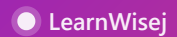


KI-Coding-Übung

Erstellen Sie das Beispielprojekt TicketOps dieses Moduls mit ChatGPT oder Claude aus einem fertigen Prompt, der das Modell auf die Spezifikation des Moduls verweist – und prüfen, starten und erweitern Sie dann, was Sie erhalten, auch im Vergleich mit unserem eigenen Referenz-Build auf GitHub.



Instruieren, lesen, dann auf die Probe stellen.

Diese Übung hat drei Teile, und nur der erste dreht sich um den Prompt. Sie instruieren einen Assistenten mit der Modulspezifikation und erhalten eine TicketOps-Projektmappe zurück; Sie lesen, was er geschrieben hat, und entscheiden, ob es wirklich tut, was das Modul verlangt; dann treiben Sie es weiter, als das Modul verlangt hat. Im zweiten und dritten Teil steckt das eigentliche Lernen.

Worum Sie bitten

Refaktorisieren Sie die Anwendung so, dass sie injizierte Services für Tickets, Benutzer, Berechtigungen, Benachrichtigungen und Audit-Logging verwendet. Ergänzen Sie ein Fake-Registrierungsprofil und ein Produktiv-Registrierungsprofil.

Das Modul verlangt diese Ergebnisse, und der Prompt unten fordert genau diese vom Assistenten an:

- ✓ Methode zur Service-Registrierung
- ✓ fünf Interfaces mit Fake-Implementierungen
- ✓ injizierte MainPage/Form
- ✓ Presenter oder Workflow-Service für eine Maske
- ✓ Tabelle der Service-Lebensdauern

Diese Begriffe muss der Assistent richtig verwenden. Geht die Antwort nachlässig mit ihnen um, hat er das Modul nicht verstanden:

Begriff	Was er hier bedeutet
Dependency Injection (DI)	eine Klasse benennt über ein Interface, was sie braucht, und erhält eine Implementierung von außen, statt selbst eine zu erzeugen.
Service-Container / IServiceProvider	die Registry, die einen Vertrag seiner Implementierung und Lebensdauer zuordnet und Instanzen bei Bedarf auflöst.
Service-Lebensdauer	wie lange eine aufgelöste Instanz lebt: Anwendung/Singleton (geteilt), sessionbezogen (pro Benutzer) oder transient (pro Auflösung).
Sessionbezogener Service	eine Instanz pro Benutzer-Session; der richtige Ort für benutzerbezogenen Kontext (Modul 2).
Konstruktor-Injection	Abhängigkeiten werden als Konstruktorparameter geliefert; für Klassen, die Sie oder der Container erzeugen.
Property-Injection	eine Abhängigkeit wird nach der Erzeugung einer markierten Eigenschaft zugewiesen; für vom Framework erzeugte Forms/Pages, die Sie nicht selbst bauen.
Presenter / Application-Service	eine schlichte, browserfreie Klasse mit der Entscheidungslogik, an die ein Form delegiert, wodurch diese Logik per Unit-Test prüfbar wird.
Fake / Test-Double	eine Ersatzimplementierung eines Interfaces, die in einem Test anstelle des echten Service verwendet wird.

Selbst bauen – wählen Sie Ihren Assistenten

Beide Buttons öffnen einen neuen Tab, in dem **derselbe fertige Prompt** bereits eingetragen ist. Der Prompt verweist das Modell auf die Spezifikation dieses Moduls – eine einzelne Markdown-Datei mit der Lektion, der Lab-Aufgabe und den geforderten Ergebnissen –, sodass es mit dem echten Material arbeitet, statt zu raten, worum es in Modul 8 geht. Sie können [module-08.md](#) selbst öffnen, um genau zu sehen, was es lesen wird.



ChatGPT verwenden

Öffnet chatgpt.com mit dem Prompt dieses Moduls. Funktioniert mit GPT-5 oder Codex; jeder Tarif mit Websuche ruft die Spezifikation ab.



Claude verwenden

Öffnet claude.ai mit demselben Prompt. Wenn Sie Claude Code nutzen, können Sie ihn stattdessen in einem leeren Ordner in ein Terminal einfügen.



Unverändert abgeschickt, erhalten Sie ein Projekt, kein Code-Fragment

Der Prompt verlangt am Ende die gesamte Projektmappe als einzelne `.zip`-Datei mit Anleitung zum Kompilieren und Starten. Laden Sie sie herunter, öffnen Sie die Projektmappe in Visual Studio, drücken Sie F5, und TicketOps startet auf `localhost`. Sie benötigen das .NET SDK und das NuGet-Paket `Wisej-4`; alles andere soll der Assistent laut Prompt selbst schreiben.

Der vollständige Prompt – lesen Sie ihn und passen Sie ihn an, wenn Sie ein anderes Beispiel möchten

Ich arbeite gerade Modul 8 – „Services, Dependency Injection und testbare UI-Muster“ – des LearnWisej-Kurses „Produktivarchitektur im Detail“ durch und möchte das Beispielprojekt des Moduls gemeinsam mit dir erstellen. Bitte antworte auf Deutsch.

Lies zuerst die Modulspezifikation. Sie ist eine einzelne Markdown-Datei (auf Englisch) mit der vollständigen Lektion, der Lab-Aufgabe und den geforderten Ergebnissen:
<https://learnwisej.com/assets/courses/intermediate/specs/module-08.md>

Erstelle dann ein vollständiges Wisej.NET-4-Projekt, das ich in Visual Studio öffnen kann:

1. Projekt „TicketOps“, C#, net8.0 oder neuer, NuGet-Paket Wisej-4 4.1.x, Kestrel-Host, der `app.UseWisej()` aufruft.
2. Setze um, was das Lab der Spezifikation verlangt: Refaktorisieren Sie die Anwendung so, dass sie injizierte Services für Tickets, Benutzer, Berechtigungen, Benachrichtigungen und Audit-Logging verwendet. Ergänzen Sie ein Fake-Registrierungsprofil und ein Produktiv-Registrierungsprofil.
3. Liefere diese Ergebnisse und sag mir, wo sich jedes davon in der Projektmappe befindet:
 - Methode zur Service-Registrierung
 - fünf Interfaces mit Fake-Implementierungen
 - injizierte MainPage/Form
 - Presenter oder Workflow-Service für eine Maske
 - Tabelle der Service-Lebensdauern
4. Halte dich an die Spezifikation, wo immer sie eine Wisej.NET-Klasse, -Eigenschaft oder ein -Event nennt – verwende genau diese. Wo sie eine Wahl offenlässt, sag mir, welche API du gewählt hast und warum.
5. Der Server bleibt maßgeblich: Zustand und Validierung leben in .NET, der Browser stellt dar und meldet zurück. Verlagere nie eine Entscheidung nach JavaScript, nur weil es dort einfacher ist.
6. Mach das Ergebnis vorführbar: Ich muss es starten und das Verhalten sehen können, um das es in diesem Modul geht – auch, was passiert, wenn etwas schiefgeht.

Bevor du Code schreibst, fasse in eigenen Worten zusammen, was du bauen wirst und wo jedes Teil liegen wird, damit ich den Entwurf korrigieren kann, solange das noch wenig kostet.

Wenn der Code fertig ist, packe die gesamte Projektmappe in eine einzelne `.zip`-Datei zum Herunterladen und nenne mir die genauen Schritte, um sie zu kompilieren und auszuführen.

Frag mich nach allem, was unklar ist, bevor du anfängst.

Prüfen Sie dann, was zurückkommt

Code, der kompiliert, ist noch lange kein richtiger Code. Die spannenden Fragen sind Designfragen: Ist das wirklich das Wisej.NET-Control, das die Aufgabe verlangt? Folgt der Code dem Lebenszyklus, den das

Framework vorgibt? Liegt der maßgebliche Zustand auf dem Server – oder wurde er im Browser abgelegt, weil das bequemer war? Lesen Sie das Ergebnis als Reviewer, nicht als Kunde.

Alles, was Sie korrigieren mussten, sollten Sie festhalten. Eine kurze Notiz zu den Konventionen und Entscheidungen, auf die Sie sich festgelegt haben – eingefügt in die erste Nachricht Ihrer nächsten Sitzung –, ist mehr wert als jeder noch so clevere Prompt.



Dieses Beispiel haben wir schon gebaut – vergleichen Sie Ihres mit unserem

Die Referenzimplementierung zu Modul 8 liegt im öffentlichen Beispiel-Repository. Bauen Sie zuerst Ihre eigene Version und öffnen Sie dann unsere: Die Unterschiede sind der interessante Teil.



[Beispiel zu Modul 8 auf GitHub öffnen](#)

Oder klonen Sie alle Beispiele mit `git clone https://github.com/iceteagroup/LearnWisej-Samples` und öffnen Sie `Production Architecture Deep Dive/Module 8`.



Unser Beispiel ist ein Ausgangspunkt, kein fertiges Produkt

Es ist genauso entstanden wie Ihres – per Prompt und anschließend korrigiert – und endet bewusst dort, wo das Modul endet. Rechnen Sie mit Platzhaltern, wo Produktivcode hingehört, mit knapper Fehlerbehandlung und ohne Tests. Verstehen Sie das als Einladung: Nutzen Sie, was Sie in diesem Modul gelernt haben, und dazu den Assistenten, um es weiterzuentwickeln. Code zu verbessern, den Sie nicht selbst geschrieben haben, ist die Fähigkeit, die dieser Kurs eigentlich vermittelt.

Drei Prompts für den nächsten Schritt

Richten Sie Ihren Assistenten auf das Projekt, das Sie gerade gebaut haben – oder auf unseres –, und probieren Sie diese Prompts aus. Jeder ist eine Übung zum Prüfen und Erweitern, kein Neuschreiben.

- ✓ **Machen Sie es fit für einen schlechten Tag.** *"Untersuche dieses TicketOps-Projekt auf Fehlerszenarien, die es noch nicht behandelt – fehlerhafte oder fehlende Eingaben, eine Abhängigkeit, die langsam oder nicht erreichbar ist, Arbeit, die eintrifft, nachdem der Benutzer die Seite verlassen hat, und zwei Aktionen, die sich gegenseitig in die Quere kommen. Sag mir für jedes zuerst, ob es in Wisej.NET 4 ein echtes Risiko ist und warum, und schreib erst dann die Korrektur. Wo du entscheidest, dass nichts behandelt werden muss, sag das und begründe es, statt defensiven Code hinzuzufügen."*
- ✓ **Ergänzen Sie den zweiten Fall.** *"Erweitere dieses TicketOps-Projekt um einen zweiten Fall, der demselben Muster folgt wie der erste – eine weitere Entität, Maske oder ein Control derselben Form, auf dieselbe Weise verdrahtet. Lagere wirklich Gemeinsames in eine Basisklasse oder einen Service aus, aber nur dort, wo es sich lohnt; zwei ähnliche Dinge brauchen nicht immer eine Abstraktion. Zeig mir den Diff und begründe jede Auslagerung."*
- ✓ **Beweisen Sie, dass es wirklich funktioniert.** *"Geh dieses Projekt Ergebnis für Ergebnis anhand der Modulspezifikation unter <https://learnwisej.com/assets/courses/intermediate/specs/module-08.md> durch und zeig mir den Beleg, dass jedes wirklich erledigt ist – den Codepfad, worauf ich klicken muss und was ich sehen sollte. Wo ein Ergebnis nur teilweise erfüllt ist, sag das klar, statt es schönzureden. Ergänze dann, was fehlt."*



Nehmen Sie es mit ins Lab

Das praktische Lab am Ende dieses Moduls verlangt dieselben Ergebnisse, die Sie gerade vom Assistenten haben erstellen lassen. Nutzen Sie, was Sie hier haben, als Gerüst für Ihre Antwort und als Beleg dafür – nicht als die Antwort selbst. Bewertet wird im Lab Ihre Argumentation, und dieser Teil muss von Ihnen stammen.