov okamžite[[6]](https://www.syncfusion.com/winui-controls/datagrid#:~:text=Data%20virtualization). My môžeme použiť VirtualizingStackPanel v DataGrid alebo vlastný mechanizmus načítavania stránok z dátového zdroja. Podkladom môže byť napr. IEnumerable<T> alebo DataTable s virtuálnou pamäťou (používa sa LoadMoreItems pattern). Tým sa vyhneme načítavaniu celého datasetu do pamäte.
* **Validácia**: Komponent bude podporovať **bunke ani celú tabuľku**. Podobne ako vo WPF DataGrid, aj tu umožníme nastaviť validátory na každý stĺpec (napr. podľa dátových anotácií alebo funkčných delegátov), ktoré sa vyhodnocujú pri editácii bunky (real-time) alebo pri potvrdení celého riadku (batch). WPF DataGrid podporuje validáciu na úrovni buniek aj riadkov[[7]](https://learn.microsoft.com/en-us/dotnet/desktop/wpf/controls/how-to-implement-validation-with-the-datagrid-control#:~:text=The%20DataGrid%20%20control%20enables,the%20%204%20control%20provides); náš dizajn umožní pridať ValidationRule pre celý objekt (celý riadok) alebo pre jednotlivé vlastnosti. Chyby sa budú hlásiť pomocou notifikácií na modeli (napr. implementácia INotifyDataErrorInfo), aby ich UI mohla zobraziť (červené orámovanie bunky, vypisanie custom validacnych chybovych hlasok do special stlpca ValidAlerts). Tým poskytujeme okamžitú spätnú väzbu.
* **Smart správa riadkov**: Komponent zabezpečí, že tabuľka má vždy aspoň určitý počet riadkov (napríklad minimálne 1). Po úprave údajov skontrolujeme, či je posledný riadok vyplnený; ak áno, automaticky vytvoríme nový prázdny riadok na koniec (súvisí s pridaním nových záznamov). Tým používateľ neustále vidí na konci riadok na zadávanie nových dát. Ak používateľ vymaže existujúce riadky, minimalizujeme tabuľku na nastavený počet (napr. 1).
* **Farby a štýly**: Poskytneme možnosti vlastnejšej grafickej úpravy. Napríklad cez vlastnosti alebo ResourceDictionary umožníme definovať farby pozadia, farby chýb a iné štýly. Ako uvádza Syncfusion, vzhľad DataGridu a jeho prvkov (bunky, riadky, hlavičky) sa dá prispôsobiť množstvom dostupných vlastností a štýlov[[8]](https://www.syncfusion.com/winui-controls/datagrid#:~:text=UI%20customization). Navrhneme teda tematické štýly (light/dark) a možnosť definovať vlastné XAML štýly pre rôzne typy buniek (číselné, dátumové, textové).
* **Drag & Drop**: Podpora presúvania riadkov (drag&drop) uľahčí prácu používateľom. Riešenie: v UI nastavíme AllowRowDragDrop=true a AllowDrop=true (podľa Syncfusion)[[9]](https://help.syncfusion.com/winui/datagrid/row-drag-and-drop#:~:text=WinUI%20DataGrid%20%20allows%20you,SfTreeView%20%20and%20%2016). Následne obslúžime udalosti typu DragStarting, DragOver, Dropping, Dropped na pretvorenie kolekcie dát podľa nového poradia. Princíp: drag&drop sa deje na úrovni UI, ale manipulácia s dátami je vykonaná cez metódy managera (nepíše sa v code-behind veľa logiky).
* **Výber a zmena veľkosti**: Tabuľka umožní viacnásobný výber riadkov/buniek (SelectionMode=Extended alebo Multiple), aby bol kompatibilný s bežným užívateľským očakávaním (podpora Ctrl+klik, Shift+šipka). Umožníme tiež zmenu veľkosti stĺpcov (ručné pretiahnutie alebo automatické prispôsobenie). Tieto vlastnosti budú nakonfigurovateľné parametrami control-u.
* **Import/Export**: Komponent poskytne API na načítanie a uloženie dát z/do rôznych formátov. Špecificky uvažujeme import/export z Dictionary<string, object> (kamenné dáta) a DataTable (štandardný .NET typ pre tabuľky). Používateľ si môže poslať dáta napr. vo forme List<Dictionary<string, object>> do gridu, alebo celý DataTable. Export by poskytoval rovnaké typy späť. Pre obyčajné scenáre by sme mohli tiež doplniť CSV export/import, ale minimom je spomínaný export do generických formátov.
* **Reusabilné API**: Dôležitá vlastnosť je použiteľnosť cez API aj bez UI. Preto bude mať komponent tzv. *manager* (napr. AdvancedDataGridManager), ktorý umožní manipuláciu s dátami tabuľky skriptovo. Napr. v konzolovej či serverovej aplikácii by sme mohli s AdvancedDataGridManager načítať DataTable, spustiť validáciu, a exportovať späť, bez potreby načítavať XAML. Zároveň UI verzia túto logiku využíva cez ViewModel, ktorý prepája DataGrid s managerom.

## Adresárová štruktúra a súbory

Navrhneme modulárnu štruktúru na úrovni riešenia (solution) s osobitnými projektmi pre každý komponent a testy:

* **/RpaWinUiComponentsPackage/AdvancedWinUILogger/** – Class Library projekt obsahujúci IAdvancedLogger, LoggerOptions, FileLoggerProvider, FileLogger a LoggerExtensions.
* **/RpaWinUiComponentsPackage/AdvancedWinUIDataGrid/** – Class Library projekt obsahujúci jadrovú logiku DataGrid (DataGridModel, DataGridManager, validátory), ale bez UI závislostí.
* **/RpaWinUiComponentsPackage/AdvancedWinUIDataGrid/UI/** – WinUI3 projekt (alebo WinUI komponent) obsahujúci XAML/C# kód pre vizuálny DataGrid, viazaný na core knižnicu cez ViewModel.
* **/RpaWinUiComponentsPackage/Common/** *(voliteľné)* – Pre zdieľané typy a abstrakcie, ak nejaké existujú (napr. spoločné rozhrania, error logging).
* **/tests/AdvancedWinUI.Logger.Tests/** – Unit testy pre logging (kontrola rotácie, formát logu, chybové stavy).
* **/tests/AdvancedWinUI.DataGrid.Tests/** – Unit testy pre dátový grid (testy validácie, správania pri pridávaní/vymazávaní riadkov, import/export).

Každý projekt má vlastný .csproj súbor. V koreňovom adresári bude súbor RpaWinUiComponentsPackage.sln. V dokumentácii (README, CONTRIBUTING) vysvetlíme, ktoré projekty sú k čomu určené. Túto štruktúru možno porovnať s bežným Clean Architecture template, ktorý má projekty pre Domain/Core, Application, Infrastructure a UI[[10]](https://github.com/ardalis/CleanArchitecture#:~:text=The%20goal%20of%20this%20repository,more%20about%20these%20topics%20here).

## Rozhrania a manažérsky pattern

Pre každú funkčnosť definujeme **rozhrania (contract)** a manažérske triedy (implementácie). Napríklad:

* **Logger**: Rozhranie IAdvancedLogger (s metódami Info, Warn, Error atď.). Manager/trieda AdvancedLoggerManager (alebo jednoducho FileLogger) implementuje toto rozhranie. V kóde aplikácie závisíme na IAdvancedLogger, nie na konkrétnej triede (DIP). Inštancia sa injektuje cez DI.
* **DataGrid**: Rozhranie IDataGridService alebo IAdvancedDataGrid, ktoré môže mať metódy LoadData(DataTable dt), ExportData(), ValidateAll(), AddEmptyRow(), atď. Manager/trieda AdvancedDataGridManager túto logiku implementuje. UI (WinUI Page/ViewModel) používa toto rozhranie na operácie s tabuľkou.

Tento prístup je v súlade s Clean Arch: vnútorné moduly definujú abstrakcie, vonkajšie poskytujú implementácie[[2]](https://devblogs.microsoft.com/ise/next-level-clean-architecture-boilerplate/#:~:text=To%20achieve%20a%20clean%20architecture%2C,at%20runtime%20using%20dependency%20injection). Managerová trieda vystupuje ako **Facade**, skrýva detaily implementácie a ponúka jedno prístupové miesto (aj keď vo vnútri používa viaceré menšie triedy). V kóde teda developer zavolá napr. dataGridManager.Load(...) a nemusí riešiť, ako sa interné časti skladajú. Toto uľahčuje testovanie (môžeme injektovať falošný IAdvancedDataGrid do ViewModel) a zvyšuje udržateľnosť kódu. Podľa odporúčaní Microsoft sa služby, ktoré injektujeme, majú správať ako malé, izolovateľné jednotky[[3]](https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection-guidelines#:~:text=When%20designing%20services%20for%20dependency,injection).

## Čisté API a používanie

Každý komponent vystaví jednoduché API, ktoré vyžaduje iba jeden using. Napríklad:

using AdvancedWinUILogger;  
// ...  
IAdvancedLogger logger = serviceProvider.GetRequiredService<IAdvancedLogger>();  
logger.Info("Aplikácia sa spustila.");

Rovnako pre DataGrid:

using AdvancedWinUIDataGrid;  
// ...  
IDataGridService grid = serviceProvider.GetRequiredService<IDataGridService>();  
grid.LoadData(myDataTable);

Čisté API znamená, že všetky dôležité typy sú v jednom namespace (napr. AdvancedWinUILogger, AdvancedWinUIDataGrid) a developer nepotrebuje mnoho using direktív. Pre registráciu v DI môžeme pridať extension metódu: services.AddAdvancedWinUiLogger(opts => { ... });, ktorá zaregistruje IAdvancedLogger a ILoggerProvider. Podobne services.AddAdvancedWinUiDataGrid(). Tým udržíme kód, ktorý komponent používa, čo najjednoduchší.

## Validácia, správa riadkov a UI integrácia

V časti **DataGrid**:

* **Validácia**: Pri editácii bunky spustíme ValidateCellValue, ktorá využije buď dátové anotácie modelu, alebo vlastné pravidlá. Príklad: ak používame INotifyDataErrorInfo, UI (WinUI DataGrid) môže automaticky zobraziť červené orámovanie. Celková validácia (ValidateAll) prejde všetky riadky a zhromaždí chyby.
* **Smart riadky**: V AdvancedDataGridManager budeme sledovať stav tabuľky. Ak je posledný riadok plný (všetky stĺpce majú hodnoty), vytvoríme nový prázdny. Ak niektoré užívateľ vymaže tak, že ich počet klesne pod minimálny, pridáme chýbajúce. Toto logiku môže vyvolať UI (napr. pri každom ukončení úpravy bunky) a zabezpečíme, že používateľ nikdy nemá „prázdny“ grid bez šance pridávať ďalej riadky.
* **UI integrácia**: Použijeme MVVM vzor. View (XAML s DataGridom) bude viazaná na ViewModel, ktorý intern zavolá metódy IDataGridService. Všetka business logika (validácia, správa dát) bude mimo UI. UI projekt/vrstva bude obsahovať *len* vizualizáciu (UserControls, štýly, eventhandler-y na úrovni UI, ktoré následne volajú Core služby). Tým sa vyhneme súborom „boh-god“, ktoré by robili všetko v jednom. Každá trieda či komponent má jasnú zodpovednosť (SRP) a prekopírovanie logiky medzi zložkami je minimalizované.

## README a tímová spolupráca

V dokumentácii balíka (README.md) definujeme postupy pre tím:

* **Popis komponentov**: Účel AdvancedWinUILogger a AdvancedWinUIDataGrid, hlavné vlastnosti (logovanie do súboru s rotáciou, 10M riadkov s validáciou atď.).
* **Príklad použitia**: Ukážkové kódy (snippet) registrácie služieb v DI a základného použitia (napr. prihlásenie logu a načítanie DataGrid).
* **Konfigurácia**: Vysvetlíme voľby konfigurácie LoggerOptions (napr. LogPath, MaxFileSize, MinRows, témy pre farby), ako ich nastaviť v kode alebo v appsettings.json.
* **Vývojové štandardy**: Spolu ujasníme kódové konvencie (názvy parametrov, vlastností), závislosti, verzovanie API (semantic versioning). Napríklad dodržiavať MENDA principy: každé API je spätne kompatibilné, update MAJOR ak dôjde k breaking change.
* **Použité knižnice**: Uvedieme, že používame Microsoft.Extensions.Logging.Abstractions (pre logging), System.Data pre DataTable, a že používame .NET 8, WinUI 3.
* **Pravidlá logovania**: Definujeme, čo sa loguje na ktorú úroveň (napr. Info pre štatistické udalosti, Warning pre nestabilné ale nefatálne, Error pre výnimky). Ukladanie citlivých údajov zakážeme. Tím dodržiava, že každá výnimka sa zaloguje s Error, kritické udalosti s vyššou úrovňou.
* **Kontribúcia a testovanie**: Odporúčame používať jednotkové testy (xUnit/NUnit) pri pridávaní funkcionalít. Názvoslovie v GIT e.g. branch pod označením issue # atď.

V README sa skrátka nastaví “tone” enterprise environmentu: code review, CI pipeline, jednotkové testy, konzistentné formátovanie. Taktiež sem pridáme linky na relevantné dokumenty (napr. link na Microsoft Logging docs, SOLID princípy atď.), aby mal tím referencie.

## Zhrnutie

Navrhovaný balík je modulárny, riadi sa princípmi **Clean Architecture** a **SOLID**, využíva DI a testovateľné služby[[2]](https://devblogs.microsoft.com/ise/next-level-clean-architecture-boilerplate/#:~:text=To%20achieve%20a%20clean%20architecture%2C,at%20runtime%20using%20dependency%20injection)[[3]](https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection-guidelines#:~:text=When%20designing%20services%20for%20dependency,injection). Komponenty sú oddelené logikou/UI, navzájom nezávislé, s čitateľným API („jedno using“). AdvancedLogger sa integruje ako ILoggerProvider so špeciálnymi nastaveniami súborov a rotácie, využíva ILogger rozhranie a vlastné extensions (napr. logger.Info())[[4]](https://learn.microsoft.com/en-us/dotnet/core/extensions/custom-logging-provider#:~:text=It%27s%20a%20good%20practice%20to,very%20fast%20in%20most%20implementations). AdvancedDataGrid ponúka efektívne spracovanie veľkých objemov dát (virtualizácia[[6]](https://www.syncfusion.com/winui-controls/datagrid#:~:text=Data%20virtualization)), komplexnú validáciu (buniek aj riadkov[[7]](https://learn.microsoft.com/en-us/dotnet/desktop/wpf/controls/how-to-implement-validation-with-the-datagrid-control#:~:text=The%20DataGrid%20%20control%20enables,the%20%204%20control%20provides)), a užívateľsky prívetivé funkcie (drag&drop[[9]](https://help.syncfusion.com/winui/datagrid/row-drag-and-drop#:~:text=WinUI%20DataGrid%20%20allows%20you,SfTreeView%20%20and%20%2016), prispôsobiteľné štýly[[8]](https://www.syncfusion.com/winui-controls/datagrid#:~:text=UI%20customization)). Štruktúra projektu a definície rozhraní zabezpečia, že každý komponent je robustný, rozšíriteľný a pripravený pre enterprise nasadenie.

**Zdroj:** Návrh vychádza z moderných odporúčaní tvorby softvéru, napr. principy Clean Architecture a SOLID[[2]](https://devblogs.microsoft.com/ise/next-level-clean-architecture-boilerplate/#:~:text=To%20achieve%20a%20clean%20architecture%2C,at%20runtime%20using%20dependency%20injection)[[1]](https://learn.microsoft.com/en-us/shows/visual-studio-live-2017/w19#:~:text=The%20SOLID%20principles%20are%20the,will%20even%20check%20out%20some), oficiálnej dokumentácie .NET Logging a DI[[4]](https://learn.microsoft.com/en-us/dotnet/core/extensions/custom-logging-provider#:~:text=It%27s%20a%20good%20practice%20to,very%20fast%20in%20most%20implementations)[[3]](https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection-guidelines#:~:text=When%20designing%20services%20for%20dependency,injection) a príkladov špičkových DataGrid knižníc[[6]](https://www.syncfusion.com/winui-controls/datagrid#:~:text=Data%20virtualization)[[9]](https://help.syncfusion.com/winui/datagrid/row-drag-and-drop#:~:text=WinUI%20DataGrid%20%20allows%20you,SfTreeView%20%20and%20%2016)[[7]](https://learn.microsoft.com/en-us/dotnet/desktop/wpf/controls/how-to-implement-validation-with-the-datagrid-control#:~:text=The%20DataGrid%20%20control%20enables,the%20%204%20control%20provides), aby komponenty boli spoľahlivé a vysoko výkonné.

[[1]](https://learn.microsoft.com/en-us/shows/visual-studio-live-2017/w19" \l ":~:text=The%20SOLID%20principles%20are%20the,will%20even%20check%20out%20some) SOLID – The Five Commandments of Good Software | Microsoft Learn

<https://learn.microsoft.com/en-us/shows/visual-studio-live-2017/w19>

[[2]](https://devblogs.microsoft.com/ise/next-level-clean-architecture-boilerplate/#:~:text=To%20achieve%20a%20clean%20architecture%2C,at%20runtime%20using%20dependency%20injection) Next-Level Boilerplate: An Inside Look Into Our .Net Clean Architecture Repo - ISE Developer Blog

<https://devblogs.microsoft.com/ise/next-level-clean-architecture-boilerplate/>

[[3]](https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection-guidelines#:~:text=When%20designing%20services%20for%20dependency,injection) Dependency injection guidelines - .NET | Microsoft Learn

<https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection-guidelines>

[[4]](https://learn.microsoft.com/en-us/dotnet/core/extensions/custom-logging-provider#:~:text=It%27s%20a%20good%20practice%20to,very%20fast%20in%20most%20implementations) Implement a custom logging provider - .NET | Microsoft Learn

<https://learn.microsoft.com/en-us/dotnet/core/extensions/custom-logging-provider>

[[5]](https://learn.microsoft.com/en-us/dotnet/core/extensions/logging-providers#:~:text=%7B%20options.FileName%20%3D%20%22azure,) Logging providers - .NET | Microsoft Learn

<https://learn.microsoft.com/en-us/dotnet/core/extensions/logging-providers>

[[6]](https://www.syncfusion.com/winui-controls/datagrid#:~:text=Data%20virtualization) [[8]](https://www.syncfusion.com/winui-controls/datagrid#:~:text=UI%20customization) WinUI DataGrid Control | Lightning Fast Data Binding | Syncfusion

<https://www.syncfusion.com/winui-controls/datagrid>

[[7]](https://learn.microsoft.com/en-us/dotnet/desktop/wpf/controls/how-to-implement-validation-with-the-datagrid-control#:~:text=The%20DataGrid%20%20control%20enables,the%20%204%20control%20provides) How to: Implement Validation with the DataGrid Control - WPF | Microsoft Learn

<https://learn.microsoft.com/en-us/dotnet/desktop/wpf/controls/how-to-implement-validation-with-the-datagrid-control>

[[9]](https://help.syncfusion.com/winui/datagrid/row-drag-and-drop#:~:text=WinUI%20DataGrid%20%20allows%20you,SfTreeView%20%20and%20%2016) Row drag and drop in WinUI DataGrid control | Syncfusion®

<https://help.syncfusion.com/winui/datagrid/row-drag-and-drop>

[[10]](https://github.com/ardalis/CleanArchitecture#:~:text=The%20goal%20of%20this%20repository,more%20about%20these%20topics%20here) GitHub - ardalis/CleanArchitecture: Clean Architecture Solution Template: A proven Clean Architecture Template for ASP.NET Core 9

<https://github.com/ardalis/CleanArchitecture>