

KI in der Access-Entwicklung

Teilnehmerunterlage zum Doppelvortrag

AEK 2026 Nürnberg 10. und 11. Oktober 2026

AVANTGARDE

Weitermachen im KI-Club für Access-Entwickler

Ein Vortrag zeigt, was möglich ist. Im eigenen Projekt kommen dann die Fragen, die in keiner Unterlage stehen: Warum hängt der Agent an diesem Formular? Wie richte ich Claude so ein, dass es meine Konventionen kennt? Was mache ich mit dem Modul, das seit fünfzehn Jahren niemand anfassen will? Für genau diese Fragen gibt es den [Avantgarde KI-Club für Access-Entwickler](#).

Der größte Mehrwert: der monatliche Club-Termin

Einmal im Monat treffen sich alle Clubmitglieder zu einem gemeinsamen Termin. Dort tauschen Sie sich mit anderen Access-Entwicklern aus, die vor denselben Fragen stehen, und bringen Ihre eigenen Probleme mit: wie sich ein konkretes Access-Problem mit KI lösen lässt, wie man Claude für die eigene Arbeit einrichtet, welche Werkzeuge sich bewährt haben und welche nicht.

Jedes Mitglied bringt Erfahrungen aus eigenen Projekten mit. Davon profitieren alle: Was einer gelöst hat, muss der Nächste nicht noch einmal mühsam herausfinden.

Konditionen

Was	Wie
Mitgliedsbeitrag	99 Euro monatlich zzgl. MwSt.
Laufzeit	Jahresvertrag
Enthalten	monatlicher Club-Termin mit Austausch und offener Fragerunde

Unverbindlich vormerken lassen

Die Anmeldung läuft über ein kurzes Microsoft-Forms-Formular:
QR-Code scannen oder
<https://forms.cloud.microsoft/e/FuTZgTmL2g> öffnen.

Die Anmeldung ist unverbindlich. Mit dem Absenden des Formulars kommt noch kein Vertrag zustande. Sie erhalten zuerst alle Informationen zum KI-Club und entscheiden danach, ob Sie Mitglied werden.



Inhaltsverzeichnis

Weitermachen im KI-Club für Access-Entwickler	1
1 Worum es geht	4
1.1 Was Sie mitnehmen sollen	4
2 Teil 1: KI verstehen	4
2.1 Die vier Bauformen	4
2.2 Kontext, Projekte, Skills	4
2.2.1 Kontext	4
2.2.2 Projekte	5
2.2.3 Skills	5
2.3 Wo die Grenzen wirklich liegen	5
2.4 Ein Beispiel, wie man es nicht macht	5
3 Teil 2: MCP in der Praxis	6
3.1 Was ein MCP-Server ist	6
3.2 Wie wenig Code das ist	6
3.3 MCP ist kein anderer Zugang, sondern eine Schicht	6
3.4 Das Schwierige ist nicht MCP, sondern Access	7
3.4.1 COM gehört einem Thread	7
3.4.2 Die Falle in der Falle	7
3.4.3 Access wartet auf einen Mausklick	7
3.4.4 Bereiche gibt es nicht immer	7
3.4.5 Ihr Bestandscode redet mit dem Anwender	7
3.4.6 Reservierte Wörter, die niemand kennt	8
4 Eigenbau oder Fertiglösung	8
4.1 Warum die Demo auf unmateria setzt	8
4.2 Wann Eigenbau sinnvoll ist	8
4.3 Sicherheit ist hier etwas anderes als bei einer API	9
5 Voraussetzungen, die selten jemand nennt	9
5.1 Ohne Versionsverwaltung geht es nicht	9
5.2 Die Windows-Kommandozeile ist ungeeignet	9
5.3 Access führt Ihren Code vielleicht gar nicht aus	9
5.4 Was sich an Nutzen messen lässt	10
6 Drei Ebenen, auf denen KI vorkommt	10
6.1 Was das für die Architektur bedeutet	11
7 Datenschutz, DSGVO und EU AI Act	11
7.1 Die Fragen, die Sie vor dem ersten Prototyp klären	11
7.2 Was der EU AI Act für typische Access-Anwendungen bedeutet	11
7.3 Prompt Injection	11
8 Demo 3: Modernisierung, und warum sie über MCP läuft	12
8.1 Was umgebaut wird	12
8.2 Der Erfolg ist gemessen, nicht behauptet	12
8.3 Zwei Funde, die den Aufwand wert waren	12
8.3.1 Auch der neue Code von gestern ist morgen Bestand	12

8.3.2 Ein Refactoring kann kaputtmachen, was es verbessern soll	13
9 Stufe 3 zum Anfassen: dieselbe Anwendung als Weblösung	13
9.1 Die eine Frage, die über die Migration entscheidet	13
9.2 Und genau so ist es in Access	13
10 Warum die Demo-Datenbank so aussieht	13
11 Kontakt	14

1 Worum es geht

Diese Unterlage begleitet zwei Vorträge. Der erste ordnet ein, was in der KI-Welt gerade passiert und was davon für Access-Entwickler zählt. Der zweite zeigt an einem durchgehenden Beispiel, wie sich das konkret nutzen lässt: der Badverwaltung [Südbad Duisburg](#).

Alles, was hier steht, ist in der Praxis erprobt und nicht abgeschrieben. Die Unterlage ist so aufgebaut, dass Sie jeden Schritt im eigenen Projekt nachvollziehen können.

1.1 Was Sie mitnehmen sollen

- Eine Landkarte: welches KI-Werkzeug wofür, und wo die Grenzen liegen
- Das Verständnis, was ein MCP-Server ist und warum er den Unterschied macht
- Drei Ebenen, auf denen KI in Ihren Projekten vorkommen kann
- Eine ehrliche Einschätzung, was heute funktioniert und was nicht
- Die rechtlichen Punkte, die in der Praxis regelmäßig unterschätzt werden

2 Teil 1: KI verstehen

2.1 Die vier Bauformen

Bauform	Was es ist	Wofür im Alltag
Chatfenster	Ein Modell, ein Textfeld, kein Zugriff auf Ihre Welt	Fragen, Erklären, Entwürfe, Textarbeit
Assistent in der IDE	Vervollständigung im Editor, sieht die offene Datei	Tippen abkürzen, Muster fortschreiben
Agent mit Werkzeugen	Darf Dateien lesen, Befehle ausführen, Ergebnisse prüfen	Mehrschrittige Arbeit, Umbauten, Recherche im Bestand
Agent mit MCP	Zusätzlich angebunden an Fremdsysteme, hier: Access	Die Anwendung selbst verändern

Der Sprung, der zählt, liegt zwischen Zeile zwei und drei. Ein Assistent schlägt Text vor, den Sie übernehmen. Ein Agent handelt, prüft das Ergebnis und korrigiert sich. Das ist ein Unterschied in der Art, nicht im Grad.

2.2 Kontext, Projekte, Skills

Drei Begriffe, die in der Praxis alles entscheiden.

2.2.1 Kontext

Das Modell sieht nur, was im Fenster steht. Nichts anderes. Es erinnert sich nicht an gestern, es kennt Ihre Datenbank nicht, und es weiß nicht, was Sie im Nachbarfenster gemacht haben. Jede Werkzeugbeschreibung, jede Datei, jede Zeile Verlauf belegt Platz in diesem Fenster.

Die Größenordnung: Bei großen MCP-Servern verbrauchen allein die Werkzeugbeschreibungen über 50.000 Token, bevor die eigentliche Aufgabe beginnt. Und die Treffsicherheit, mit der ein Modell das richtige Werkzeug auswählt, lässt [ab etwa 30 bis 50 Werkzeugen](#) spürbar nach.

Ein Werkzeugkasten mit 360 Funktionen ist deshalb kein Vorteil, sondern eine Hypothek. Worauf es ankommt, ist nicht die kleinere Zahl, sondern der passendere Zuschnitt – dazu mehr in Abschnitt 4.1.

2.2.2 Projekte

Ein Projekt bündelt wiederkehrenden Kontext: Ihre Namenskonventionen, die Eigenheiten der Kundendatenbank, die Regel, dass Beträge immer als Currency zu führen sind. Einmal hinterlegt statt in jedem Gespräch neu erklärt.

2.2.3 Skills

Eine Anleitung, die das Modell bei Bedarf nachliest. Etwa: „So sieht bei uns ein Formular aus.“ Der Unterschied zum Projekt ist, dass ein Skill nur geladen wird, wenn er gebraucht wird, und deshalb keinen Platz kostet, solange er nicht dran ist.

2.3 Wo die Grenzen wirklich liegen

Es heißt oft	Tatsächlich
„Die KI kann kein VBA“	Doch, gut sogar. Was sie nicht kann, ist Ihre Fachlichkeit erraten.
„Die KI baut hässliche Formulare“	Stimmt, sie setzt Koordinaten blind. Deshalb prüft ein guter MCP-Server das Layout nach jeder Änderung deterministisch nach.
„Die KI halluziniert“	Beim Raten ja. Mit Werkzeugen, die den echten Zustand liefern, deutlich seltener. Der Unterschied ist <i>lesen statt vermuten</i> .
„Das ersetzt Entwickler“	Nein. Es verschiebt den Aufwand von der Umsetzung zur Spezifikation und zur Prüfung. Beides bleibt Ihre Arbeit.

2.4 Ein Beispiel, wie man es nicht macht

Im Umlauf ist eine Grafik mit zwei Köpfen nebeneinander: der eine, beschriftet „ChatGPT user“, fast dunkel, der andere flächig orange leuchtend. Darunter: „MIT’s first brain scan study of ChatGPT users revealed shocking results.“

Was tatsächlich in der Arbeit steht (Kosmyna et al., MIT Media Lab, 2025, „Your Brain on ChatGPT“, arXiv:2506.08872):

- **54 Teilnehmer**, drei Gruppen zu je 18: mit LLM, mit Suchmaschine, ohne Hilfsmittel
- Aufgabe: **Aufsätze schreiben**, drei Sitzungen
- Methode: **EEG**, also Messung elektrischer Aktivität an der Kopfhaut. **Keine Scans**. Die farbigen Schnittbilder der Grafik sehen aus wie fMRT-Aufnahmen; EEG erzeugt so etwas nicht. Die Darstellung ist eine Illustration, keine Studiendarstellung.
- Befund: Die Gruppe ohne Hilfsmittel zeigte die stärkste neuronale Konnektivität, die LLM-Gruppe die schwächste. Die LLM-Aufsätze glichen einander stark; zwei Lehrkräfte nannten sie „soulless“. Die LLM-Gruppe konnte die eigenen Texte schlechter wiedergeben.

- Status zum Erscheinen: **Preprint**. Die Autorinnen und Autoren haben selbst vor der Überinterpretation gewarnt.

Der Kern ist trotzdem ernst zu nehmen: Wer sich das Ergebnis liefern lässt, ohne den Weg mitzugehen, behält weniger. Für Entwickler heißt das nicht „nicht benutzen“, sondern: [lesen, was der Agent geschrieben hat](#).

3 Teil 2: MCP in der Praxis

3.1 Was ein MCP-Server ist

Model Context Protocol. Ein Server stellt der KI eine begrenzte Menge Funktionen bereit und beschreibt sie. Die KI entscheidet, welche sie aufruft, und bekommt das Ergebnis zurück. Mehr nicht.

Für Access heißt das: die KI ruft `formular_anlegen` auf, der Server macht daraus COM-Automatisierung, und in Access entsteht ein Formular.

3.2 Wie wenig Code das ist

Ein minimaler MCP-Server für Access kommt mit acht Werkzeugen aus und passt in eine Datei mit rund 400 Zeilen, davon etwa die Hälfte Kommentar. Ein Werkzeug sieht so aus:

```
@server.tool()
def tabelle_beschreiben(name: str) -> str:
    """Liefert die Feldliste einer Tabelle mit Typ, Groesse
    und Pflichtangabe."""
    def arbeit():
        db = _access().CurrentDb()
        td = db.TableDefs(name)
        zeilen = [f"Tabelle {td.Name}"]
        for f in td.Fields:
            typ = _DAO_TYPEN.get(f.Type, f"Typ {f.Type}")
            zeilen.append(f" {f.Name}: {typ}")
        return "\n".join(zeilen)
    return _im_com_thread(arbeit)
```

Der Docstring ist die Beschreibung, die die KI liest. Die Typangaben werden zum Aufrufsche-ma. Sonst nichts.

3.3 MCP ist kein anderer Zugang, sondern eine Schicht

Ein Access-MCP-Server spricht über COM mit dem Access-Objektmodell. Genau dieselbe Schnittstelle nutzt ein Python-Skript mit `pywin32`, nur direkt und ohne Protokoll dazwischen. [MCP eröffnet also keine neuen Möglichkeiten](#). Es ändert, wer entscheidet, welcher Aufruf als nächstes kommt.

	Skript	MCP
Entscheidung	vorher, durch den Entwickler	zur Laufzeit, durch das Modell
Ergebnis	deterministisch, wiederholbar	vom Vorgefundenen abhängig
Stärke	Massenerzeugung	explorative Arbeit im Bestand

Die Demo-Datenbank zu diesem Vortrag ist deshalb *nicht* über einen MCP-Server entstanden, sondern mit Python über COM. 2.373 Zutrittsbuchungen einzeln über einen Agenten einzufügen wäre absurd; ein Skript ist deterministisch, in sechs Minuten durch und selbst ein Deliverable. Wer beides verwechselt, ist mit beidem unzufrieden.

Der Umweg war der Gewinn: alle Fallen im nächsten Abschnitt sind auf dem direkten Weg über COM aufgetreten. Genau diese Probleme muss ein MCP-Server für Access lösen. Wer nie selbst gegen Access automatisiert hat, sieht bei einer Layoutprüfung oder einem Dialog-Wächter nur Features. Danach sieht man Narben.

3.4 Das Schwierige ist nicht MCP, sondern Access

Beim Bau eines eigenen MCP-Servers und der Demo-Datenbank sind wir in genau diese Fallen gelaufen. Alle sind hier verifiziert, nicht abgeschrieben.

3.4.1 COM gehört einem Thread

COM-Objekte im Single Threaded Apartment dürfen nur von dem Thread benutzt werden, der sie erzeugt hat. Ein asynchroner Server würde die Aufrufe auf wechselnden Threads ausführen, Access quittiert das mit „Katastrophaler Fehler“. Lösung: ein einziger Arbeitsthread.

3.4.2 Die Falle in der Falle

Der naheliegende Weg ist, jeden Aufruf sauber mit `CoInitialize` und `CoUninitialize` zu klammern. Das beendet aber das Apartment und entwertet *alle* darin entstandenen Verweise. Der nächste Aufruf läuft gegen -2147220995 „Objekt ist nicht mit dem Server verbunden“. Die Meldung zeigt nie auf die Ursache, immer auf den unschuldigen Folgeaufruf.

3.4.3 Access wartet auf einen Mausklick

Ungespeicherte VBA-Module lässt Access beim nächsten Speichervorgang über den Dialog „Speichern unter“ abfragen. In einer Automatisierung kommt der Klick nie. Das Aufbauskript hing minutenlang *ohne jede Fehlermeldung*, bis ein Blick auf die Fenstertitel der Access-Prozesse den Dialog zeigte. Genau deshalb hat ein ausgereifter MCP-Server einen Dialog-Wachhund als eigenes Bauteil.

3.4.4 Bereiche gibt es nicht immer

Ein per `CreateForm` erzeugtes Formular hat ausschließlich den Detailbereich. Kopf- und Fußbereich lassen sich von außen auch nicht nachrüsten. Ein per `CreateReport` erzeugter Bericht hat dagegen Seitenkopf und Seitenfuß, aber keinen Berichtskopf.

3.4.5 Ihr Bestandscode redet mit dem Anwender

Der VBA-Code der Demo meldet Fehler über `MsgBox`. Genau so sieht gewachsener Access-Code aus, und genau das macht ihn für einen Agenten unbedienbar: `MsgBox` ist modal, der Aufruf kehrt nicht zurück, bis jemand auf OK klickt. Ein Agent klickt nicht. Das Skript hängt, ohne Fehlermeldung, unbegrenzt.

Beim Testen der Demo liefen sechs solcher Dialoge auf. Der Testlauf brauchte deshalb einen Wächter-Thread, der modale Fenster wegklickt, solange ein eigener Aufruf läuft, und der jedes Wegklicken protokolliert.

Die Lehre für jede Modernisierung: **Entscheidung und Benachrichtigung trennen**. Eine Funktion liefert ein Ergebnis, die Oberfläche zeigt es an. Code, der beides vermischt, lässt sich weder automatisieren noch testen noch in ein Web-Frontend heben.

3.4.6 Reservierte Wörter, die niemand kennt

CREATE TABLE tblFeedback (... , Note SHORT, ...) scheitert mit „Syntaxfehler in Felddefinition“. Grund: NOTE ist in Jet ein Datentyp-Alias für Memo. Der Parser liest Note als Typ und stolpert über das SHORT dahinter. Die Meldung nennt weder das Feld noch das Wort.

Solche Eigenheiten sind der Grund, warum ein ausgereifter Access-MCP-Server bei mehreren tausend Zeilen landet. Nicht Funktionsumfang, sondern zwanzig Jahre Access.

4 Eigenbau oder Fertiglösung

Lösung	Technik	Werkzeuge	Wofür
Eigenbau (Beispiel oben)	Python, pywin32	8	Verstehen, eigene Erweiterungen
unmateria / MCP-Access	Python, pywin32	68	Produktive Arbeit im Bestand
brickly26 / MS-Access-mcp	.NET 8	360	Maximale API-Abdeckung
access-mcp.ai	.NET 9, kommerziell	27	Fertig, mit Support und Audit-Trail

4.1 Warum die Demo auf unmateria setzt

- **Zuschnitt statt Vollständigkeit.** Nicht die kleinere Zahl gewinnt, sondern der passendere Zuschnitt. 360 Werkzeuge spiegeln die COM-API wider, 68 spiegeln Arbeitsschritte wider, die acht des Eigenbaus genau eine Aufgabe. Die Treffsicherheit der Werkzeugauswahl lässt [ab etwa 30 bis 50 Werkzeugen](#) spürbar nach, und bei großen Servern verbrauchen allein die Tool-Definitionen über 50.000 Token, bevor die Aufgabe beginnt. Auch 68 liegt also über der Schwelle. Entscheidend ist, dass diese 68 nach Arbeitsschritten geschnitten sind statt nach API-Oberfläche.
- **Layoutprüfung nach jeder Änderung.** Überlappungen, Kontrast, abgeschnittene Beschriftungen werden gemessen, nicht bewertet. Damit lässt sich der Einwand „die KI baut hässliche Formulare“ mit Zahlen beantworten.
- **Codeausführung standardmäßig gesperrt.** Die Werkzeuge, die beliebiges VBA ausführen, sind hinter einer Umgebungsvariablen weggeschaltet.
- **Python, also bitnessunabhängig.** Relevant, wenn Access 32-Bit ist.

4.2 Wann Eigenbau sinnvoll ist

Wenn Sie fachliche Werkzeuge brauchen statt technischer. Ein Werkzeugbeitragslauf_starten ist für die KI leichter richtig zu benutzen als zwanzig SQL-Aufrufe, die zusammen dasselbe tun. Der Eigenbau lohnt sich dort, wo Sie Ihre Fachlichkeit in die Werkzeuge legen, nicht dort, wo Sie Access-Automatisierung nachbauen. Die haben andere schon gemacht.

Zwei Faustregeln bringen das auf den Punkt:

- Muss ein Modell drei bis fünf Werkzeugaufrufe kombinieren, um ein offensichtliches Ziel zu erreichen, sind die Werkzeuge nicht abstrahiert genug.
- Benennen Sie aus der Sicht des Anwenders und in dessen Sprache. createUser gegen provision_trial_account, getUserDetails gegen summarize_customer_activity.

Deshalb heißen die Werkzeuge im Beispiel oben `formular_anlegen` und `tabelle_beschreiben` und nicht `create_form` und `describe_table`. Das Modell versteht deutsche Werkzeugnamen genauso gut, und im Team liest sich der Aufrufverlauf wie ein Protokoll.

4.3 Sicherheit ist hier etwas anderes als bei einer API

Der Unterschied in einem Satz: [Der „Execute“-Knopf lässt sich nicht verstecken.](#)

Bei einer klassischen API entscheiden Sie, wann ein Aufruf passiert. Bei einem MCP-Server entscheidet das Modell. Daraus folgt:

- Das Modell darf nur Werkzeuge *sehen*, die es benutzen darf. Alles andere kostet nur Kontext und lädt zu Fehlern ein.
- Die Eingabe wird vom Modell interpretiert, nicht von Ihrem Code. Validierung gehört auf die Serverseite.
- Rechnen Sie mit Wiederholungen. Idempotenz und Rate Limits sind hier keine Kür.

5 Voraussetzungen, die selten jemand nennt

Aus der Praxis mit KI-Agenten in Bestandsprojekten stammen einige Erfahrungswerte, die in keinem Produktvideo vorkommen.

5.1 Ohne Versionsverwaltung geht es nicht

[Git ist Voraussetzung, nicht Kür.](#) Ein Agent, der Dateien ändert, ist ohne Rückweg ein Risiko und kein Werkzeug. Für Access ist das unbequem, weil viele Bestandsprojekte nie unter Versionsverwaltung standen.

Der Ausweg ist bekannt, wird aber selten genutzt: Formulare, Berichte, Module und Makros lassen sich über `SaveAsText` und `LoadFromText` als Text exportieren und importieren. Damit wird eine Access-Anwendung diff-fähig und versionierbar. Wer mit einem Agenten arbeiten will, sollte diesen Schritt vorher gehen, nicht nachher.

5.2 Die Windows-Kommandozeile ist ungeeignet

Ein KI-Agent in der klassischen Windows-`cmd` ist in der Praxis schlicht unbrauchbar. Nutzbare Alternativen sind Windows Terminal mit PowerShell, WSL oder die Integration in die IDE. Das klingt nach einer Kleinigkeit und entscheidet trotzdem darüber, ob der erste Versuch Freude macht oder Ärger.

5.3 Access führt Ihren Code vielleicht gar nicht aus

Die unangenehmste Falle der ganzen Reihe, weil sie sich als Anwendungsfehler tarnt.

Steht die Makroeinstellung von Access auf `VBAWarnings = 4`, also „Alle Makros ohne Benachrichtigung deaktivieren“, dann führt Access für jede Datenbank außerhalb eines [vertrauenswürdigen Speicherorts](#) keinen einzigen VBA-Befehl aus. Ohne Meldung. Ohne Leiste am oberen Rand. Ohne irgendeinen Hinweis.

Was Sie dann sehen:

- Formulare öffnen sich, aber `Form_Load` läuft nicht
- Listenfelder, die zur Laufzeit gefüllt werden, bleiben leer
- Schaltflächen tun beim Klick nichts

- Es sieht exakt so aus, als wäre der Code kaputt

Beim Bau der Demo für diesen Vortrag ist genau das passiert. Erst ein Blick in die Registrierung hat es gezeigt: VBAWarnings = 4, und der Projektordner war nicht eingetragen. Hätte ich das nicht geprüft, wäre die Demo auf der Bühne gestorben, und ich hätte live nach einem Fehler gesucht, den es nicht gibt.

Zwei Einstellungen sind zu unterscheiden, sie werden oft verwechselt:

Einstellung	Wofür
Vertrauenswürdiger Speicherort	Damit VBA überhaupt <i>ausgeführt</i> wird. Betrifft jede Anwendung.
„Zugriff auf das VBA-Projektobjektmodell vertrauen“	Damit ein Programm den Code <i>lesen und schreiben</i> darf. Braucht jeder MCP-Server, der VBA erzeugt.

Für die Arbeit mit einem Agenten brauchen Sie beide. Der Weg über die Oberfläche: *Datei, Optionen, Trust Center, Einstellungen für das Trust Center*. Tragen Sie dort gezielt nur den Projektordner als vertrauenswürdigen Speicherort ein und lassen Sie die globale Makroeinstellung unverändert.

5.4 Was sich an Nutzen messen lässt

Die folgenden Erfahrungswerte sind deshalb interessant, weil sie aus Projekten mit Bestandssoftware stammen, nicht aus einer Demo:

- Erweiterung eines Bestandsprojekts inklusive Unit-Test erfolgreich
- Metadaten einer Fremdsoftware analysiert, Signatur entschlüsselt, geändert und wieder verschlüsselt
- Interne Vorhaben, die sonst liegen bleiben, in zwei bis drei Minuten als Machbarkeitsnachweis
- **Faktor 5 bis 10** gegenüber den bis dahin genutzten Werkzeugen
- Den gewonnenen Zeitanteil in technische Schulden, Tests und Dokumentation stecken

Und die Kehrseite, ebenso beobachtet: Dummy-Projekte wurden nicht als solche erkannt, und die Skill-Strukturen der verschiedenen Claude-Oberflächen unterscheiden sich. Beides kostet Zeit, wenn man es nicht weiß.

6 Drei Ebenen, auf denen KI vorkommt

Ebene	Wer benutzt sie	Was dabei entsteht
KI für den Entwickler	Sie	Schnellere Entwicklung, Wartung, Dokumentation
KI im Produkt	Ihre Anwender	Ein Funktionsumfang, den Sie verkaufen können
KI für die Modernisierung	Sie im Kundenprojekt	32 auf 64 Bit, Backend-Migration, Ablösung von Access

Die mittlere Ebene wird am häufigsten übersehen und ist die interessanteste. Technisch ist es dasselbe Muster wie MCP: die KI bekommt eine begrenzte Menge Funktionen und darf ausschließlich diese aufrufen. Nur sitzt nicht der Entwickler davor, sondern der Anwender, der wissen will, wie voll das Bad ist oder der einen Mangel melden möchte.

6.1 Was das für die Architektur bedeutet

- Die KI darf genau das, was der angemeldete Benutzer darf. Keine Sonderrechte, keine Ausnahmen.
- Jeder Werkzeugaufruf wird protokolliert, mit Benutzer und Zeitstempel.
- Aktionen mit Folgen, die sich nicht zurücknehmen lassen, sind gesperrt, unabhängig davon, wer danach fragt.
- Was die KI tut, muss der Anwender sehen können, nicht nur das Ergebnis.

7 Datenschutz, DSGVO und EU AI Act

Der Teil, der in der Praxis am häufigsten unterschätzt wird.

7.1 Die Fragen, die Sie vor dem ersten Prototyp klären

- **Wo werden die Daten verarbeitet?** Rechenzentrumsstandort, und ob ein Auftragsverarbeitungsvertrag vorliegt.
- **Werden die Daten zum Training verwendet?** Bei Geschäftskundenverträgen in der Regel nicht, bei kostenlosen Zugängen oft doch.
- **Welche Daten verlassen das Haus überhaupt?** Ein Schema ist etwas anderes als ein Datenbestand. Für Entwicklungsarbeit reicht meist das Schema.
- **Wie lange wird gespeichert?** Aufbewahrungsfristen des Anbieters gegen Ihre Löschkonzepte.

7.2 Was der EU AI Act für typische Access-Anwendungen bedeutet

Die meisten Anwendungen, um die es hier geht, fallen in die niedrigste Risikoklasse. Ein Co-Pilot, der Auskunft über Öffnungszeiten und Kursbelegungen gibt, ist kein Hochrisikosystem. Zwei Punkte gelten aber trotzdem:

- **Transparenzpflicht.** Der Anwender muss erkennen können, dass er mit einem KI-System spricht. Das ist billig umzusetzen und wird trotzdem oft vergessen.
- **KI-Kompetenz.** Wer KI-Systeme betreibt, muss dafür sorgen, dass die damit befassten Personen sie verstehen. Auch das ist eine Pflicht, keine Empfehlung.

Sobald Personalentscheidungen, Bonitätsbewertungen oder biometrische Daten ins Spiel kommen, ändert sich die Einordnung grundlegend. Dann gehört das geprüft, bevor programmiert wird.

7.3 Prompt Injection

Das praktische Sicherheitsthema. Ein Angreifer schreibt Anweisungen in Daten, die die KI später liest: in eine Mängelbeschreibung, einen Dateinamen, ein Notizfeld. Liest die KI das als Anweisung statt als Inhalt, führt sie fremde Befehle mit Ihren Rechten aus.

Gegenmittel, die tatsächlich helfen:

- Werkzeuge, die beliebigen Code ausführen, standardmäßig abschalten. Eine Bestätigungsabfrage hilft nicht, denn der eingeschleuste Text kann die Bestätigung gleich mitliefern.
- Die KI bekommt die Rechte des Benutzers, nicht mehr.
- Datenbanken aus unbekannter Quelle sind kein Spielplatz für einen Agenten.

8 Demo 3: Modernisierung, und warum sie über MCP läuft

Demo 1 war Neubau, Demo 2 Erweiterung. Beides ließe sich auch skripten. Modernisierung nicht: [was zu tun ist, weiß man erst, wenn man den Bestand gelesen hat](#). Die KI muss suchen, wo eine Funktion überall aufgerufen wird, bevor sie deren Signatur ändert, und danach prüfen, ob noch alles übersetzt. Das ist der Fall, für den MCP gebaut ist.

8.1 Was umgebaut wird

- **Bit-Fähigkeit:** Declare in einen #If VBA7-Block mit PtrSafe und LongPtr
- **Entkopplung:** keine MsgBox mehr in Fachfunktionen. Die Funktion liefert ein Ergebnis, das Formular zeigt es an
- **Absicherung:** parametrisierte Abfragen statt eingesetzter Werte
- **Disziplin:** Option Explicit, Typen, doppelte Validierung an einer Stelle

Was *nicht* passiert: die Fachlichkeit ändert sich nicht. Ein Refactoring, das die Fachlichkeit mitändert, ist kein Refactoring, sondern ein neues Projekt.

8.2 Der Erfolg ist gemessen, nicht behauptet

	Bestand	Modernisiert
Module ohne Option Explicit	4	0
Declare ohne PtrSafe	1	0
MsgBox in Fachmodulen	3	1
SetWarnings False	3	0
Werte per Verkettung in SQL	59	54
parametrisierte Abfragen	0	4
Variablen ohne Typ	25	3

Zwei Zahlen sind bewusst ehrlich statt schön. Die verbliebene MsgBox steckt im Verleihmodul aus Demo 2: die alten Funktionen bleiben stehen, daneben liegen entkoppelte Hüllen, damit sich beides vergleichen lässt. Und die Verkettungen in SQL sinken nur moderat, weil die restlichen numerische Schlüssel sind, keine Texte. Ein Refactoring, das Zahlen schön statt Risiken zu senken, wäre das falsche Vorbild.

8.3 Zwei Funde, die den Aufwand wert waren

8.3.1 Auch der neue Code von gestern ist morgen Bestand

Das Verleihmodul war beim Bau in Demo 2 sauber geschrieben: Option Explicit, Typen, parametrisierte Abfragen, ordentliche Fehlerbehandlung. Und es meldet Fehler per MsgBox. [Sauber geschrieben heißt nicht entkoppelt](#).

8.3.2 Ein Refactoring kann kaputtmachen, was es verbessern soll

Der erste Entwurf der modernisierten Buchungsfunktion lieferte einen benutzerdefinierten Typ zurück (`Public Type`). Das liest sich gut, ist aber über COM **nicht übertragbar**: die Funktion war damit von außen unaufzurufbar, also für Tests, ein Web-Frontend und jeden Agenten unbrauchbar. Genau das, was das Refactoring verbessern sollte, wäre kaputtgegangen.

Gefunden hat es die Zeile „Von außen aufrufbar“ im Vergleichsskript. Ohne diese Messung wäre der Fehler bis auf die Bühne durchgerutscht. Die Lehre ist allgemeiner als der Fall: **eine Verbesserung, die niemand misst, ist eine Behauptung.**

9 Stufe 3 zum Anfassen: dieselbe Anwendung als Weblösung

Die Badverwaltung gibt es zum Vergleich noch einmal als Weblösung, in zwei Betriebsarten. Der Unterschied liegt **ausschließlich in der Datenschicht**; die Oberfläche ist beide Male dieselbe Datei.

	Standalone	Portal
Datenbank	IndexedDB im Browser (NoSQL)	SQLite auf dem Server
Server	keiner	Python-Standardbibliothek
Anmeldung	nein	ja, drei Rollen
Geschäftsregeln	im Browser	auf dem Server
Protokoll	nein	jede Schreibaktion
Arbeitsplätze	einer	beliebig viele

9.1 Die eine Frage, die über die Migration entscheidet

Stellen Sie beide Fassungen nebeneinander und fragen Sie: **Wo stehen die Geschäftsregeln?**

In der Standalone-Fassung steht die Regel „eine verbrauchte Karte wird nicht erneut entwertet“ im Browser. Wer die Entwicklerwerkzeuge öffnet, hebt sie aus. Für einen Kiosk ohne Netz ist das in Ordnung. Für eine Kasse nicht.

In der Portalfassung steht dieselbe Regel auf dem Server. Der Browser darf fragen, entscheiden tut der Server. Erst dadurch lässt sich protokollieren, wer wann was getan hat, und erst diese Fassung ist die, über die man mit Blick auf DSGVO und EU AI Act überhaupt reden kann.

9.2 Und genau so ist es in Access

Regeln im Frontend-VBA verhalten sich wie Regeln im Browser: der Anwender hat die Datei. Wer nur das Backend nach SQL Server verschiebt, hat die Datenbank umgezogen und die Architektur behalten.

Die Migration beginnt deshalb nicht bei der Technik, sondern bei der Frage, welche Regel wohin gehört. Prüfung, Berechtigung und Protokoll gehören dorthin, wo der Anwender nicht hinkommt. Das ist Arbeit, bei der eine KI gut unterstützen kann; die Entscheidung bleibt Ihre.

10 Warum die Demo-Datenbank so aussieht

Die Demo-Datenbank enthält bewusst gewachsenen VBA-Code: kein `Option Explicit`, zusammengebautes SQL, dieselbe Prüfung an zwei Stellen getippt, verschluckte Fehler und eine `Declare`-Anweisung ohne `PtrSafe`, die unter 64-Bit-Access sofort bricht. Das ist Absicht. Es ist das Material, an dem sich Refactoring und die Bit-Umstellung überhaupt zeigen lassen.

11 Kontakt

Björn Fox

Solution Architect

Avantgarde Business Solutions GmbH, Neuss

Schwerpunkte: Entwicklung und Integration von KI-Lösungen im Unternehmenskontext, ERP-Projekte mit Sage 100, Warenwirtschaft, Produktion und Finanzwesen. Treiber des [Avantgarde Co-Pilot](#).

Website	www.avantgar.de
Avantgarde Co-Pilot	ava-co-pilot.de
LinkedIn	linkedin.com/company/avantgarde-business-solutions-gmbh
YouTube	youtube.com/@AvantgardeBusinessSolutions
KI-Club	Unverbindlich vormerken lassen (Microsoft Forms)