

**CacheProvider (DataHandlingLibrary)**  
**Mål** | **Hur koden uppfyller det**  
**Stoppa överbelastning av API:t – återanvänd data som sällan ändras under några minuter.**  
En privat instans av MemoryCache (från System.Runtime.Caching) startas med ett eget namn, så varje konsument får ett isolerat cacheminne.

**Håll biblioteket UI-agnostiskt – cachelogiken finns i ett klassbibliotek, inte i WPF/Xamarin-sidorna.**  
CacheProvider implementerar ICacheProvider, vilket innebär att gränssnittet kan prata med ett interface och aldrig behöver oroa sig för implementationen. Om du senare byter till Redis eller LiteDB, byter du bara klass utan att röra sidorna.

**Undvik inaktuella objekt – varje post ska självförfalla.**  
Konstruktorn tar emot dataLifeTimeMinut (standard 10). AddDataToCache() lagrar posten med en absolut utgångstid av DateTimeOffset.Now.AddMinutes(\_dataLifeTimeMinut).

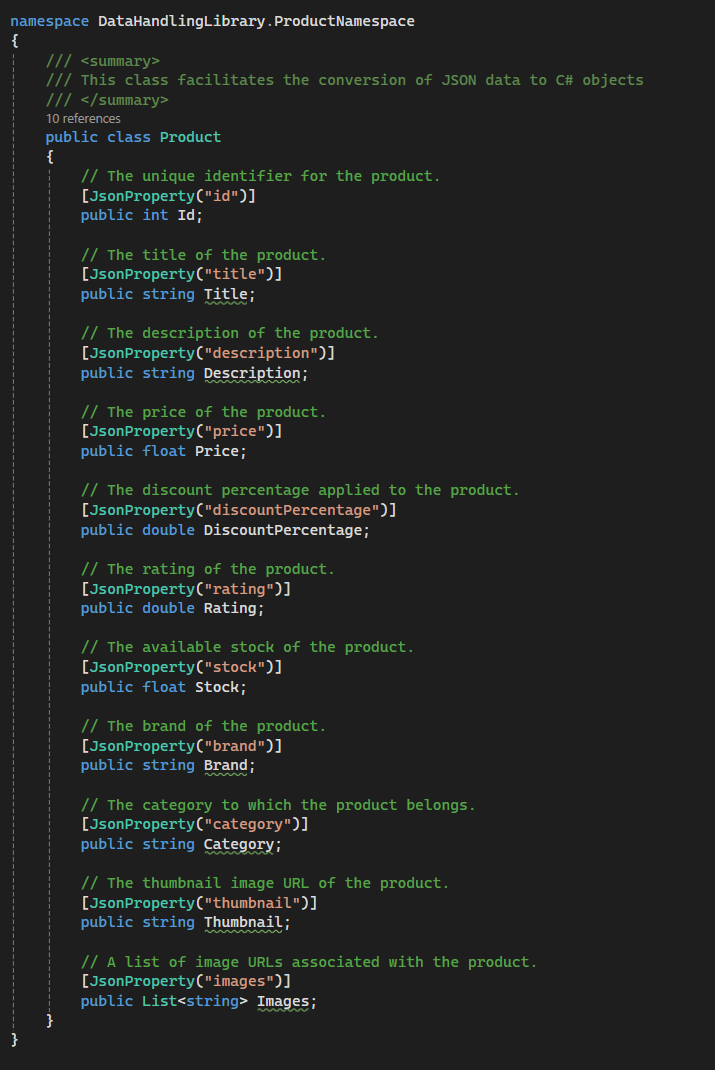
**Defensiv programmering**

* Negativa livslängder återställs till standardvärdet.
* CheckStringValue() (utelämnad i skärmdumpen) kastar ett undantag om cache-namnet eller nyckeln är null/tomma, för att skydda mot tysta fel.

**Så här fungerar det vid körning**  
En sida (t.ex. **ProductsReview**) begär data från Responses.GetProducts().

Responses anropar CacheProvider.AddDataToCache(); om den returnerar **false** finns objektet redan och läses tillbaka med GetCachedData<T>() (nästa metod i filen).

Endast när nyckeln saknas eller har förfallit, anropas ResponseHandler.SendGetRequestAsync() för att träffa DummyJSON-endpointen.  
Detta minskar API-trafiken och gör att man kan bläddra genom långa listor smidigt.



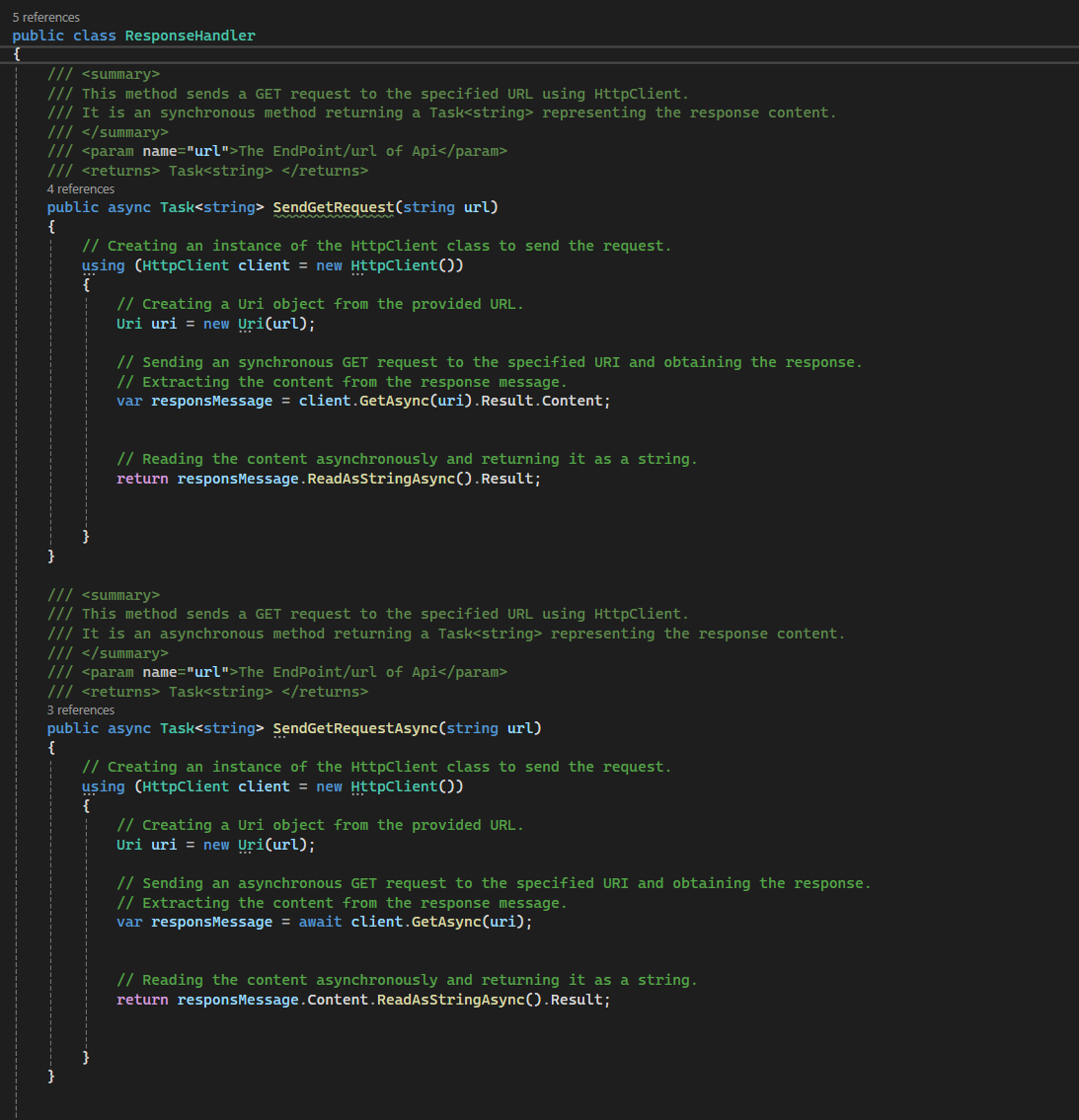
**Product DTO (DataHandlingLibrary.ProductNamespace)**  
**Designval** | **Motivering**  
**Platt POCO med publika fält (istället för egenskaper + getters)**  
Dessa objekt är rena databehållare. Genom att ta bort beteende minimerar vi minnesanvändningen och låter Newtonsoft.Json fylla dem blixtsnabbt via reflektion.

**[JsonProperty("jsonKey")]-attribut**  
DummyJSON API:et använder camelCase (t.ex. discountPercentage); vår C#-namngivning kan behålla PascalCase samtidigt som serialiseringen ändå matchar 1-till-1.**Stark typning för priser / betyg**  
float/double gör att grafer på **PriceMonitoringPage** kan plottas som numeriska serier utan konverteringar; typomvandling sker bara en gång vid deserialisering.

**Förvaras i eget namespace**  
Att separera **Products**, **Posts**, **Users**, **Comments** i olika namespaces håller IntelliSense fokuserat och förhindrar oavsiktlig felanvändning.

**Varför detta spelar roll i praktiken**  
Listcellmallarna (**ListItemCell**) får ett fullständigt ifyllt **Product**-objekt och kan binda direkt till **Title**, **Price**, **Rating** etc. – inget extra mappningslager behövs.

När du PATCH-uppdaterar en användare används ett annat DTO-objekt (**User**), men mönstret är identiskt, så nya entiteter kan läggas till på några minuter.

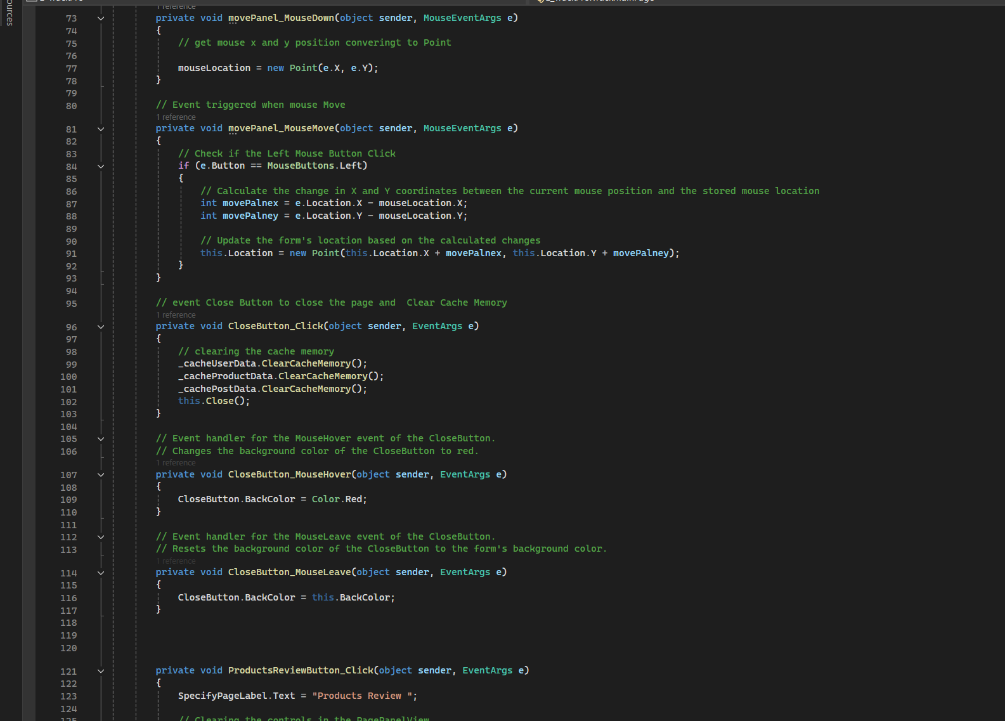
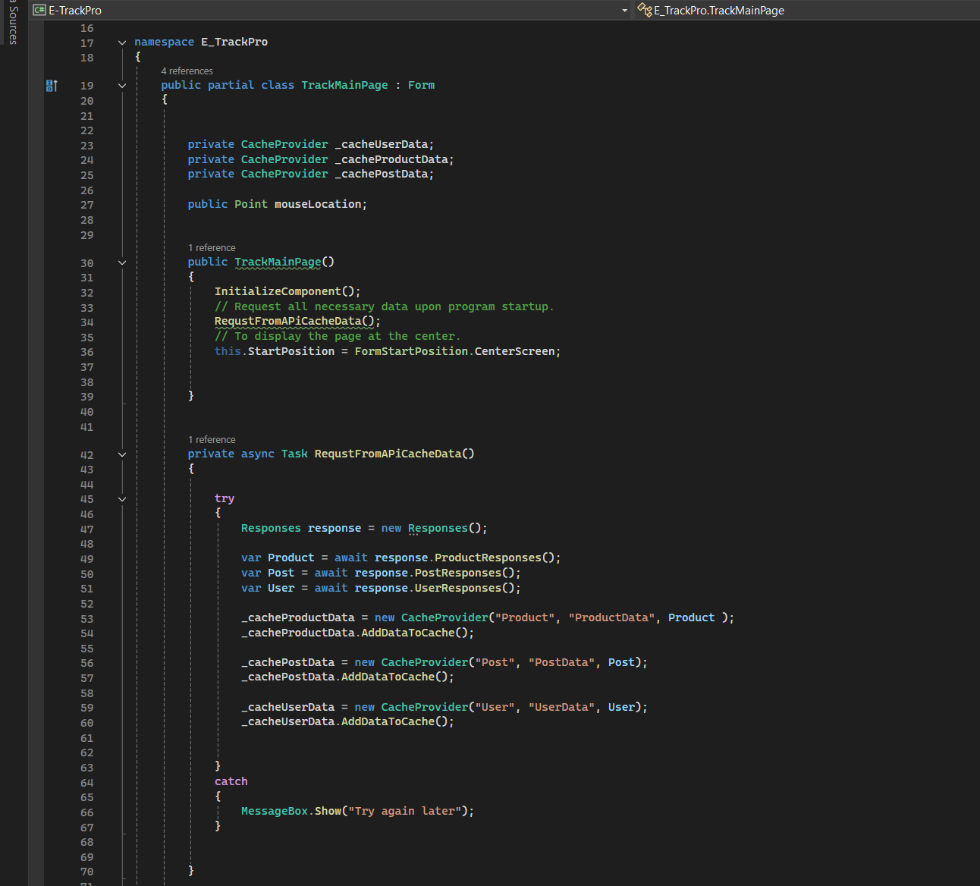


**ResponseHandler – liten HTTP-wrapper**  
**Varför omsluta HttpClient?** | **Vad koden gör**  
**En gemensam utgångspunkt för varje HTTP-anrop** – enkelt att lägga till loggning, retry-logik, autentiseringsheaders senare.  
SendGetRequest (synkron) och SendGetRequestAsync (asynkron) gör följande:

1. Skapar en tillfällig HttpClient
2. Bygger en Uri
3. Anropar GetAsync
4. Läser svaret som text

**Asynkront hela vägen för UI-trådar**  
Sidor anropar normalt den asynkrona versionen så att WPF-dispatchern/Xamarin-huvudtråden inte blockeras.  
Den synkrona varianten finns kvar för enhetstester eller konsolverktyg där blockering är okej.

**Ren strängin/ut**  
Metoden returnerar rå JSON så att DeserializeJsonData<T>() (nästa metod i filen) kan mata in det i valfritt DTO – produkter, användare, inlägg, kommentarer.



Applikationens “skal”

**WinForms som värd** – Du valde Windows Forms eftersom det startar snabbt, fungerar i alla miljöer där .NET finns och har en designer där du kan placera kontroller på några minuter. Det passar ett demoprojekt vars mål är att *visa API-integration*, inte UI-ramverksmagi.

**partial-klass** – Den ena halvan är designer-genererad kod (TrackMainPage.Designer.cs), den andra halvan (den här filen) är din **logikdel**. Delningen gör att designern kan återskapa layoutkod utan att skriva över din egen logik.

Förinläsning + cache vid start

**Varför ladda allt direkt?**  
Produkter, inlägg och användare ändras sällan på DummyJSON. Att hämta dem en gång gör att alla sidladdningar därefter blir blixtsnabba och kan fungera offline.

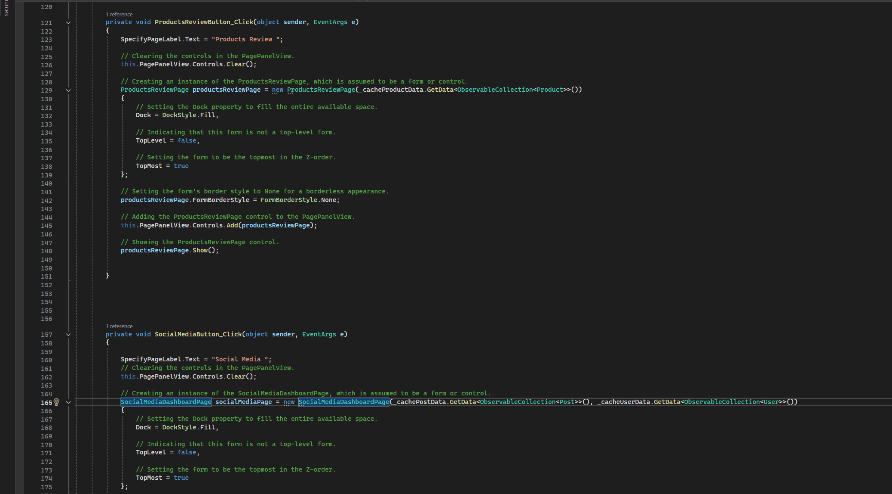
**Varför tre seriella await-anrop?**  
Parallella anrop skulle spara ~100 ms men ge mer komplexitet. Seriella await håller exemplet tydligt samtidigt som UI-tråden förblir responsiv tack vare async/await.

**Varför lindade i CacheProvider?**  
Klassen ger TTL-kontroll och isolerar cachelogiken från UI-koden.

Kantfri ruta med egen drag-logik

* **Ingen standardtitelrad** – Den mörkblå headern du ser i skärmdumparna är bara en Panel. När standardramen tas bort (FormBorderStyle.None) kan du skinna appen i den marinblå-magenta palett du valt.
* **Manuell dragning** – Utan systemets titellist försvinner också Windows inbyggda drag-funktion. Dessa två händelser implementerar dragning genom att spara muspositionen vid nedtryck och flytta formuläret därefter.

A screenshot of a computer

AI-generated content may be incorrect.

Sidomeny-knappar laddar undersidor vid behov

* **Varför rensa och sedan lägga till?**  
  PagePanelView är en placeholder-panel. Genom att rensa undviker du att flera sidor staplas.
* **Cache-data → konstruktor** – Sidan anropar aldrig API:t; den binder bara mot redan deserialiserad ObservableCollection<T>.
* **Form i en panel** – TopLevel = false gör en Form till en kontroll. Ett WinForms-knep som ger full designerstöd för varje sida utan att skriva egna UserControls.
* **Dock = Fill** – Undersidan fyller hela området, medan navigationslisten till vänster ligger kvar.

Samma logik upprepas för Social Media-dashboarden och de andra flikarna.

**Cache-tömning** – Säkerställer att inga MemoryCache-poster ligger kvar om du startar om appen i samma process (praktiskt vid debugging).

**UI-feedback** – Hover/leave-händelser färgar stängikonen röd vid hovring för att signalera “farlig” klick.