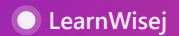


KI-gestützte Entwicklung, MCP-fähige Dokumentation und Auslieferung des Abschlussprojekts

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Disziplin beim KI-gestützten Programmieren, Zugriff auf Dokumentation, Review-Prompts, abschließende Enterprise-Verteidigung. Was das für ein Wisej.NET-Team auf Expertenniveau bedeutet – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Disziplin beim KI-gestützten Programmieren, Zugriff auf Dokumentation, Review-Prompts, abschließende Enterprise-Verteidigung.

1. Das Problem im Produktivbetrieb

Erfahrene Teams werden KI-Tools einsetzen, doch Enterprise-Arbeit mit Wisej.NET erfordert weiterhin Architektur, quellenbasierte Dokumentation, Code-Reviews, Sicherheitsreviews und klare Verantwortung im Betrieb.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
KI-gestützte Entwicklung	KI-Tools entwerfen Code, Tests und Dokumentation, während das Team dafür verantwortlich bleibt, das Ergebnis zu prüfen, mit Quellen zu belegen und zu vertreten.
Quellenbasierte Prompts	Prompts, die den Assistenten auf den tatsächlichen Projektcode, die Dokumentation und die Konventionen verweisen, damit die generierte Ausgabe zur echten Architektur passt.
MCP-Dokumentationsendpunkt	Ein Model-Context-Protocol-Server, der die Dokumentation des Teams für KI-Tools bereitstellt, sodass Antworten aus den eigenen Quellen des Projekts stammen.
Review-Checkliste	Eine feste Liste von Prüfungen zu Architektur, Sicherheit und Konventionen, die auf jede KI-generierte Änderung angewendet wird, bevor sie übernommen wird.
Architekturverteidigung	Die Designentscheidungen des Abschlussprojekts werden mit Belegen vorgestellt und begründet, damit das Team sie hinterfragen und das Ergebnis festhalten kann.
Demo des Abschlussprojekts	Die durchgängige Vorführung der fertigen Anwendung, die belegt, dass die Muster der Module unter realistischen Bedingungen zusammenspielen.

3. Der rote Faden der Lektion

KI hilft dem Team, schneller voranzukommen

KI kann Bildschirme, Services, Tests, Prompts, Dokumentation und Migrationsinventare entwerfen. Die Verantwortung für Risiken im Produktivbetrieb kann sie nicht übernehmen. Der erfahrene Entwickler muss den Architekturkontext liefern, APIs verifizieren, generierten Code prüfen und Fehlerpfade testen.

Dokumentation als Quelle für Tools nutzen

Die Dokumentation von Wisej.NET kann als verlässliche Quelle für das aktuelle Verhalten der Plattform dienen. KI-Prompts sollten Code verlangen, der den Projektkonventionen folgt, und bei plattformspezifischen APIs Quellenangaben oder Referenzen einfordern.

Tempo ohne ungeprüften Code

KI-Unterstützung ist an den Rändern der Arbeit am wertvollsten: ein Service-Gerüst aus einer Command-Definition entwerfen, Testfälle für eine Matrix von Fehlerpfaden generieren, ein Migrationsinventar aus einer Solution-Datei erstellen oder ein Runbook in eine Checkliste umwandeln. In jedem dieser Fälle weiß der Entwickler bereits, wie das richtige Ergebnis aussieht, und kann die Ausgabe schnell überprüfen. Am wenigsten wert ist sie dort, wo die Überprüfung schwer ist, etwa bei Sicherheitslogik, Nebenläufigkeit oder Plattform-APIs, die der Entwickler nicht gut kennt.

Die Regel, die das Lab durchsetzt, lautet: Generierter Code ist ein Entwurf eines unbekannten Beitragenden. Er durchläuft dasselbe Review wie jeder Pull Request, mit denselben Fragen: Wo liegt der Zustand, was kann fehlschlagen, was ist sicher, und was ändert sich im Produktivbetrieb? Der Fehlerpfad im

Lab zeigt, warum. Ein generierter Service erfindet eine Wisej.NET-API, die es nicht gibt, und speichert benutzerbezogene Daten in einem statischen Feld. Beides würde in einer schnellen Demo kompilieren, und beides würde mit echten Sessions scheitern.

Übernehmen Sie einen Vorschlag nur, wenn jemand im Team ihn auch ohne das Tool erklären kann.

Prompts und Checklisten, die auf dem Projekt beruhen

Die Qualität des generierten Codes folgt der Qualität des Kontexts, den er bekommen hat. Jede KI-Sitzung im Lab beginnt mit einem Prompt-Kopf, der die Projektregeln nennt: die Solution-Struktur, wo die Geschäftslogik liegt, dass statische Felder niemals Session-Daten halten, dass Services Berechtigungen durchsetzen und dass nur dokumentierte Wisej.NET-APIs verwendet werden dürfen – jeweils mit Referenz. Die Prompt-Bibliothek sammelt diese Köpfe und aufgabenspezifische Prompts, sodass das ganze Team von derselben Grundlage aus startet.

Die Review-Checkliste für generierten Code ist die andere Hälfte. Sie ist kurz genug, um sie auf jede Änderung anzuwenden, und konkret genug, um die Fehler abzufangen, die der Kurs behandelt hat: ein statisches Feld mit Benutzer- oder Mandantenzustand, ein UI-Handler mit Persistenzlogik, eine API, die sich in der Dokumentation nicht finden lässt, eine fehlende Berechtigungsprüfung in einem Service, eine Exception, die bis zum Benutzer durchschlägt, und eine Ressource, die nie freigegeben wird.

- Nennt der Prompt die Konventionen, denen die Ausgabe folgen muss?
- Belegt jede plattformspezifische Aussage in der Ausgabe ihre Quelle in der Dokumentation?
- Wurde die Checkliste angewendet und von einem Reviewer abgezeichnet – nicht vom Autor?

Dokumentation, die Tools und Maintainer lesen können

Ein Projekt, das mit KI-Unterstützung gewartet wird, braucht eine Dokumentation, die ein Tool abrufen kann, nicht nur eine, in der ein Mensch blättern kann. Ein MCP-Dokumentationsendpunkt stellt dem Assistenten die eigenen Architekturnotizen, Entscheidungsprotokolle, Konventionen und Runbooks des Teams bereit, sodass Antworten zu diesem Projekt aus dessen eigenen Quellen stammen und nicht aus allgemeinen Trainingsdaten. Derselbe Endpunkt bedient auch ein neues Teammitglied, das dieselben Fragen stellt.

Das funktioniert nur, wenn die Dokumentation strukturiert ist. Der Index der Projektdokumentation im Lab listet jedes Dokument mit einem einzeiligen Zweck, den darin festgehaltenen Entscheidungen und dem Datum der letzten Überprüfung auf. Die Architecture Decision Records aus Modul 1 sind die wertvollsten Einträge, denn sie erklären, warum ein Design existiert, und ein Tool kann sie zitieren, wenn es gebeten wird, dieses Design zu ändern. Es folgen Runbooks, die Berechtigungsmatrix, die Matrix der Fehlerpfade und die Konfigurationstabelle.

Schreiben Sie für Leser, die das Team nie kennengelernt haben. Stützt sich eine Notiz auf Wissen, das nur in jemandes Kopf existiert, ist sie in einem Jahr falsch.

Das Abschlussprojekt als Architektur-Review verteidigen

Das Abschlussprojekt wird als Architektur-Review für den Produktivbetrieb abgegeben, nicht als Feature-Demo. Das Demo-Skript führt einen Reviewer in fester Reihenfolge durch das EnterpriseOps Command Center: die Solution-Struktur und die Entscheidungsprotokolle, eine mandantenfähige Session und eine

abgelehnte mandantenübergreifende Anfrage, ein Command, der durch einen Service zur Datenbank und mit seinem Audit-Eintrag wieder zurückläuft, ein Hintergrundimport mit Fortschritt und Abbruch, ein Workflow mit einem kompensierten Fehler, die Diagnosesseite und das Deployment-Paket mit seinem Rollback.

Jede Station kombiniert einen funktionierenden Bildschirm mit einem Fehlerpfad, denn im Review geht es darum, was passiert, wenn etwas schiefgeht. Der Reviewer wird fragen, wo der Zustand liegt, wer was aufrufen darf, wie ein fehlerhaftes Release zurückgenommen wird und was das Team um zwei Uhr nachts tun würde. Die Erklärung zur Produktivreife beantwortet diese Fragen schriftlich, listet auf, was noch nicht erledigt ist, und benennt für jeden offenen Punkt einen Verantwortlichen.

Eine Verteidigung gelingt, wenn sich jede Entscheidung auf einen Grund zurückführen lässt, jedes Risiko eine benannte Gegenmaßnahme hat und das Team klar sagen kann, was es als Nächstes ändern würde.



Expertenregel für den Produktivbetrieb

Hören Sie nicht auf, sobald der Bildschirm korrekt aussieht. Erklären Sie, wem der Zustand gehört, welche Sicherheitsfolgen es gibt, welche Fehlerpfade existieren und wie sich das System im Produktivbetrieb verhält, bevor Sie das Modul als abgeschlossen betrachten.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ KI-Unterstützung nutzen, um die Wisej.NET-Entwicklung zu beschleunigen, ohne ungeprüften Code zu übernehmen.
- ✓ Prompts und Review-Checklisten erstellen, die auf Projektstandards und der offiziellen Dokumentation beruhen.
- ✓ MCP- und dokumentationsfähige Projektnotizen für künftige Maintainer vorbereiten.
- ✓ das Abschlussprojekt des Experten-Tracks als Architektur-Review für den Produktivbetrieb ausliefern und verteidigen.

Lab- und Prüfungsleitfaden

Was Sie im Praxis-Lab bauen, das vorgeschlagene Vorgehen und die geforderten Ergebnisse.



Praxis-Lab

Stellen Sie das Paket des Abschlussprojekts fertig. Ergänzen Sie Notizen zur KI-Nutzung, eine Prompt-Bibliothek, Links zur Dokumentation, eine Code-Review-Checkliste, ein Skript für die finale Demo, eine Gliederung für die Präsentation zur Architekturverteidigung und eine Erklärung zur Produktivreife.

Vorgeschlagenes Vorgehen

- ✓ Lesen Sie das Problem im Produktivbetrieb und benennen Sie das Architekturrisiko, bevor Sie die UI anfassen.
- ✓ Legen Sie zuerst die betroffene Grenze für Service, Command, Modell oder Komponente an oder aktualisieren Sie sie.
- ✓ Erstellen Sie den Wisej.NET-Bildschirm oder die Komponente nach den etablierten Projektstandards.
- ✓ Demonstrieren Sie auch einen Fehlerpfad, nicht nur den Happy Path.
- ✓ Halten Sie die Änderungen im Lernprotokoll fest und aktualisieren Sie die zugehörige Checkliste.

Geforderte Ergebnisse

- ✓ KI-Prompt-Bibliothek
- ✓ Review-Checkliste für generierten Code
- ✓ Index der Projektdokumentation
- ✓ Demo-Skript für das Abschlussprojekt
- ✓ Erklärung zur Produktivreife

Wiederholungsfragen

- ✓ Welcher generierte Code wurde übernommen und warum?
- ✓ Wurde jede plattformspezifische Aussage anhand der Dokumentation geprüft?
- ✓ Kann der Teilnehmer seine Entscheidungen zu Deployment, Sicherheit und Zustand verteidigen?



Expertenregel für den Produktivbetrieb

Hören Sie nicht auf, sobald der Bildschirm korrekt aussieht. Erklären Sie, wem der Zustand gehört, welche Sicherheitsfolgen es gibt, welche Fehlerpfade existieren und wie sich das System im Produktivbetrieb verhält, bevor Sie das Modul als abgeschlossen betrachten.