



WHZ Westsächsische
Hochschule Zwickau
Hochschule für Mobilität

Konzeption und Umsetzung eines Major- Upgrades einer Webapplikation mit Unterstützung eines Large Language Models

Fakultät für Physikalische Technik und Informatik
Fachgruppe Informatik

Bachelorarbeit, vorgelegt von

Lukas Spitzer

im April 2026

Betreuer: Prof. Dr. Wolfgang Golubski, Westsächsische Hochschule Zwickau

Einrichtung: N+P Informationssysteme GmbH, Meerane

Danksagung

Ich danke Prof. Dr. Wolfgang Golubski für die Betreuung dieser Bachelorarbeit seitens der Westsächsische Hochschule Zwickau.

Zudem danke ich Michael Kusche und Florian Kunze für die Betreuung dieser Bachelorarbeit seitens der N+P Informationssysteme GmbH.

Zusammenfassung

Diese Bachelorarbeit untersucht die Konzeption und Umsetzung eines Major-Upgrades einer bestehenden Webapplikation unter Einsatz eines LLM. Ziel war es, zu analysieren, inwiefern ein LLM den Migrationsprozess einer historisch gewachsenen Frontend-Architektur, konkret von DevExtreme 18.2.18 mit SPA-Framework (Knockout.js) hin zu DevExtreme 25.2.5 und React unterstützen und beschleunigen kann.

Zunächst wurden die grundlegenden Konzepte klassischer Webanwendungen mit TypeScript und imperativer DOM-Steuerung sowie moderner, deklarativer Ansätze mit React erläutert. Darauf aufbauend wurden die spezifischen Herausforderungen der Migration sowie Anforderungen an ein unterstützendes LLM definiert. Anschließend erfolgte eine detaillierte Analyse der technologischen Veränderungen zwischen den DevExtreme-Versionen sowie die Entwicklung eines strukturierten Vorgehensmodells für die Migration unter Einsatz des LLM Claude.

Im weiteren Verlauf wurde eine prototypische Zielanwendung schrittweise aufgebaut, wobei das LLM zur Generierung und Transformation von Code eingesetzt wurde. Dabei kamen zusätzliche Konzepte wie die Kontextualisierung über eine projektspezifische Wissensbasis, der Einsatz spezialisierter Agenten sowie ein planungsorientiertes Plugin zur strukturierten Problemlösung zum Einsatz.

Die Evaluation erfolgte anhand der Migration und Überprüfung mehrerer Gridseiten, bei denen Funktionalität, Vollständigkeit und Korrektheit mit dem Altsystem verglichen wurden. Dabei wurde untersucht, ob bestehende Features korrekt übernommen, verändert oder neu interpretiert wurden. Die Ergebnisse zeigen, dass ein LLM grundsätzlich in der Lage ist, komplexe Migrationsaufgaben zu unterstützen und funktionierende Zielstrukturen zu erzeugen. Grundlegende Funktionalitäten wie Datenverarbeitung, Routing und Zustandsmanagement konnten erfolgreich umgesetzt werden. Gleichzeitig traten jedoch auch Einschränkungen auf, wie unvollständige UI-Übernahmen, fehlerhafte oder erfundene Funktionalitäten sowie Probleme bei komplexeren Zusammenhängen. Als zentrale Grenze der automatisierten Migration wurden die inhaltliche Komplexität der Anwendung, die Notwendigkeit manueller Nachbearbeitung sowie die zeitliche Begrenzung des Projekts identifiziert.

Inhaltsverzeichnis

Danksagung	2
Zusammenfassung	3
Abkürzungsverzeichnis	7
Abbildungsverzeichnis	8
1 Einleitung	9
1.1 Motivation	9
1.2 Forschungsfrage	9
2 Theoretische Grundlagen	10
2.1 Aufbau von Webanwendungen mit TypeScript	10
2.1.1 DOM und imperative UI-Steuerung	10
2.1.2 SPA-Framework	10
2.1.3 Routing und Navigation	10
2.2 Aufbau von Webanwendungen mit React	11
2.2.1 Deklarativer UI-Ansatz	11
2.2.2 Komponenten und Renderbaum	11
2.2.3 Zustandsverwaltung	11
2.2.4 Event Handling	12
2.2.5 Navigation und Ressourcen	12
2.3 Stand der Forschung	12
3 Probleme und Anforderungen	13
3.1 Problemstellung	13
3.2 Anforderung an das LLM	13
4 Analyse DevExtreme Versionen	14
4.1 Funktionale und technologische Änderungen	14
4.1.1 DevExtreme 18.2.18 → 20.2	14
4.1.2 DevExtreme 20.2 → 21.2	14
4.1.3 DevExtreme 21.2 → 23.2	15
4.1.4 DevExtreme 23.2 → 25.2.5	15

5	Vorgehensmodell	17
5.1	Grundlegendes Vorgehensmodell	17
5.2	Kontextualisierung des LLM	17
5.2.1	Projektkontext und Systemverständnis	17
5.2.2	Definition von Architekturregeln	18
5.2.3	Arbeitsanweisung und Entwicklungsprozesse	18
5.2.4	Build- und Integrationwissen.....	18
5.2.5	Einsatz von (Sub)-Agenten.....	18
5.2.6	Dokumentation von Fehlerquellen	18
5.3	Agenten.....	19
5.3.1	Aufbau eines Agenten	19
5.3.2	Einordnung im Projektkontext.....	20
5.4	Einsatz des Superpower-Plugins	20
5.4.1	Funktionsweise des Plugins	20
5.5	Erstellen der Grundanwendung	21
5.5.1	Beispiel 1: Layout und UI-Grundlagen	22
5.5.2	Beispiel 2: Grid-Wrapper	25
5.5.3	Beispiel 3: Gridseite Operation Detail	28
5.6	Probleme und Herausforderungen.....	30
6	Evaluation	31
6.1	Testen der Gridseiten	31
6.2	Ergebnisse der Tests.....	38
7	Interpretation und Ausblick	39
7.1	Grenzen der Evaluation	39
7.2	Ausblick.....	39
	Literaturverzeichnis	40
	Anhang.....	45
	Generierter Code im Testsystem.....	48
	CLAUDE.md	53

Agent frontend-specialist	57
Erklärung zur Verwendung maschineller Hilfsmittel	59
Selbstständigkeitserklärung.....	60

Abkürzungsverzeichnis

DOM Document Object Model

LLM Large Language Model

SPA Single-Page-Application

Abbildungsverzeichnis

Abbildung 6.1 Maschinenübersicht Vorher	31
Abbildung 6.2: Maschinenübersicht Nachher.....	32
Abbildung 6.3: React Nachher 1 Minute später	33
Abbildung 6.4: TypeScript Vorher	34
Abbildung 6.5: React Danach	34
Abbildung 6.6: React Danach bearbeitet	35
Abbildung 6.7: Maschinendetail Vorher	36
Abbildung 6.8: React Nachher.....	36
Abbildung 6.9: React Danach bearbeitet	37

1 Einleitung

1.1 Motivation

Viele Unternehmen, auch die Firma N+P Informationssysteme GmbH, nutzt moderne Web-Applikationen basierend auf UI-Frameworks. Oft bestehen diese Systeme in den Unternehmen über Jahre auf dem gleichen technologischen Stand, d.h. Updates auf aktuelle Versionen werden nicht durchgeführt. Da in neuen Versionen aber nicht nur neue Funktionalitäten entwickelt wurden, sondern auch Performance-Optimierungen und sicherheitsrelevante Aktualisierungen bereitgestellt werden, kann dies über Jahre zu Sicherheitsrisiken und Kompatibilitätsproblemen in den Unternehmen führen.[1] Speziell betrachten wir hier die im Unternehmen N+P Informationssysteme GmbH genutzte Java-Script-Bibliothek DevExtreme in der veralteten Version 18.2.18 aus dem Jahr 2018. Jedes Jahr gibt es 2 neue Major-Releases [1], im März 2026 erschien die Version 25.2.5, auf die jetzt umgestellt werden soll, zusätzlich soll eine Überführung vom DevExtreme SPA Framework (mit Knockout.js) auf React umgesetzt werden. Somit liegen zwischen diesen beiden Versionen mehrere Major-Releases mit zahlreichen Breaking Changes. Ein direktes Upgrade ist aufgrund möglicher Kompatibilitätsbrüche, veränderter APIs, geänderter Build-Prozesse sowie neuer Framework-Integrationen nicht trivial. Auf der Homepage von DevExtreme kann als maximales Update von der Version 18.2.18 auf die Version 23.2.15 umgestellt werden [2], außerdem zeigt die lange Liste der Änderungen in den Versionen die Komplexität dieser Umstellung. Unternehmen stellt dies somit vor technische, organisatorische und ressourcenbezogene Herausforderungen. Somit stellt sich die Frage, wie ein LLM bei der technologischen Modernisierung in Web-Frontend-Architekturen unterstützen kann.

1.2 Forschungsfrage

Aus dem vorherigen Abschnitt ergeben sich folgende Forschungsfragen:

Kann Claude bei der Überführung vom DevExtreme SPA Framework auf React unterstützen?

Kann Claude bei der Überführung von DevExtreme 18.2.18 auf 25.2.5 unterstützen?

Welche Nebenwirkungen treten auf und wo liegen die Grenzen der Migration mit Claude?

2 Theoretische Grundlagen

Das Kapitel 2.1. vermittelt die Grundlagen für die bei der N+P Informationssysteme GmbH aktuell eingesetzte Version der Java-Script-Bibliothek DevExtreme. Im Kapitel 2.2 wird der Aufbau von React behandelt in welches das System überführt werden soll.

2.1 Aufbau von Webanwendungen mit TypeScript

2.1.1 DOM und imperative UI-Steuerung

Das DOM stellt die Struktur einer Website als Baum dar, in dem jedes HTML-Element als Knoten vorhanden ist. In klassischen JavaScript und TypeScript Anwendungen wird dieses DOM direkt manipuliert. [3]

Die Benutzeroberfläche entsteht dabei durch gezielte Änderungen am DOM, beispielsweise durch das Ausblenden von Elementen oder das Aktualisieren von Inhalten. Dieses Vorgehen wird als imperative Programmierung bezeichnet, da konkret beschrieben wird, welche Schritte ausgeführt werden müssen, um einen Zustand der Oberfläche zu erreichen. [3]

2.1.2 SPA-Framework

Ein SPA Framework ermöglicht es Inhalte dynamisch innerhalb einer Seite zu laden, ohne dass ein vollständiger Seitenwechsel erfolgt. Dabei wird häufig mit Platzhaltern gearbeitet, die zur Laufzeit durch Inhalte ersetzt werden. [4], [5]

Die Steuerung erfolgt intern durch das Framework, welches, basierend auf bestimmten Regeln, entscheidet welche Inhalte geladen und angezeigt werde. Die tatsächliche Struktur der Oberfläche ergibt sich somit erst während der Laufzeit. [4]

2.1.3 Routing und Navigation

Routing beschreibt die Zuordnung von URLs zu bestimmten Inhalten einer Anwendung. In SPA-Ansätzen wird dies durch das Framework selbst gesteuert. Dabei wird die aktuelle URL analysiert und daraus abgeleitet, welche Inhalte dargestellt werden sollen. Die Navigation erfolgt innerhalb der Seite, ohne dass ein vollständiger Seiten-Reload notwendig ist. [6]

2.2 Aufbau von Webanwendungen mit React

2.2.1 Deklarativer UI-Ansatz

React basiert im Gegensatz zu imperativen DOM-Manipulation auf einem deklarativen Ansatz. Dabei wird beschrieben, wie die Benutzeroberfläche für einen bestimmten Zustand aussehen soll, anstatt die einzelnen Schritte zur Aktualisierung der Oberfläche vorzugeben. Die Oberfläche wird somit als Funktion des Zustandes betrachtet. Änderungen am Zustand führen automatisch dazu, dass die betroffenen Teile der Oberfläche neu berechnet und aktualisiert werden. [7], [8], [9]

2.2.2 Komponenten und Renderbaum

Die Benutzeroberfläche wird aus sogenannten Komponenten aufgebaut, die eine hierarchische Struktur bilden. Eine Komponente ist eine wiederverwendbare Einheit, die einen Teil der Oberfläche beschreibt. Diese Komponenten entsprechen in ihrer Funktion einzelnen UI-Elementen oder Containern. [10]

Die Definition erfolgt mithilfe von JSX, einer Syntaxerweiterung, die HTML-ähnliche Strukturen innerhalb von JavaScript bzw. TypeScript erlaubt. [11]

DevExtreme stellt für React spezielle Wrapper-Komponenten bereit, die direkt in JSX verwendet werden können. Diese Komponenten bilden das Gegenstück zu klassischen UI-Elementen und ermöglichen eine deklarative Beschreibung komplexer Oberflächen. [12]

Eigenschaften wie Größe, Layout oder Verhalten werden über Props definiert, die beim Aufruf der Komponente übergeben werden. [13]

2.2.3 Zustandsverwaltung

Ein zentrales Konzept in React ist der sogenannte State. States werden innerhalb von Komponenten definiert und können sich im Laufe der Laufzeit ändern. [14]

Änderungen am State führen automatisch zu einer Aktualisierung der Benutzeroberfläche.

Dadurch entsteht eine Kopplung zwischen State und Display. [15]

States können mithilfe von Hooks verwaltet werden. [18]

Hooks sind Funktionen in React, mit denen Komponenten Zustände und andere Funktionen wie Lebenszyklus-Logiken nutzen können, ohne Klassen zu verwenden. [16], [17]

Im Gegensatz zur imperativen DOM-Manipulation wird der Zusammenhang zwischen Zustand und UI direkt im Code beschrieben. [7]

2.2.4 Event Handling

Auch in React erfolgt die Verarbeitung von Benutzerinteraktionen über events. Diese werden jedoch deklarativ an Komponenten gebunden. [18]

Event Handler werden als Funktionen definiert und über Props an Komponenten übergeben.

Beim Auftreten eines events wird die entsprechende Funktion ausgeführt. [19], [20], [21]

DevExtreme-Komponenten unterstützen dieses Modell, indem sie events als Props bereitstellen, die mit Callback-Funktionen verknüpft werden können. [22]

2.2.5 Navigation und Ressourcen

Die Navigation innerhalb einer React-Anwendung erfolgt typischerweise über Bibliotheken wie React Router, die eine clientseitige Navigation ohne Neuladen der Seite ermöglichen. [23]

Ressourcen wie Stylesheets oder Bilder werden analog zu TypeScript-Anwendungen über Imports eingebunden. Zusätzlich bietet DevExtreme vordefinierte Themes zur Gestaltung der Benutzeroberfläche. [24], [25], [26]

2.3 Stand der Forschung

Im Rahmen der Literaturrecherche konnten keine wissenschaftliche Publikationen gefunden werden, die eine Migration von einer DevExtreme auf eine höhere Version konzipiert oder implementiert. Ebenso betrifft dies die Überführung vom DevExtreme SPA Framework auf React.

3 Probleme und Anforderungen

3.1 Problemstellung

Das zentrale Problem der vorliegenden Arbeit besteht in dem erheblichen Aufwand, der mit der Migration der DevExtreme Version 18.2.18 zu 25.2.5 und der Umstellung vom DevExtreme SPA Framework zu React verbunden ist. Aufgrund der großen Anzahl an Dateien und einer historisch gewachsenen Codebasis wurde der Migrationsaufwand seitens der Firma N+P Informationssysteme GmbH auf etwa ein Jahr geschätzt.

Mit dem Ziel, die in Kapitel 1.2 gestellten Forschungsfragen zu beantworten, wird im Rahmen dieser Bachelorarbeit das LLM Claude Opus 4.6 und 4.7 für die Migration des NuPMES anhand des Monitors getestet. NuPMES ist eine Eigenentwicklung der Firma N+P Informationssysteme GmbH, welches ein MES ist und eine Schnittstelle zum ERP-System AP+ hat. Das NuPMES liefert Informationen zu Betriebs-, Maschinen- und Geschäftsdaten, für weitere Informationen siehe [27]. Dabei wird in dieser Arbeit nur der Monitor, d.h. die Anwendung für die Angestellten und nicht das Terminal, die Anwendung für die gewerblichen Arbeitnehmer (Werker) untersucht. [27], [28], [29]

Somit ergeben sich folgende Anforderungen an das LLM für die Migration des Monitors.

3.2 Anforderung an das LLM

Der Einsatz eines LLM soll den Migrationsprozess unterstützen und beschleunigen.

Das LLM muss in der Lage sein folgende Funktionalitäten korrekt umzusetzen:

- Alle Felder haben dieselben Eigenschaften wie vorher
- Alle UI-Elemente sind vollständig vorhanden und sichtbar
- Grid-Funktionalitäten bleiben erhalten
- Abhängigkeiten zwischen Feldern bleiben erhalten
- Navigation und Routing entspricht exakt der alten Anwendung
- Daten werden korrekt geladen, aktualisiert und gespeichert
- Berechtigungen wirken sich korrekt auf Sichtbarkeit und Zugriff aus
- Lade- und Fehlzustände werden korrekt angezeigt

4 Analyse DevExtreme Versionen

Dieses Kapitel analysiert die wichtigsten funktionalen, technologischen und konzeptionellen Änderungen von DevExtreme zwischen den Versionen 18.2.18 und 25.2.5. Der Fokus liegt bewusst auf Hauptversions-Releases, da DevExpress zwei Hauptversionen pro Jahr veröffentlicht und sich in der Vergangenheit auf neue Komponenten, API-Überarbeitungen und Fehlerbehebungen in diesen Releases konzentriert hat. [1] Es werden nur Änderungen berücksichtigt, die für Projekte relevant sind, die DevExtreme ohne einen Framework-Wrapper nutzen, d.h. direkt aus JavaScript, TypeScript und CSS über das DevExtreme-Core Paket berücksichtigt.

Die Analyse ist in zwei Teile gegliedert: Zunächst werden die funktionalen und technologischen Neuerungen beschrieben, anschließend werden Breaking Changes und Deprecations systematisch dargestellt und kurz erläutert.

4.1 Funktionale und technologische Änderungen

Zur besseren Einordnung der technischen Änderungen wird der Migrationspfad zusätzlich in vier klar abgegrenzte Phasen unterteilt.

4.1.1 DevExtreme 18.2.18 → 20.2

In dieser Phase stehen Leistungsoptimierungen an den Komponenten für große Datenmengen sowie Vorarbeiten an den Theming- und Modulsystemen im Vordergrund.

Bereits in v18.1 führte DevExpress die aktuelle Generation des virtuellen Scrollens für DataGrid und TreeList ein: Das Grid rendert nur Zeilen, die sich innerhalb des Ansichtsbereichs befinden, entfernt beim Scrollen Spalten außerhalb des Bildschirms aus dem DOM und wechselt von synchronem zu asynchronem Zeilen-Rendering, wenn die Komplexität des Markups andernfalls zu Frame-Einbrüchen führen würde. Diese Mechanismen wurden in den Versionsreihen 19.x und 20.x erweitert und stabilisiert. [30], [31]

Der Satz vordefinierter Themes (Generic, Generic Compact, Material) und der ThemeBuilder existierten bereits in 18.2, doch die Konfiguration erfolgte über statische LESS/SCSS-Strukturen. Während der 20.x-Reihe modernisierte DevExpress die Theme-Paketierung weiter, was später die für Version 25.2 dokumentierten, auf CSS-Variablen basierenden, ThemeBuilder-Abläufe ermöglichte. [32], [33]

4.1.2 DevExtreme 20.2 → 21.2

In dieser Phase liegt der Schwerpunkt auf der TypeScript Unterstützung und der ersten Welle der Modernisierung des visuellen Designs.

Ab der 21.x-Reihe veröffentlicht DevExtreme komponentenbezogene event- und Argumenttypen als benannte Exporte aus jedem Widget-Modul. Zum Beispiel `import { ContentReadyEvent, ContextMenuPreparingEvent } from „devextreme/ui/data_grid“`. Dies ersetzt die frühere Abhängigkeit von allgemeinen *DevExpress.**-Deklarationen und ermöglicht eine strikte Typisierung von event handling Routinen in reinen TypeScript Projekten. [34], [35]

Mit Version 21.1 wurde das Fluent-Design neben den bestehenden Designs Generic, Generic Compact und Material eingeführt, wodurch Nicht-Wrapper-Projekte ein modernes Standard-Erscheinungsbild erhalten, das nicht von einem Framework abhängig ist. [36]

Mit Version 21.1 wurde außerdem das horizontale virtuelle Scrollen in den Scheduler integriert. Dies ist derselbe, ausschließlich auf den Viewport bezogene Rendering-Ansatz, der bereits vom DataGrid verwendet wird, was für Anwendungen wichtig ist, die den Scheduler in zeitintensive Ansichten der Fertigungsebene einbetten. [36]

4.1.3 DevExtreme 21.2 → 23.2

Diese Phase ist geprägt von Arbeiten zur Barrierefreiheit, der fortgesetzten Abkündigung veralteter APIs und der weiteren Modularisierung der verteilbaren Bundles.

Mit Version 23.1 wurden die *aria-** Attribute in allen UI-Komponenten umfassend überarbeitet und das Verhalten deaktivierter Elemente an den WAI-ARIA-Standard angepasst, sodass Screenreader deaktivierte Elemente bei Fokus melden. Version 23.2 hat dies speziell auf DataGrid ausgeweitet. Alle von den Barrierefreiheitsprüfern (Axe, WAVE, Lighthouse) gemeldeten Probleme wurden behoben, die Tastaturnavigation innerhalb adaptiver Zeilen wurde verbessert und Screenreader-Ansagen für geänderte Zellen und gelöschte Zeilen im Batch-Bearbeitungsmodus wurden hinzugefügt. Probleme mit dem Farbkontrast im DataGrid wurden in derselben Version behoben. [37], [38], [39]

APIs, die in früheren Versionen als veraltet markiert wurden, werden in dieser Phase aktiv ersetzt oder entfernt. Konkrete Codeanpassungen sind daher erforderlich und nicht optional. [40] Die für *devextreme/ui/**, *devextreme/data/** und *devextreme/common/** dokumentierten Einstiegspunkte pro Modul bilden die Grundlage für die 23.x-Bundles; ES-Modul-Importe (`import { CustomStore } from „devextreme/common/data“`) werden neben CommonJS vollständig unterstützt. [44]

4.1.4 DevExtreme 23.2 → 25.2.5

Zu diesem Zeitpunkt entspricht die Bibliothek den modernen Konventionen für ES-Module und TypeScript, und der Schwerpunkt der Releases verlagert sich auf strengere Typisierung und weitere Optimierungen des Rendering-Pfads.

Mit Version 24.2 wurden benannte Konfigurationskomponenten eingeführt, die streng typisiert sind und deren Geltungsbereich auf ihre übergeordnete UI-Komponente beschränkt ist, sodass

IntelliSense nur die Eigenschaften vorschlägt, die für diesen verschachtelten Kontext tatsächlich gültig sind, und der Compiler ungültige Kombinationen ablehnt. Ab dieser Version kann die gesamte öffentliche API eines Framework-Wrappers über einen einzigen Paket-Einstiegspunkt importiert werden. [41], [42]

Version 25.2 fügt eine gezielte Optimierung der Neu-Rendering-Vorgänge für *Accordion*, *Tabs*, *TabPanel* und *List* hinzu. Wenn *repaintChangesOnly* aktiviert ist, werden nach Neuordnungsvorgängen nur die betroffenen Elemente neu gerendert, was die DOM-Aktualisierungen bei großen Sammlungen deutlich reduziert. DataGrid und TreeList stützen sich weiterhin auf die zuvor etablierten Mechanismen für virtuelles Scrollen. [43]

ThemeBuilder in Version 25.2 ist vollständig auf CSS-Variablen ausgerichtet und enthält CLI-Tools (*build-theme*, *export-theme-vars*, *export-theme-meta*) zur Erstellung von Themes aus Metadatendateien. Dies entspricht dem Tooling-Standard, den ein Nicht-Wrapper-Projekt für benutzerdefiniertes Theming verwenden sollte. [32]

5 Vorgehensmodell

Dieses Kapitel beschreibt das Vorgehen zur Durchführung des Major-Upgrades sowie die Untersuchung des Einsatzes eines LLM im Migrationsprozess.

5.1 Grundlegendes Vorgehensmodell

Anstatt die bestehende Anwendung vollständig auf einmal zu migrieren, wurde zunächst eine Grundanwendung (Baseline) erstellt, die als Ausgangspunkt für alle weiteren Schritte dient. Diese Grundanwendung und alle weiteren Gridseiten wurden mithilfe von Claude Opus 4.6 und Claude Opus 4.7 generiert und anschließend schrittweise erweitert.

Das Vorgehen lässt sich wie folgt zusammenfassen:

1. Erstellen einer minimalen Grundanwendung
2. Schrittweises Erweitern um einzelne Gridseiten
3. Sukzessive Integration komplexere Komponenten und Logiken

5.2 Kontextualisierung des LLM

Die Datei CLAUDE.md stellt eine zentrale Komponente beim Einsatz von Claude Code dar. Sie dient als projektspezifische Wissensbasis, die vom Modell zu Beginn jeder Sitzung eingelesen wird und somit dauerhaft Kontext bereitstellt. Dadurch kann das LLM konsistent mit der Architektur, den Regeln und den Workflows eines Projekts arbeiten. Laut der offiziellen Dokumentation wird diese Datei genutzt, um Coding-Standards, Workflows und Projektarchitektur festzulegen. Ohne diese Kontextualisierung wäre das Modell gezwungen, Annahmen zu treffen, was die Fehleranfälligkeit deutlich erhöht. [45], [46]

Im Folgenden werden die wichtigsten Bestandteile der vorliegenden CLAUDE.md anhand konkreter Beispiele erläutert. Die Erläuterungen zur Erstellung der CLAUDE.md und alle Aussagen in diesem Kapitel sind den Quellen [45] und [46] entnommen. Die CLAUDE.md ist mit ihren Aussagen im Anhang zu finden.

5.2.1 Projektkontext und Systemverständnis

Die folgenden Informationen wurden an Claude übergeben:

„NuPMES is a browser-based Manufacturing Execution System (MES)...“

Zusätzlich werden zwei Anwendungen unterschieden:

„Monitor“ (DevExtreme-basiert) und „Terminal“ (Touch-Interface)

Diese Kontextinformationen dienen dazu, dem Modell ein einheitliches Verständnis der Systemarchitektur sowie der unterschiedlichen Benutzeroberflächen zu vermitteln. Dadurch kann Claude Code projektspezifische Anforderungen besser einordnen und zwischen Verwaltungsoberflächen und vereinfachten Produktionsinterfaces unterscheiden. Ohne diese Kontextualisierung bestünde das Risiko, dass generische oder nicht zur Architektur passende UI- bzw. Code-Strukturen erzeugt werden.

5.2.2 Definition von Architekturregeln

Ein weiterer Abschnitt ist das sogenannte Manager Pattern:

„Always create managers via UnitOfWork“

„**✗** WRONG – var itemManager = new ItemManager();“

Diese Regeln definieren verbindliche Strukturen für die Instanziierung und Nutzung von Komponenten innerhalb der Anwendung. Sie stellen sicher, dass das Modell keine generischen Standardlösungen generiert, die nicht mit der bestehenden Architektur kompatibel sind. CLAUDE.md dient hier als Mechanismus zur Durchsetzung architektureller Konsistenz innerhalb des generierten Codes.

5.2.3 Arbeitsanweisung und Entwicklungsprozesse

Die Datei enthält zudem konkrete Workflows, beispielsweise:

„Load ticket → analyze & plan → discuss plan with user → create task structure → execute with agents → resume later if needed.“

Diese strukturierte Reihenfolge unterstützt das LLM dabei, Aufgaben nicht isoliert, sondern systematisch umzusetzen. Claude Code kann eigenständig Features entwickeln und Code verändern, benötigt dafür jedoch klare Prozessvorgaben, um konsistente Ergebnisse zu liefern.

5.2.4 Build- und Integrationwissen

Weitere Bestandteile sind konkrete Build- und Entwicklungsanweisungen, z. B.:

„npm run dev“ und

„dotnet build NuPMESServer.sln“,

sowie der Hinweis auf die automatische Generierung von Schnittstellen (IPC):

„Generates TypeScript interfaces from .NET contracts“.

Diese Informationen ermöglichen es dem LLM, nicht nur Code zu erzeugen, sondern diesen korrekt in den technischen Gesamtkontext einzuordnen. Ohne dieses Wissen könnten z. B. falsche Schnittstellen oder inkonsistente Typdefinitionen entstehen.

5.2.5 Einsatz von (Sub)-Agenten

Die CLAUDE.md Datei definiert außerdem fünf Agenten mit den zugeordneten Aufgabenbereichen.

Dadurch wird auch der Kontext isoliert gehalten, was dazu führt, dass Informationen aus früheren Aufgaben nicht zukünftige Aufgaben beeinflussen können. [47]

Nähere Informationen und Erläuterung zu den Agenten folgen in Kapitel 5.3.

5.2.6 Dokumentation von Fehlerquellen

Ein weiterer Abschnitt sind Hinweise auf Fehlerquellen und Einschränkungen:

„Never use new for managers“

„DevExtreme version: 18.2.18“

Diese sogenannten „Common Pitfalls“ adressieren typische Fehler, die sowohl Entwickler als auch LLMs machen könnten. Durch diese expliziten Hinweise wird das Risiko fehlerhafter Codegenerierung reduziert und die Qualität der Ergebnisse verbessert.

5.3 Agenten

Im Kontext moderner LLM beschreibt der Begriff Agent eine spezialisierte Instanz des Modells, die für klar definierte Aufgabenbereiche konfiguriert ist. Anstatt ein einzelnes Modell universell für alle Aufgaben zu verwenden, werden mehrere Agenten mit spezifischen Rollen, Anweisungen und Verantwortlichkeiten eingesetzt. Dadurch werden komplexe Aufgabe in kleinere Teilaufgabe zerlegt.

Im Beispiel des NuPMES wurden 5 Agenten definiert:

- frontend-specialist (siehe Anhang),
- backend-csharp-developer,
- code-reviewer,
- fullstack-feature-developer,
- qa-expert

Alle Agenten wurden gemäß nachfolgender Quelle aufgebaut: [47]

5.3.1 Aufbau eines Agenten

Im Folgenden wird der Aufbau eines Agenten anhand des Agenten „frontend-specialist“ erklärt. Ein Agent besteht im Wesentlichen aus einer strukturierten Beschreibung seiner Rolle, seines Kontexts sowie konkreter Arbeitsanweisungen.

1. Rollenbeschreibung:

„You are a frontend specialist focused on DevExtreme, TypeScript, and modern frontend development.“

2. Kontext und Technologien:

„The project uses DevExtreme 18.2.18 with Knockout.js and is being migrated to React and TypeScript.“

3. Aufgabenbeschreibung:

„Your tasks include: - Migrating existing UI components
- Creating new React components
- Improving frontend structure“

4. Regeln und Einschränkung:

„Do not introduce new frameworks. Follow existing architecture patterns. Prefer small, incremental changes.“

5. Erwartetes Ausgabeformat

„Always provide:

- Explanation of changes
- Clean and structured code
- Step-by-step reasoning“

5.3.2 Einordnung im Projektkontext

Durch die Verwendung spezialisierter Agenten konnte im Projekt eine Trennung von Verantwortlichkeiten erreicht werden. Beispielsweise wurde die Migration von UI-Komponenten durch den „frontend-specialist“ durchgeführt, während Backend-Logik durch den entsprechenden Backend-Agenten bearbeitet wurde. Anschließend erfolgte eine Qualitätssicherung durch einen „code-reviewer“ Agenten.

5.4 Einsatz des Superpower-Plugins

Neben der Nutzung spezialisierter Agenten wurde im Projekt zusätzlich das Plugin „Superpowers“ innerhalb von Claude Code eingesetzt. Grundidee des Plugins ist es, nicht direkt Code zu generieren, sondern zunächst eine strukturierte Problemanalyse und Planung durchzuführen.

5.4.1 Funktionsweise des Plugins

Das Superpowers-Plugin folgt einem mehrstufigen Vorgehen:

1. Analysephase:

Zunächst wird das Problem analysiert und in Teilaspekte zerlegt. Dabei werden mögliche Lösungsansätze identifiziert und strukturiert beschrieben.

2. Erstellung einer Plan-Datei:

Die Ergebnisse der Analyse werden in einer Plan-Datei dokumentiert. Diese enthält:

- Beschreibung des Problems
- Zieldefinition
- geplante Umsetzungsschritte

3. Erstellung einer Spezifikation (Spec-Datei)

Auf Basis der Plan-Datei erzeugt das Plugin eine strukturierte Spezifikation. Diese folgt einem festen Schema:

- Problem
- Impact
- Fix

4. Automatische Umsetzung

Nach Freigabe der Spezifikation wird die Umsetzung durch das LLM durchgeführt. Dabei orientiert sich das Modell an der zuvor definierten Struktur.

5.5 Erstellen der Grundanwendung

Um jederzeit einen Vergleich zu haben und die Ergebnisse auswerten zu können, wurde die neue React-Anwendung in einem neuen Verzeichnis erstellt. Nachdem die Erstellung der ersten Grundanwendung durch Claude scheiterte, wurde die Erstellung der Grundanwendung in kleinere Schritte unterteilt.

Die wichtigsten Schritte lassen sich wie folgt zusammenfassen:

1. Initiales Grundgerüst: Aufbau einer neuen React-Anwendung parallel zur bestehenden Struktur
2. Layout und UI-Grundlagen: Erstellung eines zentralen Layouts mit Sidebar, Header und grundlegenden UI-Komponenten
3. Theming und Designsystem: Umstellung auf DevExtreme 25.2., Entfernung individueller Styles zugunsten eines einheitlichen Themas
4. Grundlegende Infrastruktur: Typsicheres System, Notification-System, Authentifizierungslogik
5. Service- und Datenebene: Einführung einer Service-Schicht, Verbesserungen an Grid-Komponenten, Stabilisierung der Authentifizierung

Die folgenden Beispiele zeigen zunächst einen Ausschnitt aus dem Aufbau der Grundanwendungen zwischen HTML und React, dann folgt ein Vergleich zwischen 2 TypeScript Dateien anhand des Grid-Wrappers und zum Schluss folgt ein Beispiel zwischen TypeScript und React anhand der Gridseite OperationDetail.

5.5.1 Beispiel 1: Layout und UI-Grundlagen

Das Layout des Altsystems basiert auf einem statischen HTML-Template:

```
<div class="nupDesktop-layout" data-options="dxLayout : { name: 'nupDesk-top'
}>
  <header id="header">
    <div id="showSideDrawerButton" data-bind="dxButton: { onClick:
Nup.viewManager.showSidedrawer }"></div>
    <h3 data-bind="text: Nup.viewManager.currentTitle"></h3>
  </header>

  <div id="sidedrawer"></div>

  <div id="content-wrapper">
    <div id="buttonToolbar" data-bind="dxToolbar:
Nup.toolbarViewModel.toolbarOptions"></div>
    <div data-options="dxContentPlaceholder : { name: 'content' }"></div>
  </div>
</div>
```

Auszug aus NupDesktop.html im Altsystem

Das von Claude Code generierte Monitor Layout mit React:

```
export function MonitorLayout(): React.ReactElement {
  const isNarrow = useMediaQuery('(max-width: 1024px)');
  const [sidebarOpen, setSidebarOpen] = useState(!isNarrow);
  return (
    <div>
      <NpLoadingBar />
      <Header onMenuToggle={() => /*...*/} />
      <div>
        <Drawer
          opened={sidebarOpen}
          openedStateMode={isNarrow ? 'overlap' : 'shrink'}
          render={() => <Sidebar onNavigate={handleNavigate} />}>
          <main>
            <Suspense fallback={<NpSuspenseLoading />}>
              <Outlet />
            </Suspense>
          </main>
        </Drawer>
      </div>
    </div>
  );
}
```

Auszug aus MonitorLayout.tsx im Testsystem

Ein Unterschied zwischen dem Altsystem und dem Testsystem liegt im Umgang mit Zuständen und globaler Logik. Im Altsystem wird auf globale Objekte wie *Nup.viewManager* oder *Nup.userManager* zugegriffen.

Im Testsystem ist dieses Muster nicht mehr vorhanden. Stattdessen wird Zustand lokal innerhalb von Komponenten über *useState* und über dedizierte Stores verwaltet. [48]

Ein weiterer Unterschied betrifft das Rendering von Inhalten. In dem Altsystem wird der Hauptinhalt über das DevExtreme SPA Framework gesteuert. Das Element

```
<div data-options="dxContentPlaceholder : { name: 'content' }"></div> [4]
```

fungiert dabei als Platzhalter, der zur Laufzeit vom Framework befüllt wird. Welche Inhalte dort erscheinen, wird indirekt über Routing-Definitionen und den *viewManager* gesteuert.

In Testsystem übernimmt diese Aufgabe der React-Router über die Komponente `<Outlet />`.

Dieser rendert abhängig von der aktuellen Route die passende Komponente. Diese Änderung ist zwingend, da React kein internes SPA-Framework wie DevExtreme verwendet, sondern Routing als eigenständige Bibliothek organisiert. [49]

Auch die Sidebar folgt in dem Altsystem einem anderen Modell. Dort existiert lediglich ein leeres `<div id="sidedrawer"></div>`, das zur Laufzeit durch den *viewManager* manipuliert wird.

Funktionen wie *Nup.viewManager.showSidedrawer()* greifen direkt in den DOM ein und steuern das Verhalten imperativ.

Im Textsystem hingegen ist die Sidebar eine eigenständige Komponente, die als Teil des Renderbaums definiert ist. Ihr Zustand (z. B. geöffnet oder geschlossen) wird über React-State gesteuert. Diese Änderung ist eine direkte Konsequenz des React-Modells, in dem UI als Funktion des Zustands beschrieben wird und nicht mehr durch direkte DOM-Manipulation entsteht. [14]

Neben den erzwungenen Änderungen durch die Umstellung vom DevExtreme SPA Framework auf React hat Claude mehrere Änderungen eingeführt, die über den reinen Technologiewechsel hinausgehen.

Ein wesentlicher Punkt ist der Wechsel von imperativer zu deklarativer Logik. In dem Altsystem wird Verhalten aktiv gesteuert, beispielsweise durch explizite Aufrufe wie *showSidedrawer()* oder durch DOM-Manipulation. Im Testsystem wird dagegen beschrieben, wie der Zustand aussehen soll, und React kümmert sich um die Umsetzung im DOM. Die Sidebar ist hierfür ein gutes Beispiel: statt sie manuell ein- und auszublenden, wird ihr Zustand deklarativ über *useState* beschrieben. [48]

Des Weiteren wird im Altsystem das Verhalten hauptsächlich über CSS-Klassen gesteuert (z. B. *mui--hidden-xs*). Claude verlagert im Testsystem diese Logik in den Code durch:

```
const isNarrow = useMediaQuery('(max-width: 1024px)'); [51]
```

```
openedStateMode={isNarrow ? 'overlap' : 'shrink'} [50]
```

Zusätzlich wurden neue Konzepte eingeführt, die in dem Altsystem nicht vorhanden waren. Dazu gehört insbesondere Lazy Loading auf Routing-Ebene. Durch Konstrukte wie:

```
const ItemGrid = React.lazy(() => import('./views/item/ItemGrid')); wird jede Seite erst dann ge-  
laden, wenn sie tatsächlich benötigt wird. [52]
```

In dem Altsystem existiert dieses Konzept nicht. Alle Views sind Teil des initial geladenen Bundles oder werden über das Framework intern verwaltet.

Ergänzend dazu wurde ein strukturiertes Handling von Ladezuständen eingeführt. Im Testsystem wird über: `<Suspense fallback={<NpSuspenseLoading />}>` definiert, was während des Ladens einer Komponente angezeigt wird. In dem Altsystem erfolgt das Laden hingegen implizit über Framework-Mechanismen oder globale Loader wie *dxLoadIndicator*. [53], [54]

Während in dem Altsystem viele Mechanismen (Routing, Rendering, State) indirekt über Framework-Interna und globale Manager laufen, ist im Testsystem alles explizit im Code definiert.

5.5.2 Beispiel 2: Grid-Wrapper

```
export class NpDataGrid extends DevExpress.ui.dxDataGrid
implements INpClientImportantControl<NpDataGridOptions> {
  constructor(element: any, options?: NpDataGridOptions) {
    let customLayout = {
      allowedPageSizes: options["allowedPageSizes"],
      columns: options["columns"],
      pageIndex: options["pageIndex"],
      pageSize: options["pageSize"],
      searchText: options["searchText"]
    };
    if (options["onContextMenuPreparing"] === undefined) {
      options["onContextMenuPreparing"] = (e: any) => { /*...*/ };
    }
    if (options["stateStoring"] == null) {
      options["stateStoring"] = {
        type: 'custom',
        enabled: true,
        storageKey: NpDataGrid.getStorageKeyByElement(element),
        customLoad: () => this.loadLayout(customLayout, element.id),
        customSave: (layout) => this.saveLayout(element.id, layout),
      };
    }
    if (options["onContentReady"] === undefined) {
      options["onContentReady"] = (_: any) =>
        NpDataGrid.updateGridAndParentGrids($(element));
    }
  }
}
```

Ausschnitt aus NpDataGrid.ts im Altsystem

```

import DataGrid, {Properties, ContextMenuPreparingEvent, ContentReadyEvent,}
from "devextreme/ui/data_grid";
import CustomStore from "devextreme/data/custom_store";
import type { Item } from "devextreme/ui/context_menu";
import { INpClientImportantControl } from "../iNpControl";
import { NpDataGridOptions } from "../npDataGridOptions";
export class NpDataGrid<TRow, TKey = number>
  extends DataGrid<TRow, TKey>
  implements INpClientImportantControl<NpDataGridOptions<TRow, TKey>> {
  constructor(element: HTMLElement, options: Properties<TRow, TKey>) {
    const customLayout = {
      allowedPageSizes: options.pager?.allowedPageSizes,
      columns: options.columns,
      pageIndex: options.paging?.pageIndex,
      pageSize: options.paging?.pageSize,
      searchText: options.searchPanel?.text,
    };
    options.onContextMenuPreparing ??= (
      e: ContextMenuPreparingEvent<TRow, TKey>
    ) => {
      e.items = [
        ...(e.items ?? []),
        { text: "Reset layout", icon: "revert" } as Item,
      ];
    };
    options.stateStoring ??= {
      type: "custom",
      enabled: true,
      storageKey: NpDataGrid.getStorageKeyByElement(element),
      customLoad: () => this.loadLayout(customLayout, element.id),
      customSave: (layout) => this.saveLayout(element.id, layout),
    };
    options.onContentReady ??= (
      e: ContentReadyEvent<TRow, TKey>
    ) => {
      NpDataGrid.updateGridAndParentGrids(e.element);
    };
    super(element, options);
  }
}

```

Ausschnitt aus NpDataGrid.tsx im Testsystem

Vergleichen wir als Erstes die Änderungen die Claude vorgenommen hat um *NpDataGrid* an die Version 25.2.5 anzupassen.

Das Altsystem basiert auf einem globalen Namespace (*DevExpress.ui.dxDataGrid*), der über Side-Effect-Imports registriert wird. Im Testsystem existiert dieser Mechanismus nicht mehr.

Stattdessen muss das Grid explizit als ES-Modul importiert werden (*import DataGrid from "devextreme/ui/data_grid"*). Dadurch entfällt die Abhängigkeit von Lade-Reihenfolgen und globalem Zustand. [56]

Auch die Options-Struktur hat sich verändert. Im Testsystem erfolgt der Zugriff über stringbasierte Keys (*options["pageSize"]*), da kein echtes Typsystem vorhanden ist. Das Testsystem verwendet strukturierte Interfaces (*Properties<TRow, TKey>*), wodurch auf verschachtelte Eigenschaften wie *options.paging.pageSize* zugegriffen wird. Diese Änderung ist notwendig, da DevExtreme intern die Options neu modelliert hat. [57], [58], [59]

Das Event-System ist ebenfalls vollständig überarbeitet. Früher waren Event-Parameter untypisiert (*any*). Jetzt stellt DevExtreme für jedes Event eigene Typen bereit (z. B. *ContentReadyEvent*, *ContextMenuPreparingEvent*). [34], [35], [43]

Zusätzlich hat Claude die implizite jQuery-Abhängigkeit entfernt. Während das alte Grid mit jQuery-Objekten arbeitet (*\$(element)*), wird jetzt im Testsystem ein natives HTML-Element erwartet. Diese Änderung ist nicht zwingend, da DevExtreme intern jQuery nun als optionale Abhängigkeit zur Verfügung steht. [43]

Claude hat aber nicht nur den Code an die Version angepasst. Statt klassischer *if (undefined)* Checks wird nun der Nullish-Coalescing-Operator (*??=*) verwendet.

Das Event-Handling wurde erweitert. Im Testsystem wird z. B. im „onContextMenuPreparing“ direkt Funktionalität ergänzt („*Reset layout*“). Das zeigt, dass der Wrapper nicht mehr nur defensive Defaults setzt, sondern aktiv Verhalten kapselt. [60]

Außerdem hat Claude die Imports angepasst, sodass nur noch die Event-Typen importiert werden, die am Ende auch genutzt werden.

5.5.3 Beispiel 3: Gridseite Operation Detail

```

this.operation = ko.observable();
this.machineList = ko.observableArray([]);
activate() {
  this.loadData();
}
this.operationFormOptions = {
  items: [
    {
      dataField: "jobId",
      editorType: "dxSelectBox",
      editorOptions: { dataSource: this.jobLookup }
    }
  ]
};
this.tabs = [
  { title: "Machines", template: "machineTab" }
];

this.machineColumns = [
  { dataField: "machineName" },
  { dataField: "status" }
];

this.machineGrid.refresh();

```

Ausschnitte aus OperationDetail.ts im Altsystem

```

const detail = useNpDetail<OperationDto>({
  entity,
  formItems
});
function getFormItems(): NpFormItem<OperationDto>[] {
  return [
    {
      dataField: 'jobId',
      label: { text: t('job') },
      lookupId: LookupId.Job
    }
  ];
}
export function OperationDetail() {
  const detail = useNpDetail({ entity, formItems });
  return (
    <NpDetailPage detail={detail}>
      <NpTabs>
        <JobTreeTab />
        <OperationMachineTab variant="actual" />
        <OperationMachineTab variant="planned" />
        <OperationMachineTab variant="possible" />
      </NpTabs>
    </NpDetailPage>
  );
}

```

Ausschnitte aus OperationDetail.tsx im Testsystem

Im Altsystem ist die Seite als klassisches ViewModel aufgebaut. Die Klasse *OperationDetail-ViewModel* in *operationDetail.ts* fasst mehrere Aufgaben in einer einzigen Einheit: Datenverwaltung (DTOs und Knockout Observables, siehe [61]), die vollständige UI-Definition (Formulare, Grid Konfigurationen), Geschäftslogik und Routing bezogene Logik. Dieser hohe Konsolidierungsgrad ergibt sich aus dem verwendeten Architekturansatz, bei dem ViewModel als zentrale Steuerungseinheit dienen.

Darüber hinaus gibt es eine Vorlage in Form einer .dxview-Datei, die im Wesentlichen aus nichts anderem als einem leeren Container besteht. Der eigentliche Inhalt wird zur Laufzeit dynamisch vom DevExtreme SPA Framework generiert. Obwohl dieses Modell die Vorlage und die Logik formal trennt, bedeutet dies, dass die tatsächliche Struktur der Benutzeroberfläche in der Vorlage nicht direkt sichtbar ist, sondern stattdessen durch die ViewModel Logik erzeugt wird. [4], [62]

Dieses Muster ist im Testsystem nicht mehr vorhanden. Die Benutzeroberfläche wird vollständig deklarativ in JSX beschrieben. Anstelle einer zentralen Klasse ist die Logik auf mehrere spezialisierte Bausteine verteilt. Der Hook „useNpDetail“, die Komponente „NpDetailPage“ und die einzelnen Registerkartenkomponenten erfüllen jeweils eine klar definierte Aufgabe. Diese Aufteilung ist eine direkte Folge des komponentenbasierten React-Modells und ersetzt das monolithische ViewModel Konzept. [7], [10], [63]

Auch die Datenbindung unterscheidet sich grundlegend. Das Altsystem basiert auf Knockout Observables, siehe [61], die manuell abonniert (subscribed), aktualisiert und synchronisiert werden müssen. In React gibt es dieses Konzept nicht, da der Zustand über Hooks und Props gesteuert wird. [9], [13]

Über den reinen Technologiewechsel hinaus führt Claude im Testsystem mehrere strukturelle Änderungen ein, die es wert sind, einzeln hervorgehoben zu werden.

Im Altsystem sind Daten, Benutzeroberfläche und Logik in einer einzigen Klasse gebündelt. Im Testsystem sind diese Bereiche aufgeteilt. Der Hook „useNpDetail“ kapselt den Lebenszyklus der Entität (Laden, Speichern, Bearbeitungsmodus), die Komponente *NpDetailPage* stellt die generische Seitenstruktur bereit und spezialisierte Registerkarten repräsentieren isolierte Bereiche der Benutzeroberfläche. Infolgedessen wird die Seite nicht mehr „gebaut“, sondern „zusammengesetzt“. [63], [64]

Das Altsystem verfügt über keine echte Typprüfung für Formulardefinitionen wie *NpFormItem*. Feldnamen werden typischerweise über Strings oder indirekte Konstrukte referenziert. Im Testsystem wird dies durch eine auf *OperationDto* basierende Typisierung ersetzt, sodass jedes Formularfeld zur Kompilierzeit geprüft wird. Lookups werden nicht mehr manuell geladen und verwaltet, sondern automatisch über *LookupId* aufgelöst. [67]

Im Altsystem laufen mehrere *dxDataGrid* Instanzen parallel, jede mit einem eigenen *ArrayStore*, siehe [65]. Diese Speicher müssen manuell befüllt und aktualisiert sowie bei jeder Änderung der zugrunde liegenden Daten explizit neu geladen werden. Referenzen auf die Grid-Instanzen werden ebenfalls beibehalten, sodass Aktualisierungen und Auswahlen imperativ gesteuert werden können. Im Testsystem entfällt dieses Muster. Eine einzige wiederverwendbare *OperationMachineTab* Komponente wird durch Varianten gesteuert und deckt mehrere Anwendungsfälle ab. Manuelle Speicher und explizite Aktualisierungslogik sind nicht mehr erforderlich. Datenfluss und Aktualisierungen werden von den zugrunde liegenden Hooks übernommen. [15], [66]

Im Altsystem werden Daten manuell über Methoden wie *loadData()* und über Knockout-Mechanismen geladen. Zustände wie Bearbeiten, Speichern oder Initialisierung müssen explizit verwaltet werden. Im Testsystem wird dieser gesamte Prozess durch den Hook *useNpDetail* abstrahiert, der das Laden, die Validierung, das Speichern und die Statusverwaltung übernimmt. [48], [61], [63]

5.6 Probleme und Herausforderungen

Trotz der Vorteile von Claude zeigten sich auch folgende Einschränkungen:

- Die generierten Spezifikationen waren nicht immer vollständig korrekt
- Eine manuelle Überprüfung blieb weiterhin notwendig
- Bei sehr komplexen Aufgaben musste die Planung zusätzlich verfeinert werden

Dies verdeutlicht, dass das Superpower Plugin zwar eine erhebliche Unterstützung bietet, jedoch keine vollständige Automatisierung ermöglicht.

Durch die große Komplexität des Frontends zeigte sich außerdem, dass Claude Schwierigkeiten hatte, größere zusammenhängende Strukturen korrekt zu erkennen.

Bei komplexen Seiten traten folgende Fehler auf:

- Teile der Logik wurden ausgelassen
- Abhängigkeiten wurden nicht vollständig berücksichtigt
- Bestehende Funktionalitäten gingen verloren

6 Evaluation

Um die generierten Gridseiten und dessen Funktionalitäten zu testen wurde ein Vergleich zwischen dem Altsystem und der von Claude generierten Seite, dem Textsystem durchgeführt. Dabei wurde darauf geachtet, ob Features fehlen oder auch neue Features erfunden wurde. Des Weiteren wurde darauf geachtet, dass bestehende Features auch funktionieren. Darunter fällt das Bearbeiten der Informationen, das Bearbeiten der Tabellen (hinzufügen, entfernen und filtern der Spalten), die korrekte Verlinkung zu den jeweiligen Gridseiten und das Laden der richtigen Informationen aus dem Backend. Zusätzlich wurde darauf geachtet, ob gewünschte Nebenwirkungen richtig auftreten.

6.1 Testen der Gridseiten

1. Test: Maschinenübersicht:

Aktualisieren						
LoMaGr001	LoMaGr002	LoMaGr003	LoMaGr004	LoMaGr005	LoMaGr006	LoMaGr007
MaGr001.001 Status: Rüsten Arbeitsgang: DL000050.003.010 Artikel: HC020 - HC020 Bediener: Claire Grube Soll 10 / Gut 0 / Aussch. 0 / Nacharb. 0 4398:11:21 von 08:30:00	MaGr001.010 Status: kein Arbeitsgang angemel... Arbeitsgang: - Artikel: - Bediener: Claire Grube	MaGr001.011 Status: kein Arbeitsgang angemel... Arbeitsgang: - Artikel: - Bediener: Anna Bolika				
MaGr001.012 Status: Rüsten Arbeitsgang: DL000100.003.010 Artikel: H-020 - H-020 Bediener: Anna Bolika Soll 20 / Gut 0 / Aussch. 0 / Nacharb. 0 11865:20:07 von 08:25:00	MaGr001.013 Status: Rüsten Arbeitsgang: DL000092.003.010 Artikel: H-004 - H-004 Bediener: Anna Bolika Soll 20 / Gut 0 / Aussch. 0 / Nacharb. 0 11865:19:36 von 03:25:00	MaGr001.014 Status: Rüsten Arbeitsgang: DL000095.002.010 Artikel: H-009 - H-009 Bediener: Wendy Lator Soll 10 / Gut 0 / Aussch. 0 / Nacharb. 0 11865:19:12 von 02:35:00				
MaGr001.015 Status: Rüsten Arbeitsgang: DL000121.003.010 Artikel: H-002 - H-002 Bediener: Anna Bolika Soll 20 / Gut 0 / Aussch. 0 / Nacharb. 0	MaGr001.016 Status: Rüsten Arbeitsgang: DL000123.002.010 Artikel: H-005 - H-005 Bediener: Wendy Lator Soll 10 / Gut 0 / Aussch. 0 / Nacharb. 0	MaGr001.017 Status: Rüsten Arbeitsgang: DL000092.002.010 Artikel: H-003 - H-003 Bediener: Anna Bolika Soll 10 / Gut 0 / Aussch. 0 / Nacharb. 0				

Abbildung 6.1 Maschinenübersicht Vorher

<									
LOMAGR001 LOMAGR002 LOMAGR003 LOMAGR004 LOMAGR005 LOMAGR006 LOMAGR007 LOMAGR008 LOMAGR009 I >									
MaGr001.001 Status Rüsten Arbeitsgang DL000050.003.010 Artikel HC020 - HC020 Bediener Claire Grube Soll 10 / Gut 0 / Aussch. 0 / Nacharb. 0 4396:57:24 von 08:30:00					MaGr001.010 Status kein Arbeitsgang angemeldet Arbeitsgang - Artikel - Bediener Claire Grube				
MaGr001.011 Status kein Arbeitsgang angemeldet Arbeitsgang - Artikel - Bediener Anna Bolika					MaGr001.012 Status Rüsten Arbeitsgang DL000100.003.010 Artikel H-020 - H-020 Bediener Anna Bolika Soll 20 / Gut 0 / Aussch. 0 / Nacharb. 0 11864:06:10 von 08:25:00				
MaGr001.013 Status Rüsten Arbeitsgang DL000092.003.010 Artikel H-004 - H-004 Bediener Anna Bolika Soll 20 / Gut 0 / Aussch. 0 / Nacharb. 0					MaGr001.014 Status Rüsten Arbeitsgang DL000095.002.010 Artikel H-009 - H-009 Bediener Wendy Lator Soll 10 / Gut 0 / Aussch. 0 / Nacharb. 0				

Abbildung 6.2: Maschinenübersicht Nachher

Wie in den beiden Maschinenübersichten zu sehen ist, wurden alle Maschinen mit den richtigen Details angezeigt. Allerdings ist zu sehen, dass der Button „Aktualisieren“ fehlt. Des Weiteren wurden 2 Buttons hinzugefügt: ein „Zurück“ Button oben links und ein „Auto-Aktualisierung“ Button oben rechts. Diese Buttons wurden von Claude erfunden und dazu gedichtet. Die Maschinen aktualisieren sich immer in einem vorgegebenen Intervall. Mit dem „Auto-Aktualisierung“ Button kann man dies ein- und ausschalten.

<									
LOMAGR001 LOMAGR002 LOMAGR003 LOMAGR004 LOMAGR005 LOMAGR006 LOMAGR007 LOMAGR008 LOMAGR009 >									
MaGr001.001					MaGr001.010				
Status Rüsten					Status kein Arbeitsgang angemeldet				
Arbeitsgang DL000050.003.010					Arbeitsgang -				
Artikel HC020 - HC020					Artikel -				
Bediener Claire Grube					Bediener Claire Grube				
Soll 10 / Gut 0 / Aussch. 0 / Nacharb. 0									
4396:58:24 von 08:30:00									
MaGr001.011					MaGr001.012				
Status kein Arbeitsgang angemeldet					Status Rüsten				
Arbeitsgang -					Arbeitsgang DL000100.003.010				
Artikel -					Artikel H-020 - H-020				
Bediener Anna Bolika					Bediener Anna Bolika				
Soll 20 / Gut 0 / Aussch. 0 / Nacharb. 0					Soll 20 / Gut 0 / Aussch. 0 / Nacharb. 0				
11864:07:10 von 08:25:00					11864:07:10 von 08:25:00				
MaGr001.013					MaGr001.014				
Status Rüsten					Status Rüsten				
Arbeitsgang DL000092.003.010					Arbeitsgang DL000095.002.010				
Artikel H-004 - H-004					Artikel H-009 - H-009				
Bediener Anna Bolika					Bediener Wendy Lator				
Soll 20 / Gut 0 / Aussch. 0 / Nacharb. 0					Soll 10 / Gut 0 / Aussch. 0 / Nacharb. 0				

Abbildung 6.3: React Nachher 1 Minute später

Die automatische Aktualisierung der Maschinen und die Verlinkung zu der entsprechenden Gridseite wurden erfolgreich implementiert und werden entsprechend Abbildung 6.3. von Minute 57 auf Minute 58 angezeigt.

2. Test: Maschinen/Arbeitsplätze:

+ Hinzufügen		Aktualisieren			
Maschine	Maschinenbeschreibung	Status	Seit	Manuell	
Q	Q			(Alle)	
▶ K001.M001.1	⊞ MaGr001.001	Rüsten	13:48:55 18.10.24	Nach links	
▶ K001.M001.2	⊞ MaGr001.002	Rüsten	16:24:10 10.12.24	Nach rechts	
▶ K001.M001.3	⊞ MaGr001.003	Rüsten	16:24:43 10.12.24		
▶ K001.M001.4	⊞ MaGr001.004	Rüsten	16:24:58 10.12.2024		
▶ K001.M001.5	⊞ MaGr001.005	Rüsten	16:25:12 10.12.2024		
▶ K001.M001.6	⊞ MaGr001.006	Rüsten	16:25:25 10.12.2024		
▶ K001.M001.7	⊞ MaGr001.007	Rüsten	16:25:58 10.12.2024		
▶ K001.M001.8	⊞ MaGr001.008	Rüsten	16:26:17 10.12.2024		
▶ K001.M001.9	⊞ MaGr001.009	Rüsten	16:26:58 10.12.2024		
K001.M001.10	⊞ MaGr001.010	kein Arbeitsgang angemeldet	17:11:11 23.10.2025		
K001.M001.11	⊞ MaGr001.011	kein Arbeitsgang angemeldet	17:22:31 23.10.2025		
▶ K001.M001.12	⊞ MaGr001.012	Rüsten	10:40:11 11.12.2024		
▶ K001.M001.13	⊞ MaGr001.013	Rüsten	10:40:42 11.12.2024		
▶ K001.M001.14	⊞ MaGr001.014	Rüsten	10:41:05 11.12.2024		
▶ K001.M001.15	⊞ MaGr001.015	Rüsten	10:41:39 11.12.2024		
10	20	50	100	Seite 1 von 17 (327 Elemente)	

Abbildung 6.4: TypeScript Vorher

+ Hinzufügen					
Maschine	Maschinenbeschreibung	Status		Manuell	
Q	Q			(Alle)	
▶ K001.M001.1	MaGr001.001	Rüsten		Nach links verschieben	
▶ K001.M001.2	MaGr001.002	Rüsten		Nach rechts verschieben	
▶ K001.M001.3	MaGr001.003	Rüsten		Position fixieren	
▶ K001.M001.4	MaGr001.004	Rüsten	- 10.12.24, 16:24	Fixierung aufheben	
▶ K001.M001.5	MaGr001.005	Rüsten	- 10.12.24, 16:25		
▶ K001.M001.6	MaGr001.006	Rüsten	- 10.12.24, 16:25		
▶ K001.M001.7	MaGr001.007	Rüsten	- 10.12.24, 16:25		
▶ K001.M001.8	MaGr001.008	Rüsten	- 10.12.24, 16:26		
▶ K001.M001.9	MaGr001.009	Rüsten	- 10.12.24, 16:26		
K001.M001.10	MaGr001.010	kein Arbeitsgang angemeldet	- 23.10.25, 17:11		
K001.M001.11	MaGr001.011	kein Arbeitsgang angemeldet	- 23.10.25, 17:22		
▶ K001.M001.12	MaGr001.012	Rüsten	- 11.12.24, 10:40		
10	20	50	100	Seite 1 von 17 (327 Elemente)	

Abbildung 6.5: React Danach

In diesem Beispiel werden alle Maschinen mit ihren Details korrekt angezeigt.

Allerdings hat Claude die Spalten „Status“ und „Seit“ zu einer Spalte zusammengefügt und in einer Spalte „Status“ angezeigt. Auch wird die Information zentral angezeigt statt wie im Original linksbündig. Im Gegensatz zum ersten Beispiel wurde der „Aktualisieren“ Button nach oben rechts verschoben. Auch in diesem Beispiel hat Claude Buttons hinzugefügt, die es vorher nicht gab. Dazu zählt der Button von oben rechts nach links: „Aktualisieren“, „Layout zurücksetzen“, „Filter zurücksetzen“ und „Spaltenauswahl“. Der Button „Filter zurücksetzen“ wurde neu erfunden und existierte vorher nicht. Die Buttons „Spaltenauswahl“ und „Layout zurücksetzen“ waren ursprünglich im Pop-up-Fenster. In dem Layout ist weiterhin zu sehen das im Pop-up-Fenster auch 4 Buttons hinzugefügt wurde. Die Buttons „Nach dieser Spalte gruppieren“, „Alle Gruppierungen entfernen“, „Nach Links verschieben“ und „Nach Rechts verschieben“ wurden frei erfunden. Der Button „Fixieren“ wurde korrekt übernommen, hat jedoch jetzt eine andere Bezeichnung bekommen, und zwar „Position fixieren“. Dasselbe ist mit dem Button „Lösen“ passiert, der zu „Fixierung aufheben“ wurde.

Die Veränderung des Pop-up-Fensters ist problematisch, weil dies auf jeder Gridseite mit einer Tabelle für den Nutzer ersichtlich ist, nicht aber die Buttons.

Nach Einsetzen von Filtern erhält man folgendes Ergebnis:

Maschine ↑	Maschinenbeschreibung	Status	Aktiv	Manuell
Q K002	Q MaGr003	wahr	falsch	
K002.M003.1	MaGr003.001	kein Arbeitsgang angemeldet - 17.01.25, 10:00	☒	☐
K002.M003.10	MaGr003.010	kein Arbeitsgang angemeldet - 12.12.24, 13:40	☒	☐
K002.M003.11	MaGr003.011	kein Bediener angemeldet - 05.08.20, 11:04	☒	☐
K002.M003.12	MaGr003.012	kein Bediener angemeldet - 05.08.20, 11:05	☒	☐
K002.M003.13	MaGr003.013	kein Bediener angemeldet - 05.08.20, 11:05	☒	☐
K002.M003.14	MaGr003.014	kein Bediener angemeldet - 05.08.20, 11:05	☒	☐
K002.M003.15	MaGr003.015	kein Bediener angemeldet - 05.08.20, 11:05	☒	☐
K002.M003.16	MaGr003.016	kein Bediener angemeldet - 05.08.20, 11:05	☒	☐
K002.M003.17	MaGr003.017	kein Bediener angemeldet - 05.08.20, 11:05	☒	☐
K002.M003.18	MaGr003.018	kein Bediener angemeldet - 05.08.20, 11:06	☒	☐
K002.M003.19	MaGr003.019	kein Bediener angemeldet - 05.08.20, 11:06	☒	☐
> K002.M003.2	MaGr003.002	Rüsten - 11.12.24, 09:23	☒	☐

10 20 50 100

Seite 1 von 1 (25 Elemente) < 1 >

Abbildung 6.6: React Danach bearbeitet

Alle Filter und Buttons die in dem Altsystem existierten, haben die korrekte Funktionalität und haben alle das gewünschte Ergebnis angezeigt.

3. Test: Maschinendetails:

Speichern
Löschen
Aktualisieren
Arbeitsgänge umplanen

Maschine: * Manuell: ☒ Aktiv: ☒

Maschinenbeschreibung:

Maschinengruppe: *

Logische Maschinengruppe:

Kostenstelle:

Typ: *

Status: Seit: 18.10.2025, 13:48

Angemeldete Bediener	Angemeldete Arbeitsgänge	Zugeordnete Arbeitsgänge	Abgeschlossene Arbeitsgänge	Entnahmelagerplätze	Zugangslagerp	
Arbeitsgang	AG-Beschreibung	Artikel	Maschinengruppe	Status	Start Soll ↑	Ende Soll
Q	Q	Q	(Alle)	(Alle)	Q	Q
DL000031.003.010	PROD-001 AG10 KoSt001 MaGr001 tr 00:05:00 te 00:10:00	H-002	K001.M001 - MaGr001	Gestartet	23.03.2021, 09:16	23.03.2021, 13:11
DL000031.002.010	PROD-001 AG10 KoSt001 MaGr001 tr 00:05:00 te 00:10:00	H-001	K001.M001 - MaGr001	Menge gemeldet	23.03.2021, 11:36	23.03.2021, 13:21
DL000032.003.010	PROD-001 AG10 KoSt001 MaGr001 tr 00:05:00 te 00:10:00	H-004	K001.M001 - MaGr001	Gestartet	23.03.2021, 13:11	24.03.2021, 08:06

Abbildung 6.7: Maschinendetail Vorher

Bearbeiten
Löschen

Maschine
K001.M001.1

Maschinenbeschreibung
MaGr001.002

Logische Maschinengruppe
LoMaGr001 - LoMaGr001

Typ
Testterminal - Testterminal

Manuell
✓

Aktiv
✓

Maschinengruppe
K001.M001 - MaGr001

Kostenstelle
K001 - KoSt001

Status
Rüsten - 18.10.25, 13:48

ANGEMELDETE BEDIENER ANGEMELDETE ARBEITSGÄNGE **ZUGEORDNETE ARBEITSGÄNGE** ABGESCHLOSSENE ARBEITSGÄNGE ENT >

Arbeitsgang	AG-Bes...	Artikel	Maschi...	Status	Start... ↑	Ende Soll	Sollme...	Gut	Aussch...	Produkt...
DL000031.003.010	PROD-0...	H-002	K001.M...	Gestartet	23.3.20...	23.3.20...	20	10	0	03:25:00
DL000031.002.010	PROD-0...	H-001	K001.M...	Menge ...	23.3.20...	23.3.20...	10	0	0	01:45:00
DL000032.003.010	PROD-0...	H-004	K001.M...	Gestartet	23.3.20...	24.3.20...	20	0	0	03:25:00
DL000033.003.010	PROD-0...	H-006	K001.M...	Nicht g...	23.3.20...	24.3.20...	20	0	0	05:05:00
DL000032.002.010	PROD-0...	H-003	K001.M...	Gestartet	24.3.20...	24.3.20...	10	0	0	01:45:00

10 **20** 50 100

Seite 1 von 4 (79 Elemente) < **1** 2 3 4 >

Abbildung 6.8: React Nachher

Im oberen Abschnitt der Abbildungen sieht man jeweils die Maschinendetails und im unteren Abschnitt die entsprechenden Arbeitsgänge der Maschine.

Wie im 2. Test ist der „Aktualisieren“ Button des oberen Abschnitts nach oben rechts verschoben genauso wie die Buttons: „Aktualisieren“, „Layout zurücksetzen“, „Filter zurücksetzen“ und „Spaltenauswahl“ im unteren Abschnitt. Auch „Status“ und „Seit“ werden wieder als eine Information dargestellt.

Der Button „Speichern“ erscheint jetzt nur wenn man den „Bearbeiten“ Button klickt. „Arbeitsgänge umplanen“ existiert nicht, da die entsprechende Gridseite nicht existiert.

Im oberen Abschnitt links neben dem „Aktualisieren“ Button sind 3 neue Buttons hinzugekommen. Diese sind von links nach rechts: „Formular + Tabs“, „Nur Formular“ und „Nur Tabs“.

Bei Betätigung des Button „Formular + Tabs“ werden die Maschinendetails und die Arbeitsgänge angezeigt, bei „Nur Formular“ werden nur die Maschinendetails und bei „Nur Tabs“ nur die Arbeitsgänge angezeigt. Alle Informationen zu den Maschinen und den dazugehörigen Arbeitsgängen werden korrekt dargestellt.

Nach dem Bearbeiten der Maschinendetails erhält man folgendes Ergebnis:

The screenshot displays a web interface for managing machine details and work steps. At the top, there are buttons for 'Bearbeiten' (Edit) and 'Löschen' (Delete). Below this, the machine details are shown in a grid-like format:

Maschine	Manuell	Aktiv
K001.M001.1	✓	X
Maschinenbeschreibung	Maschinengruppe	
MaGr001.001	K003.M005 - MaGr005	
Logische Maschinengruppe	Kostenstelle	
LoMaGr003 - LoMaGr003	FREMD - Fremdvergabe	
Typ	Status	
Testterminal - Testterminal	kein Bediener angemeldet - 19.	

Below the details, there are tabs for 'ANGEMELDETE BEDIENER', 'ANGEMELDETE ARBEITSGÄNGE', 'ZUGEORDNETE ARBEITSGÄNGE' (selected), and 'ABGESCHLOSSENE ARBEITSGÄNGE'. The 'ZUGEORDNETE ARBEITSGÄNGE' tab shows a table with the following columns: Arbeits..., AG-Bes..., Artikel, Maschi..., Status, Start..., Ende Soll, Sollme..., Gut, Aussch..., Produkt..., and Produkt... The table is currently empty, displaying 'Keine Daten' (No data). At the bottom, there is a pagination bar showing 'Seite 1 von 1 (0 Elemente)' and a red circle with the number '1'.

Abbildung 6.9: React Danach bearbeitet

Die Informationen nach Bearbeiten des Arbeitsganges werden gespeichert und dargestellt. Die Maschine wurde korrekt auf inaktiv gesetzt und damit automatisch auch der Bediener abgemeldet.

6.2 Ergebnisse der Tests

Zu den im Kapitel 6.1 gezeigten Testbeispielen wurde noch 25 weitere Gridseiten erstellt und getestet. Wenn man wiederkehrende Fehler, wie das Tabellen-Pop-up, zur Seite stellt, wurden in 10 von 25 Seiten Mängel festgestellt. Dabei ist anzumerken, dass desto einfacher die Gridseite aufgebaut ist, desto weniger bis kleinere Mängel treten auf.

Damit ergibt sich folgendes Ergebnis an erfüllten Anforderungen:

- Alle Felder haben dieselben Eigenschaften wie vorher
- Abhängigkeiten zwischen Feldern bleiben erhalten
- Navigation und Routing entspricht exakt der alten Anwendung
- Daten werden korrekt geladen, aktualisiert und gespeichert
- Berechtigungen wirken sich korrekt auf Sichtbarkeit und Zugriff aus

Nicht erfüllte Anforderungen:

- Alle UI-Elemente sind vollständig vorhanden und sichtbar
- Grid-Funktionalitäten bleiben erhalten
- Lade- und Fehlzustände werden korrekt angezeigt

Wie in Kapitel 5.5 und 5.6 beschrieben entwickelte Claude eigenständig eine Umsetzung für eine Überführung vom DevExtreme SPA Framework zu React und der DevExtreme Version 18.2.8 zur Version 25.2.5. Claude optimierte und änderte einzelne Funktionalitäten. Dabei ist auch in Kapitel 6.1 zu sehen, dass Claude einige UI-Elemente hinzugefügt hat die es vorher gar nicht gab oder auch UI-Elemente vergessen hat.

7 Interpretation und Ausblick

Für die Anforderungen an das LLM kann also festgehalten werden:

Die Implementierung der Grundanwendung in den jeweiligen Gridseiten, wie zum Beispiel in Kapitel 5.5.3 beschrieben, ist eine Antwort auf die Forschungsfragen „Kann Claude bei der Überführung vom DevExtreme SPA Framework auf React unterstützen?“ und „Kann Claude bei der Überführung von DevExtreme 18.2.18 auf 25.2.5 unterstützen?“ (siehe Kapitel 1.2)

In der „Prüfungsordnung für den Bachelorstudiengang Informatik“ der Westsächsischen Hochschule Zwickau ist die Bearbeitungszeit für diese Bachelorarbeit auf 10 Wochen beschränkt. Diese zeitliche Begrenzung und die u.a. in Kapitel 6.1 und 6.2 festgestellten Ergebnisse liefern die Antwort, die für die Forschungsfrage „Welche Nebenwirkungen treten auf und wo liegen die Grenzen der Migration mit Claude?“ gefunden wurde.

7.1 Grenzen der Evaluation

Die getestete Anwendung wurde auf die Funktionalität der Grundanwendung und der jeweiligen Gridseiten getestet. Wie in 6.1 beschrieben, waren diese Tests fehlerbehaftet. Ob wirklich alle Funktionalitäten richtig migriert wurden, ist aufgrund der Komplexität des NuPMES und der vorher beschriebenen zeitlichen Einschränkungen nicht möglich. Dazu bedarf es ausführlich erstellter Frontendtests, um sicherzustellen, dass alle Edge-Cases und alle Features der DevExtreme Version 25.2.5 berücksichtigt wurden.

Es wurde ausschließlich ein Teil des Frontends, der Monitor, für die Migration getestet. Wie in dem Kapitel 6.1 und 6.2 gezeigt, wurde nicht alle Anforderungen, die an das LLM gestellt wurden, erfüllt.

7.2 Ausblick

Der Einsatz von Claude Opus 4.6 und 4.7 kann einen Startpunkt für die Migration des NuPMES sein.

Für zukünftige Arbeiten wäre ein Vergleich zwischen verschiedenen LLM Modellen sinnvoll, um herauszufinden, ob andere LLM Modelle bessere Ergebnisse liefern. Außerdem bietet es sich an den Kontext des LLM noch weiter einzuschränken, wie z. Bsp. mit stärkeren Constraints um Nebenwirkungen zu minimieren.

Des Weiteren sind ausgiebige Frontendtests zu erstellen, um alle wichtigen Fälle auf ihre Funktionalität zu überprüfen.

Literaturverzeichnis

- [1] DevExpress: *Minor/Maintenance Updates*. - <https://www.devexpress.com/support/versions.xml#major>; abgerufen am 12. April 2026
- [2] DevExpress: Version History. - <https://supportcenter.devexpress.com/versionhistory>; abgerufen am 12. April 2026
- [3] TypeScript: DOM-Manipulation. - <https://www.typescriptlang.org/docs/handbook/dom-manipulation.html>; abgerufen am 12. April 2026
- [4] DevExtreme: DevExtreme jQuery/JS - Views and Layouts https://js.devexpress.com/jQuery/Documentation/17_2/Guide/SPA_Framework/Views_and_Layouts/; abgerufen am 12. April 2026
- [5] DevExtreme: jQuery/JS SPA Framework - Markup Components – ViewPlaceholder. - http://js.devexpress.com/jQuery/Documentation/18_1/ApiReference/SPA_Framework/Markup_Components/dxViewPlaceholder/; abgerufen am 12. April 2026
- [6] DevExtreme: DevExtreme jQuery/JS - Navigation and Routing. - https://js.devexpress.com/jQuery/Documentation/17_2/Guide/SPA_Framework/Navigation_and_Routing/; abgerufen am 12. April 2026
- [7] React: Reacting to Input with State <https://react.dev/learn/reacting-to-input-with-state#how-declarative-ui-compares-to-imperative>; abgerufen am 13. April 2026
- [8] React: *Rules of React*, - <https://react.dev/reference/rules#react-calls-components-and-hooks>; abgerufen am 13. April 2026
- [9] React: *State as a Snapshot*, - <https://react.dev/learn/state-as-a-snapshot#rendering-takes-a-snapshot-in-time>; abgerufen am 13. April 2026
- [10] React: *Your First Component*, - <https://react.dev/learn/your-first-component#components-ui-building-blocks>; abgerufen am 13. April 2026
- [11] React : *Writing Markup with JSX*, - <https://react.dev/learn/writing-markup-with-jsx>; abgerufen am 13. April 2026
- [12] DevExtreme: *DevExtreme React - Component Configuration Syntax*, - https://js.devexpress.com/React/Documentation/Guide/React_Components/Component_Configuration_Syntax/; abgerufen am 13. April 2026
- [13] React: *Passing Props to a Component*, - <https://react.dev/learn/passing-props-to-a-component>; abgerufen am 13. April 2026
- [14] React: *State: A Component's Memory*, - <https://react.dev/learn/state-a-components-memory>; abgerufen am 13. April 2026
- [15] React: *A Component's Memory*, - <https://react.dev/learn/state-a-components-memory#when-a-regular-variable-isnt-enough>; abgerufen am 13. April 2026

- [16] React: *Introducing Hooks*, - <https://legacy.reactjs.org/docs/hooks-intro.html>; abgerufen am 14. April 2026
- [17] React: *Rules of Hooks*, - <https://legacy.reactjs.org/docs/hooks-overview.html>; abgerufen am 14. April 2026
- [18] React: *Built-in React Hooks*, - <https://react.dev/reference/react/hooks#state-hooks>; abgerufen am 14. April 2026
- [19] React: *Responding to Events*, - <https://react.dev/learn/responding-to-events>; abgerufen am 14. April 2026
- [20] React: *Responding to Events*, - <https://react.dev/learn/responding-to-events#adding-event-handlers>; abgerufen am 14. April 2026
- [21] React: *Responding to Events*, - <https://react.dev/learn/responding-to-events#naming-event-handler-props>; abgerufen am 14. April 2026
- [22] React: *DevExtreme React - Component Configuration Syntax*, - https://js.devexpress.com/React/Documentation/Guide/React_Components/Component_Configuration_Syntax/#Event_Handling; abgerufen am 14. April 2026
- [23] React: *Single Page App*, - <https://reactrouter.com/how-to/spa>; abgerufen am 14. April 2026
- [24] DevExtreme: *DevExtreme React - Component Configuration Syntax*, - https://js.devexpress.com/React/Documentation/Guide/React_Components/Component_Configuration_Syntax/#Markup_Customization/Using_a_Custom_Component; abgerufen am 14. April 2026
- [25] Vite: *Static Asset handling*, - <https://vite.dev/guide/assets>; abgerufen am 14. April 2026
- [26] DevExtreme *DevExtreme React - Predefined Themes*, - https://js.devexpress.com/React/Documentation/Guide/Themes_and_Styles/Predefined_Themes/; abgerufen am 15. April 2026
- [27] Nupis: *Echtzeitdatenerfassung liefert aussagefähige Informationen in der diskreten Fertigung*, - <https://www.nupis.de/produkte/nupmes>; abgerufen am 15. April 2026
- [28] Introducing Claude Opus 4.6, - <https://www.anthropic.com/news/claude-opus-4-6>; abgerufen am 15. April 2026
- [29] Introducing Claude Opus 4.7, - <https://www.anthropic.com/news/claude-opus-4-7>; abgerufen am 15. April 2026
- [30] Devextreme: *JavaScript/jQuery DataGrid - Enhance Performance on Large Datasets*, - https://js.devexpress.com/jQuery/Documentation/Guide/UI_Components/DataGrid/Enhance_Performance_on_Large_Datasets/; abgerufen am 15. April 2026
- [31] DevExtreme: *JavaScript/jQuery DataGrid – scrolling*, - https://js.devexpress.com/jQuery/Documentation/ApiReference/UI_Components/dxDataGrid/Configuration/scrolling/; abgerufen am 15. April 2026

- [32] DevExtreme: *DevExtreme jQuery/JS - ThemeBuilder*, -
https://js.devexpress.com/jQuery/Documentation/Guide/Themes_and_Styles/ThemeBuilder/; abgerufen am 15. April 2026
- [33] DevExtreme: *DevExtreme jQuery/JS - Predefined Themes*, -
https://js.devexpress.com/jQuery/Documentation/Guide/Themes_and_Styles/Predefined_Themes/; abgerufen am 15. April 2026
- [34] DevExtreme: *JavaScript/jQuery DataGrid – CellClickEvent*, -
https://js.devexpress.com/jQuery/Documentation/ApiReference/UI_Components/dxDataGrid/Types/CellClickEvent/; abgerufen am 15. April 2026
- [35] DevExtreme: *JavaScript/jQuery DataGrid – InitializedEvent*, -
https://js.devexpress.com/jQuery/Documentation/ApiReference/UI_Components/dxDataGrid/Types/InitializedEvent/; abgerufen am 15. April 2026
- [36] DevExtreme: *DevExtreme JavaScript Components v21.1*, -
[https://www.devexpress.com/subscriptions/new-2021-1.xml#devextreme](https://www.devexpress.com/subscriptions/new-2021-1.xml#devextreme;);
- [37] DevExpress: *DevExtreme (v23.1, v23.2) — Accessibility Enhancements (Angular, React, Vue, jQuery)*, -
<https://community.devexpress.com/blogs/javascript/archive/2024/01/19/devextreme-v23-1-v23-2-accessibility.aspx>; abgerufen am 15. April 2026
- [38] DevExpress: *DevExtreme JavaScript Components v23.2*, -
<https://www.devexpress.com/subscriptions/new-2023-2.xml#devextreme>;
abgerufen am 15. April 2026
- [39] DevExpress: *DevExtreme — Year-End Roadmap v23.2 (Angular, React, Vue, jQuery & ASP.NET Core)*, -
<https://community.devexpress.com/blogs/javascript/archive/2023/08/15/devextreme-roadmap-v23-2.aspx>; abgerufen am 15. April 2026
- [40] DevExtreme: *JavaScript/jQuery Form - Accessibility*, -
https://js.devexpress.com/jQuery/Documentation/Guide/UI_Components/Form/Accessibility/; abgerufen am 15. April 2026
- [41] DevExpress: *JavaScript & ASP.NET Core Components v24.2*, -
<https://www.devexpress.com/subscriptions/new-2024-2.xml#devextreme>;
abgerufen am 15. April 2026
- [42] DevExtreme: *DevExtreme jQuery/JS - Best Practices*, -
https://js.devexpress.com/jQuery/Documentation/Guide/Common/TypeScript_Guides/Best_Practices/; abgerufen am 15. April 2026
- [43] DevExtreme: *DevExtreme jQuery/JS - v25.2 Release Notes*, -
https://js.devexpress.com/jQuery/Documentation/Guide/Common/Whats_New/v25.2_Release_Notes/; abgerufen am 15. April 2026

- [44] DevExpress: *Exposes DevExtreme data layer.*, - https://github.com/DevExpress/devextreme-documentation/blob/26_1/concepts/Common/Modularity/02%20DevExtreme%20Modules%20Structure/common/data.md; abgerufen am 15. April 2026
- [45] Claude Code Docs: *How Claude remembers your project*, - <https://code.claude.com/docs/en/memory>; abgerufen am 15. April 2026
- [46] Claude: *CLAUDE.md-Dateien verwenden: Anpassung von Claude Code für ihre Codebasis*, - <https://claude.com/de-de/blog/using-claude-md-files>; abgerufen am 15. April 2026
- [47] Claude Code Docs: *Create custom subagents*, - <https://code.claude.com/docs/en/sub-agents>; abgerufen am 15. April 2026
- [48] React: *useState*, - <https://react.dev/reference/react/useState>; abgerufen am 16. April 2026
- [49] React Router: *Routing*, - <https://reactrouter.com/start/declarative/routing>; abgerufen am 16. April 2026
- [50] DevExtreme: *JavaScript/jQuery Drawer Options*, - https://js.devexpress.com/jQuery/Documentation/ApiReference/UI_Components/dxDrawer/Configuration/#openedStateMode; abgerufen am 16. April 2026
- [51] MUI: *useMediaQuery*, - <https://mui.com/material-ui/react-use-media-query/>; abgerufen am 16. April 2026
- [52] React: *lazy*, - <https://react.dev/reference/react/lazy>; abgerufen am 16. April 2026
- [53] DevExtreme: *JavaScript/jQuery LoadIndicator API*, - https://js.devexpress.com/jQuery/Documentation/ApiReference/UI_Components/dxLoadIndicator/; abgerufen am 16. April 2026
- [54] React: *<Suspense>*, - <https://react.dev/reference/react/Suspense>; abgerufen am 16. April 2026
- [55] Redux: *Store*, - <https://redux.js.org/api/store>; abgerufen am 16. April 2026
- [56] DevExtreme: *JavaScript/jQuery DataGrid API*, - https://js.devexpress.com/jQuery/Documentation/ApiReference/UI_Components/dxDataGrid/; abgerufen am 16. April 2026
- [57] DevExtreme: *JavaScript/jQuery DataGrid Options*, - https://js.devexpress.com/jQuery/Documentation/ApiReference/UI_Components/dxDataGrid/Configuration/; abgerufen am 16. April 2026
- [58] DevExtreme: *JavaScript/jQuery DataGrid – paging*, - https://js.devexpress.com/jQuery/Documentation/ApiReference/UI_Components/dxDataGrid/Configuration/paging/; abgerufen am 16. April 2026

- [59] DevExtreme: *JavaScript/jQuery DataGrid – ContentReadyEvent*, - https://js.devexpress.com/jQuery/Documentation/ApiReference/UI_Components/dxDataGrid/Types/ContentReadyEvent/; abgerufen am 16. April 2026
- [60] DevExtreme: *JavaScript/jQuery DataGrid Options*, - https://js.devexpress.com/jQuery/Documentation/ApiReference/UI_Components/dxDataGrid/Configuration/#onContextMenuPreparing; abgerufen am 16. April 2026
- [61] Knockout.: *Observables*. - <https://knockoutjs.com/documentation/observables.html>; abgerufen am 16. April 2026
- [62] DevExtreme: *jQuery/JS SPA Framework - Markup Components – View*, - https://js.devexpress.com/jQuery/Documentation/18_2/ApiReference/SPA_Framework/Markup_Components/dxView/; abgerufen am 16. April 2026
- [63] React: *Reusing Logic with Custom Hooks*; - <https://react.dev/learn/reusing-logic-with-custom-hooks>; abgerufen am 16. April 2026
- [64] React: *Thinking in React*, - <https://react.dev/learn/thinking-in-react#step-1-break-the-ui-into-a-component-hierarchy>; abgerufen am 16. April 2026
- [65] DevExtreme: *DevExtreme jQuery - ArrayStore API*, - https://js.devexpress.com/jQuery/Documentation/ApiReference/Data_Layer/ArrayStore/; abgerufen am 16. April 2026
- [66] DevExtreme: *JavaScript/jQuery DataGrid Methods*, - https://js.devexpress.com/jQuery/Documentation/ApiReference/UI_Components/dxDataGrid/Methods/#refresh; abgerufen am 16. April 2026
- [67] DevExtreme: *JavaScript/jQuery Form Options*, - https://js.devexpress.com/jQuery/Documentation/ApiReference/UI_Components/dxForm/Configuration/; abgerufen am 16. April 2026
- [68] LanguageTooler GmbH: *Mehr als eine Rechtschreibprüfung*. Internetquelle, 2026.– <https://languagetool.org/de>; abgerufen am 20. April 2026
- [69] DeepL SE: *DeepL*. Internetquelle, 2026.– <https://www.deepl.com/de/translator>; abgerufen am 20. April 2026
- [70] Visual Studio Code: *The open source AI code editor*, -<https://code.visualstudio.com/>; abgerufen am 20. April 2026
- [

Anhang

Bestehender Code im Altsystem

viewManager.ts

```
export class ViewManager {
  hideMenuOverlay() {
    const sideDrawer = $('#sidedrawer');
    if (sideDrawer == null) return;
    sideDrawer.removeClass('active').appendTo(document.body);
  }
  hideSidedrawer() {
    const bodyEl = $('body');
    const className = 'hide-sidedrawer';
    bodyEl.toggleClass(className);
    userContext.setSideBarHidden(bodyEl.hasClass(className));
    userManager.saveDataInStorage();
  }
  navigateToLoginView() { App.navigate(this.loginUrl); viewManager.setSubHeaderText(' '); }
  navigateToHomeView() { App.navigate(this.homeUrl); viewManager.setSubHeaderText(' '); }
  navigateBack(defer: JQueryDeferred<void>) { App.back(); defer.resolve(); }

  navigate(view: string, id: number = 0, replaceHistory: boolean = false) {
    this.navigateToView(view, id, null, replaceHistory);
  }
  private navigateToView(view: string, id = 0,
    requestParameter: RequestParameter[] = null,
    replaceHistory = false) {
    if (id > 0) view += "/" + id;
    view += urlHelper.getUrlParameter(requestParameter);
    App.navigate(view, { target: replaceHistory ? 'current' : 'blank' });
  }
}

export let viewManager = new ViewManager();
```

viewmodel.ts

```
export abstract class ViewModel {  
  
    feature: FeatureId;  
    private viewModelState: ViewmodelState;  
    params: any;  
    lookupRequestEntries: ILookupRequestEntry[];  
    lookupDatas: ILookupResponseEntry[];  
    toolbarButtons: ToolbarButton = ToolbarButton.None;  
    extraToolbarButtons: Array<ToolbarButtonObject> = new Array();  
    service: string;  
    protected helpPopupOptions: DevExpress.ui.dxPopupOptions = {};  
    protected excelExportOptions: ToolbarExcelExportOptions = {};  
    canLeaveCurrentView: boolean = true;  
    isCurrentViewLeaving: boolean = false;  
    constructor(params: any) {  
        this.params = params;  
        Nup.currentViewModel = this;  
        this.feature = FeatureId.Empty;  
        this.viewModelState = ViewmodelState.Empty;  
        this.setService();  
        this.createEventHandlers();  
        this.registerEventHandlers();  
    }  
    viewRendered() {  
        toolbarViewModel.initToolbarButton(this.toolbarButtons, this.extraToolbarButtons);  
        if (this.params?.id) {  
            this.execForEachForm(f => this.updateRequiredFormItems(f));  
        }  
    }  
    viewShowing() { toolbarViewModel.refreshButtonStyle(); }  
    viewShown() { this.loadLookups(); }  
    abstract load(id?: number);  
    // ~600 lines of lookup loading, layout persistence, toolbar handling
```

npDataGrid

```
import { isArrayNotNullOrEmpty } from "../helper/arrayHelper";
import { KeyCode } from "../helper/const";
import { localize } from "../helper/stringHelper";
import { userContext } from "../manager/customUserContext";
import { INpClientImportantControl } from "../iNpControl";
import { checkFeatures } from "../npControl";
import { NpDataGridOptions } from "../npDataGridOptions";

export class NpDataGrid extends DevExpress.ui.dxDataGrid
  implements INpClientImportantControl<NpDataGridOptions> {

  constructor(element: any, options?: NpDataGridOptions) {
    let customLayout = {
      allowedPageSizes: options["allowedPageSizes"],
      columns:          options["columns"],
      pageIndex:        options["pageIndex"],
      pageSize:         options["pageSize"],
      searchText:       options["searchText"]
    };

    if (options["onContextMenuPreparing"] === undefined) {
      options["onContextMenuPreparing"] = (e: any) => {
        let ctxItems = e.items;
        let itemRevertLayout = {
          disabled: false, icon: "revert",
          onClick: (_) => { this.state(customLayout); },
          text: localize("revertLayout"), value: "None", beginGroup: true,
        };
        let itemColumnChooser = {
          disabled: false, icon: "column-chooser",
          onClick: (_) => this.showColumnChooser(),
          text: localize("columnChooser"), value: "None"
        };
        NpDataGrid.setCtxItems(ctxItems, itemRevertLayout, itemColumnChooser);
      };
    }

    if (options["stateStoring"] == null) {
      options["stateStoring"] = {
        type: 'custom', enabled: true,
        storageKey: NpDataGrid.getStorageKeyByElement(element),
        savingTimeout: 100,
        customLoad: () => this.loadLayout(customLayout, element.id),
        customSave: (layout) => this.saveLayout(element.id, layout)
      };
    }

    if (options["onContentReady"] === undefined) {
      options["onContentReady"] =
        (_: any) => NpDataGrid.updateGridAndParentGrids($(element));
    }

    // ...default flags (showBorders, showColumnLines, showRowLines) elided...

    NpDataGrid.setDefaultProperties(options);
    NpDataGrid.setDefaultEventHandler(options);
    super(element, options);
    this.afterInit(options);
  }

  // ...remaining 380 lines (layout helpers, update routines) omitted...
}
```

Generierter Code im Testsystem

npOperationDetail.tsx

```

export function OperationDetail(): React.ReactElement {
  const { id } = useParams<{ id?: string }>();
  const navigate = useNavigate();
  const hasFeature = useAuthStore(s => s.hasFeature);
  const hasJobFeature = hasFeature(FeatureId.Job, Right.Read);

  const detail = useNpDetail<OperationDto>({
    entity: entities.operations,
    formItems: getFormItems,
    extraToolbarButtons: [{
      id: 'manageJobItems',
      text: t('manageJobItems'),
      icon: 'product',
      visible: hasJobFeature && !!id,
      onClick: () => navigate(`/job-items/${id}`),
    }],
  });

  return (
    <NpDetailPage detail={detail}>
      <NpTabs id="operation-detail">
        <JobTreeTab
          id="operationDetail-jobTree"
          title={t('jobTree')}
          entity={entities.operationJobTree}
          params={{ id }}
          itemsEntity={entities.items}
        />
        <OperationMachineTab id="operationDetail-actualMachines"
          title={t('actualMachines')}
          variant="state"
          entity={entities.operationActualMachines}
          params={{ id }} />
        <OperationMachineTab id="operationDetail-possibleMachines"
          title={t('possibleMachines')}
          variant="possible"
          entity={entities.operationPossibleMachines}
          params={{ id }} />
        <OperationMachineTab id="operationDetail-plannedMachines"
          title={t('plannedMachines')}
          variant="state"
          entity={entities.operationPlannedMachines}
          params={{ id }} />
        <BomTab
          id="operationDetail-material"
          title={t('material')}
          entity={entities.operationBom}
          params={{ id }}
          itemsEntity={entities.items}
        />
      </NpTabs>
    </NpDetailPage>
  );
}

```

useNpDetail.ts

```
export type DetailViewMode = 'both' | 'formOnly' | 'tabsOnly';
export type NpDetailPageButtonKey = 'save' | 'delete' | 'edit' | 'cancel';

export interface UseNpDetailOptions<T = Record<string, unknown>>
  extends UseDetailPageOptions {
  formItems:      NpFormItem<T>[] | ((formData: T) => NpFormItem<T>[]);
  entity?:        EntityDefinition;
  toolbarButtons?: NpDetailPageButtonKey[];
  colCount?:      number;
  labelLocation?: 'top' | 'left' | 'right';
  extraToolbarButtons?: NpToolbarButton[];
}

export interface UseNpDetailResult<T> {
  formProps:      NpDetailFormProps;
  links:          Record<string, HallLink | undefined>;
  editMode:       boolean;
  isNew:          boolean;
  isLoading:      boolean;
  lookupsReady:   boolean;
  viewMode:       DetailViewMode;
  toolbarProps:   NpToolbarProps;
  deleteDialogProps: NpConfirmDialogProps;
  // handlers for save / cancel / delete / edit-toggle
  // ... (~600 further lines: data fetching, validation, dirty tracking,
  //      lookup resolution, HAL link evaluation, router integration)...
}

export function useNpDetail<T>(options: UseNpDetailOptions<T>): UseNpDetailResult<T>;
```

NpDetailPage.ts

```
export interface NpDetailPageProps {
  detail: UseNpDetailResult<unknown>;
  children?: React.ReactNode; // tabs
}

export function NpDetailPage({ detail, children }: Readonly<NpDetailPageProps>) {
  const showTabs = !detail.isNew
    || Object.values(detail.formProps.formData).some(v => Array.isArray(v));
  const showForm = detail.viewMode !== 'tabsOnly';
  const showTabSection = showTabs && detail.viewMode !== 'formOnly';

  const { formProps } = detail;
  const contextValue = useMemo(() => ({
    formData: formProps.formData,
    links: detail.links,
    isEditMode: detail.editMode,
    isNew: detail.isNew,
  })), [formProps.formData, detail.links, detail.editMode, detail.isNew]);
  return (
    <NpPage page={detail}>
      {!detail.isLoading && detail.lookupsReady ? (
        <DetailPageContext.Provider value={contextValue}>
          {showForm && (
            <NpForm
              ref={formProps.ref}
              formData={formProps.formData}
              items={formProps.items}
              readOnly={formProps.readOnly}
              onFieldDataChanged={formProps.onFieldDataChanged}
            />
          )}
          {showTabSection && React.Children.count(children) > 0 && (
            <div style={{ flex: 1 }}>{children}</div>
          )}
        </DetailPageContext.Provider>
      ) : null}
      <NpConfirmDialog {...detail.deleteDialogProps} />
    </NpPage>
  );
}
```

OperationMachineTab.tsx

```
import React from 'react';
import { NpTabGrid } from '@shared/components/tabs/NpTabGrid';
import type { NpGridColumn } from '@shared/components/grid/NpGrid';
import type { EntityDefinition } from '@shared/entities';
import { t } from '@shared/services/localization';
import { LookupId } from '@shared/api/api';
import type { MachineStateDto, MachineDto } from '@shared/api/types';
import { entities } from '@shared/entities';

function getStateColumns(): NpGridColumn<MachineStateDto>[] {
  return [
    { dataField: 'machineNo', caption: t('machine'), dataType: 'string', linkTo: entities.machines },
    { dataField: 'description', caption: t('machineDescription'), dataType: 'string' },
    { dataField: 'costcenter', caption: t('costcenter'), lookupId: LookupId.Costcenter, visible: false },
    { dataField: 'groupId', caption: t('machineGroup'), lookupId: LookupId.MachineGroup },
    { dataField: 'logicalGroup', caption: t('logicalGroup'), lookupId: LookupId.LogicalMachineGroup },
    { dataField: 'failureId', caption: t('state'), lookupId: LookupId.Failure, visible: false },
  ];
}

function getPossibleColumns(): NpGridColumn<MachineDto>[] {
  return [
    { dataField: 'machineNo', caption: t('machine'), dataType: 'string', linkTo: entities.machines },
    { dataField: 'description', caption: t('machineDescription'), dataType: 'string' },
    { dataField: 'costcenter', caption: t('costcenter'), lookupId: LookupId.Costcenter, visible: false },
    { dataField: 'groupId', caption: t('machineGroup'), lookupId: LookupId.MachineGroup },
    { dataField: 'logicalGroup', caption: t('logicalGroup'), lookupId: LookupId.LogicalMachineGroup },
    { dataField: 'active', caption: t('active'), dataType: 'boolean' },
  ];
}

export interface OperationMachineTabProps {
  id: string;
  title: string;
  entity: EntityDefinition;
  params: Record<string, string | undefined>;
  variant: 'state' | 'possible';
}

export function OperationMachineTab({
  id, title, entity, params, variant,
}: Readonly<OperationMachineTabProps>): React.ReactElement {
  if (variant === 'possible') {
    return (
      <NpTabGrid<MachineDto>
        title={title} id={id} entity={entity} params={params}
        columns={getPossibleColumns()}
      />
    );
  }
  return (
    <NpTabGrid<MachineStateDto>
      title={title} id={id} entity={entity} params={params}
      columns={getStateColumns()}
    />
  );
}
```

Sidebar.tsx

```
// Signature and key responsibilities only:
export interface SidebarProps {
  onNavigate?: () => void;
}

export function Sidebar({ onNavigate }: SidebarProps): React.ReactElement {
  const hasFeature = useAuthStore(s => s.hasFeature);
  const [collapsedGroups, setCollapsedGroups] = useState<Set<string>>(() => loadCollapsed());

  // Build the visible nav tree based on the user's feature permissions
  const treeItems = useMemo(
    () => buildTreeItems(hasFeature, collapsedGroups),
    [hasFeature, collapsedGroups],
  );

  const navigate = useNavigate();

  return (
    <ScrollView>
      <TreeView
        items={treeItems}
        onItemClick={e => {
          if (e.itemData?.path) {
            navigate(e.itemData.path);
            onNavigate?.(); // notify parent so it can close the drawer
          }
        }}
        onItemExpanded={handleGroupExpanded}
        onItemCollapsed={handleGroupCollapsed}
      />
    </ScrollView>
  );
}
```

MonitorLayout.tsx

```
function useMediaQuery(query: string): boolean {
  const [matches, setMatches] = useState(() => globalThis.matchMedia(query).matches);
  useEffect(() => {
    const mql = globalThis.matchMedia(query);
    const handler = (e: MediaQueryListEvent): void => setMatches(e.matches);
    mql.addEventListener('change', handler);
    return () => mql.removeEventListener('change', handler);
  }, [query]);
  return matches;
}

export function MonitorLayout(): React.ReactElement {
  const isNarrow = useMediaQuery('(max-width: 1024px)');
  const [sidebarOpen, setSidebarOpen] = useState(!isNarrow);

  useEffect(() => { setSidebarOpen(!isNarrow); }, [isNarrow]);

  return (
    <div style={{ display: 'flex', flexDirection: 'column', height: '100vh' }}>
      <NpLoadingBar />
      <Header onMenuToggle={() => setSidebarOpen(v => !v)} />

      <Drawer
        opened={sidebarOpen}
        openedStateMode={isNarrow ? 'overlap' : 'shrink'}
        closeOnOutsideClick={isNarrow}
        onOpenedChange={setSidebarOpen}
        position="left"
        render={() => <Sidebar onNavigate={() => isNarrow && setSidebarOpen(false)} />}
        maxSize={250}
      >
        <main>
          <Suspense fallback={<NpSuspenseLoading />}>
            <Outlet />
          </Suspense>
        </main>
      </Drawer>
    </div>
  );
}
```

App.tsx

```
export function App(): React.ReactElement {
  return (
    <ErrorBoundary>
      <BrowserRouter>
        <RouteContext.Provider value={appRoutes}>
          <NotificationProvider />
          <Routes>
            <Route path="/login" element={<Login />} />

            <Route
              path="/"
              element={
                <RequireAuth>
                  <MonitorLayout />
                </RequireAuth>
              }
            >
              {appRoutes.map((route) => (
                <Route
                  key={route.path}
                  path={route.path.replace(/^\//, '')}
                  element={
                    <FeatureGuard feature={route.feature}>
                      {route.element}
                    </FeatureGuard>
                  }
                </Route>
              ))}
            </Route>
          </Routes>
        </RouteContext.Provider>
      </BrowserRouter>
    </ErrorBoundary>
  );
}
```

CLAUDE.md

```
# CLAUDE.md
This file provides guidance to Claude Code (claude.ai/code) when working with code in this repository.
**For detailed architecture and system design**, see:
- [docs/ARCHITECTURE.md](docs/ARCHITECTURE.md) - Comprehensive system architecture
- [docs/PLUGIN_ARCHITECTURE.md](docs/PLUGIN_ARCHITECTURE.md) - Plugin system deep dive
**For frontend-specific guidance**, see:
- [Cockpit/CLAUDE.md](Cockpit/CLAUDE.md) - Frontend patterns, DevExtreme, Knockout.js, Vite workflows
This file focuses on **how to work with the code** (commands, workflows, patterns, pitfalls) and
**integration between backend and frontend**.

## Project Overview
NuPMES is a browser-based Manufacturing Execution System (MES) built with .NET 8.0 and TypeScript. It
consists of two main applications:
- **Monitor**: Management interface for supervisors and administrators (DevExtreme-based)
- **Terminal**: Simplified touch interface for production workers
The system uses a sophisticated plugin architecture where all major components are loaded as plugins
coordinated by NCM.Kernel.Plugin.

## Working with Claude Code
**Git Operations**: Claude Code should ALWAYS ask before committing or pushing changes.
- Never commit/push without explicit user confirmation
- Always ask: "Ready to commit these changes?" or "Should I push to remote?"
- Wait for user approval - developers often want to test first
- Give users time to verify changes locally before sharing them
This ensures developers maintain control over their git history and can test changes before they're
committed.

## Essential Build Commands
### Backend (.NET)
```bash
Build the standard solution
dotnet build NuPMESServer.sln -c Debug
Build with customer customizations
dotnet build NuPMESServer.Custom.sln -c Debug
Run tests for specific project
dotnet test Server/ServerTest/ServerTest.csproj /p:SolutionFileName=NuPMESServer.sln -c Test
Important: Always specify solution name when testing
dotnet test NuPMESServer.sln -c Test --no-build /p:SolutionFileName=NuPMESServer.sln
```
**Critical**: Tests require SQL Server and use Liquibase for database migrations. The test pipeline
sets up a temporary database via `cicd/setup-unittest-db.ps1`.

### Frontend (TypeScript/Vite)
```bash
cd Cockpit
Development server (includes IPC generation)
npm run dev
Build both Monitor and Terminal for production
npm run build:prod
Build individual applications
npm run build:monitor
npm run build:terminal
Type checking
npm run ts-check
Generate IPC contracts from .NET (runs automatically with dev/build)
node buildIpc.js
```
**IPC Generation**: The `buildIpc.js` script generates TypeScript interfaces from .NET contracts,
ensuring type-safe communication between frontend and backend. This runs automatically with `npm run
dev` and before production builds.

## Key Code Locations
> **For Architecture Details**: See [docs/ARCHITECTURE.md](docs/ARCHITECTURE.md) and
[docs/PLUGIN_ARCHITECTURE.md](docs/PLUGIN_ARCHITECTURE.md)

**Backend**:
- `ServerDefault/Data/Manager/`: Business logic (86 manager classes)
- `ServerDefault/Data/Domain/`: Domain entities with NHibernate attributes
- `ServerDefault/Data/Repository/`: Data access layer
- `ServerDefault/Data/Ipc/`: IPC contract definitions (C#)
- `ServerDefault/Rest/Controllers/`: REST API endpoints
- `ServerDefault/Rest/SignalR/Hubs/`: Real-time communication hub
```

```

**Frontend**:
- `Cockpit/Monitor/`: Management UI (DevExtreme-based)
- `Cockpit/Terminal/`: Shop floor UI (touch-optimized)
- `Cockpit/Shared/`: Common code, managers, generated IPC contracts
- `Cockpit/Shared/ipc/`: Generated TypeScript interfaces (auto-generated)

**Other**:
- `Server/ServerTest/`: Unit tests
- `Changelog/`: Liquibase database migrations
- `Configs/`: Plugin configuration XML files

### Manager Pattern (Important!)
Business logic is in **Manager** classes (not services). Always create managers via `UnitOfWork`:

```csharp
// [X] CORRECT - In controllers (via Container.DataPlugin)
Container.DataPlugin.ExecuteUnitOfWork(unitOfWork =>
{
 var itemManager = unitOfWork.CreateManager<ItemManager>(Container.Logger);
 var item = itemManager.GetById(itemId);
}, IsolationLevel.ReadCommitted, readOnly: false);

// [X] CORRECT - Standalone (with direct dataPlugin access)
dataPlugin.ExecuteUnitOfWork(unitOfWork =>
{
 var itemManager = unitOfWork.CreateManager<ItemManager>(logger);
 var item = itemManager.GetById(itemId);
});

// [X] WRONG - bypasses DI and UnitOfWork
var itemManager = new ItemManager();
```

### Dual Solution Strategy
- **NuPMESSTest.sln**: Standard product
- **NuPMESSTest.Custom.sln**: Customer-specific customizations

**Critical**: Always specify solution name in build/test commands to ensure correct conditional compilation:
```bash
dotnet test NuPMESSTest.sln -c Test /p:SolutionFileName=NuPMESSTest.sln
```

## Database Migrations
Database schema changes are managed with Liquibase. Migrations are stored in `Changelog/changelog.*.xml` files.

**Run migrations manually:**
```bash
Linux - Update main database
cd Changelog
./update.sh
Linux - Update test database
./update-unittest.sh
Windows - Update main database
cd Changelog
update.cmd
Windows - Update test database
updateUnitTest.cmd
```

**Important**: Migrations do NOT run automatically on application startup. You must run the update scripts manually when database schema changes are needed.

## Working with Agents

**Always use specialized agents for code changes** - do not edit code directly. Direct work is only acceptable for reading, searching, explaining, or updating documentation.



Work Type	Agent
Backend C# (managers, controllers, domain, API)	`backend-csharp-developer`
Frontend (UI, DevExtreme, Knockout, TypeScript, Vite)	`frontend-specialist`
Full-stack features (database + backend + frontend)	`fullstack-feature-developer`
Code review (after any code changes)	`code-reviewer`
Tests (unit tests, test strategy, QA)	`qa-expert`


```

```
---

## Development Workflow

### Adding New Features

1. **Business Logic**: Add/modify managers in `ServerDefault/Data/Manager/`
2. **Entities**: Update domain classes in `ServerDefault/Data/Domain/` with NHibernate attributes
3. **API**: Add/modify controllers in `ServerDefault/Rest/Controllers/`
4. **Frontend**:
  - Update IPC contracts in `ServerDefault/Data/Ipc/` if API changes
  - Modify views in `Cockpit/Monitor/` or `Cockpit/Terminal/`
  - Shared code goes in `Cockpit/Shared/`
5. **Tests**: Add NUnit tests in `Server/ServerTest/`

### Working with Tests

Tests are organized in `Server/ServerTest/` using:
- **NUnit** for test framework
- **Moq** for mocking
- **FluentAssertions** for assertions
- **AutoFixture** for test data generation

**Important**: Tests require SQL Server. The CI pipeline creates a temporary database using Liquibase migrations from `Changelog/`.

### SignalR Real-time Events

Send server-push notifications via [MesHub.cs](ServerDefault/Rest/SignalR/Hubs/MesHub.cs):

```csharp
// Backend - send event
await Clients.All.SendAsync("EventName", data);

// Frontend - handle event in SignalRManager.ts
hubConnection.on("EventName", (data) => { ... });
```

### Git & GitLab Operations

Use the `glab` command via the Bash tool for ALL GitLab-related tasks including working with merge requests, issues, pipelines, and API access.

**Important**: When given a GitLab URL, use the `glab` command to get the information needed instead of trying to fetch URLs directly.

Common operations:

```bash
View merge request details
glab mr view 123
Create new merge request
glab mr create --title "Feature: Title" --description "Description" --target-branch develop
List merge requests
glab mr list --state opened
glab mr list --author @me
View merge request changes
glab mr diff 123
Merge request operations
glab mr approve 123
glab mr merge 123
Issues
glab issue view 456
glab issue list --state opened
Pipelines
glab ci view
glab ci status
API access for custom queries
glab api projects/:id/merge_requests/123
```

**CI/CD Integration**: The repository uses GitLab CI/CD pipelines. Check `.gitlab-ci.yml` for pipeline configuration. Pipeline jobs are defined in `cicd/` directory.
```

Issue Tracking & Code Quality

****Jira Integration**:** The project uses Jira for issue tracking and project management.

- ****Ticket System**:** All features, bugs, and tasks are tracked in Jira
- ****Branch Naming Convention**:** Create branches per Jira ticket using the pattern ``feature/sf-xxxx`` (e.g., ``feature/sf-8212``)
 - The ``sf-xxxx`` corresponds to the Jira ticket ID
 - Use ``feature/`` prefix for new features and enhancements
 - Use ``bugfix/`` prefix for bug fixes
- ****Commit Message Convention**:** All commit messages must start with the Jira ticket number
 - Format: ``SF-XXXX: Commit message description``
 - Example: ``SF-8213: Fix value editor refresh on parameter change``
 - Example: ``SF-8212: Add cellTemplate to show readable names in view mode``
 - This enables automatic tracking of commits to their associated Jira tickets
- ****MCP Tools Available**:** Atlassian MCP server provides tools to interact with Jira:
 - Search and view Jira issues
 - Create and update issues
 - Add comments and worklogs
 - Check issue status and transitions
 - Link issues to merge requests

****SonarQube Integration**:** Code quality is monitored using SonarQube.

- ****Two Separate Projects**:** The repository has TWO SonarQube projects (defined in `[.gitlab-ci.yml](.gitlab-ci.yml)`):
 - ``NuPMES-Backend`` (line 2) - All C# code, .NET projects, backend logic
 - ``NuPMES-Frontend`` (line 5) - All TypeScript/JavaScript code, Cockpit apps
 - ****Context Detection**:** Use the current git branch to determine which project to check:
 - If working on backend code (``*.cs`` files) → use ``NuPMES-Backend``
 - If working on frontend code (``Cockpit/`` directory) → use ``NuPMES-Frontend``
 - ****Code Analysis**:** SonarQube analyzes code for quality issues, bugs, vulnerabilities, and code smells
 - ****Quality Gates**:** Pull requests must pass quality gate requirements
 - ****MCP Tools Available**:** SonarQube MCP server provides tools to:
 - Search for issues in projects (specify ``projectKey: "NuPMES-Backend"`` or ``"NuPMES-Frontend"``)
 - Get project quality gate status
 - View component measures (coverage, complexity, violations)
 - Analyze code snippets
 - Check system health and status
 - View rule details and metrics
 - ****Workflow Preference**:** When checking SonarQube issues, use ``get_project_quality_gate_status`` with the current branch to show ONLY new issues introduced in the branch. Don't list all existing issues in the code-base - focus on what the current changes introduced, not pre-existing issues.
- Use the MCP tools to check Jira tickets and SonarQube status directly during development to stay aligned with project requirements and code quality standards.

Task Management

For complex tickets, use ``.claude/tasks/`` for persistent task tracking across sessions. Templates are in ``.claude/tasks/_TEMPLATE/``.

****Workflow**:** Load ticket → analyze & plan → discuss plan with user → create task structure → execute with agents → resume later if needed.

Task structures contain ``README.md`` (overview + agent assignments), ``TODO.md`` (checklist), and ``IMPLEMENTATION_PLAN.md`` (step-by-step changes).

Technology Stack

> ****Detailed Stack & Rationale**:** See `[docs/ARCHITECTURE.md#technology-stack-rationale](docs/ARCHITECTURE.md#technology-stack-rationale)`

****Backend**:** .NET 8.0, ASP.NET Core, NHibernate 5.3.13, Ninject, Quartz.NET, SignalR, Log4net, Liquibase

****Frontend**:** Vite 6.3.5, TypeScript 5.8.3, DevExtreme 18.2.18, Knockout.js 3.5.1, jQuery 3.7.1, SignalR client

****Testing**:** NUnit, Moq, FluentAssertions, AutoFixture, Playwright

Common Pitfalls

1. ****Never use `new` for managers**:** Use ``unitOfWork.CreateManager<T>()`` or ``DependencyResolver.CreateInstance<T>()`` to enable CUSTOM overrides
2. ****Solution name required**:** Always specify ``/p:SolutionFileName=NuPMEServer.sln`` when building/testing to ensure correct conditional compilation
3. ****IPC auto-generates**:** IPC contracts generate automatically with ``npm run dev`` and builds. Only run ``node buildIpc.js`` manually if needed outside these workflows
4. ****Virtual for customization**:** Use ``virtual`` instead of ``private`` for methods that CUSTOM solutions might override
5. ****NHibernate attributes**:** Use ``NHibernate.Mapping.Attributes`` on domain classes (not Fluent API)
6. ****Test database**:** Tests require SQL Server and use Liquibase migrations from ``Changelog/``
7. ****DevExtreme version**:** Frontend uses DevExtreme 18.2.18 - check docs for that specific version

Agent frontend-specialist

```
---
name: frontend-specialist
description: "Use this agent when working on frontend code, UI components, or client-side functionality. This includes:\\n\\n- Implementing new UI features or components in Monitor or Terminal applications\\n- Refactoring or optimizing existing frontend code\\n- Debugging UI issues, rendering problems, or client-side errors\\n- Working with DevExtreme components, Knockout.js bindings, or jQuery interactions\\n- Building or modifying TypeScript interfaces and type definitions\\n- Implementing responsive designs or touch-optimized interfaces\\n- Setting up or troubleshooting Vite build configurations\\n- Creating or updating frontend tests with Playwright\\n\\n\\nExamples:\\n\\n<example>\\n\\nuser: \"Please create a new DevExtreme DataGrid component for displaying production orders in the Monitor app\"\\n\\nassistant: \"I'll use the Task tool to launch the frontend-specialist agent to implement this DataGrid component with proper TypeScript types and Knockout bindings.\"\\n\\n</example>\\n\\n<example>\\n\\nuser: \"The Terminal touch interface is laggy when scrolling through the operation list\"\\n\\nassistant: \"Let me use the Task tool to launch the frontend-specialist agent to investigate and optimize the scrolling performance in the Terminal app.\"\\n\\n</example>\\n\\n<example>\\n\\nContext: User has just modified IPC contracts in ServerDefault/Data/Ipc/\\n\\nuser: \"I've updated the ItemDto class to include a new field\"\\n\\nassistant: \"Since the IPC contracts were modified, I'll use the Task tool to launch the frontend-specialist agent to update the frontend code to consume the new field and regenerate TypeScript interfaces.\"\\n\\n</example>"
model: sonnet
color: green
memory: project
---
```

You are an elite frontend developer with deep expertise in modern web application development, specializing in the NuPMES technology stack. Your role is to architect, implement, and optimize user interfaces with a focus on performance, accessibility, and maintainability.

****Your Technical Domain:****

- ****Core Technologies****: JavaScript, TypeScript 5.8.3, DevExtreme 18.2.18, Knockout.js 3.5.1, jQuery 3.7.1
- ****Build Tools****: Vite 6.3.5, npm/Node.js ecosystem
- ****Testing****: Playwright for E2E testing
- ****Real-time Communication****: SignalR client for server push notifications
- ****Architecture****: Dual-app structure (Monitor management UI, Terminal touch-optimized UI) with shared code

****Project-Specific Context:****

You are working on NuPMES, a browser-based Manufacturing Execution System with two distinct frontend applications:

1. ****Monitor**** (`Cockpit/Monitor/``): Management interface using DevExtreme components for supervisors and administrators
2. ****Terminal**** (`Cockpit/Terminal/``): Simplified, touch-optimized interface for production workers on the shop floor
3. ****Shared**** (`Cockpit/Shared/``): Common managers, utilities, and auto-generated IPC contracts

****Critical Project Patterns:****

1. ****IPC Contract Generation****: TypeScript interfaces are AUTO-GENERATED from C# DTOs via `buildIpc.js`. Generated files live in `Cockpit/Shared/ipc/``. Never manually edit these files. If you need new types, they must be added to `ServerDefault/Data/Ipc/`` with `[Category("Ipc")]`` attribute in C#.
2. ****DevExtreme Version****: The project uses DevExtreme 18.2.18 specifically. Always reference documentation for this version, as newer versions may have breaking changes.
3. ****Knockout.js Bindings****: UI components use Knockout.js for two-way data binding. Follow the project's existing patterns for observables and computed values.
4. ****SignalR Integration****: Real-time updates are handled through `SignalRManager.ts` in shared code. Subscribe to events from the MesHub backend.
5. ****Build Commands****:
 - Development: `npm run dev` (includes IPC generation)
 - Production: `npm run build:prod` (builds both apps)
 - Type checking: `npm run ts-check`
 - IPC regeneration (manual): `node buildIpc.js`

****Your Responsibilities:****

1. ****Write Type-Safe Code****: Leverage TypeScript's type system fully. Use generated IPC contracts for backend communication. Avoid `any` types unless absolutely necessary.
2. ****Performance Optimization****:
 - Minimize DOM operations and reflows
 - Optimize DevExtreme DataGrid rendering for large datasets
 - Implement virtual scrolling and pagination where appropriate
 - Profile and eliminate unnecessary re-renders in Knockout bindings
 - Optimize touch interactions for Terminal app responsiveness
3. ****Accessibility****: Ensure UI components are keyboard navigable and screen reader compatible. Follow WCAG

```
4. **Code Organization**:
- Place Monitor-specific code in `Cockpit/Monitor/`
- Place Terminal-specific code in `Cockpit/Terminal/`
- Extract shared utilities, managers, and components to `Cockpit/Shared/`
- Follow existing file structure patterns
5. **DevExtreme Best Practices**:
- Configure grids for optimal performance with large datasets
- Use appropriate column types and custom cell templates
- Implement proper event handlers for user interactions
- Handle validation and error states gracefully
6. **State Management**:
- Use Knockout observables for reactive UI state
- Centralize business logic in manager classes (TypeScript)
- Keep components focused on presentation
- Handle async operations with proper loading states
7. **Error Handling**:
- Implement comprehensive error boundaries
- Provide user-friendly error messages
- Log errors appropriately for debugging
- Handle network failures and timeouts gracefully
8. **Testing**:
- Write Playwright tests for critical user flows
- Ensure tests are maintainable and reliable
- Mock backend dependencies appropriately
**Quality Assurance Checklist**
Before completing any frontend task, verify:
- [ ] TypeScript compilation passes (`npm run ts-check`)
- [ ] No console errors or warnings in development
- [ ] Responsive design works across target screen sizes
- [ ] Touch interactions work smoothly on Terminal (no delays, proper feedback)
- [ ] Loading states and error states are handled
- [ ] IPC contracts are properly typed and used
- [ ] Code follows existing patterns in the codebase
- [ ] DevExtreme components are configured optimally
- [ ] Knockout bindings are efficient (no unnecessary subscriptions)
- [ ] SignalR events are properly subscribed/unsubscribed
**When You Need Clarification**
- If requirements are ambiguous, ask for specific user scenarios
- If performance targets are unclear, request measurable criteria
- If backend IPC contracts are missing, specify what DTOs are needed
- If DevExtreme component behavior is uncertain, reference version 18.2.18 docs
**Edge Cases to Consider**
- Network connectivity loss during critical operations
- Large datasets (1000+ rows) in grids
- Concurrent updates from multiple users via SignalR
- Touch vs mouse input handling in Terminal
- Browser compatibility (target modern evergreen browsers)
- Localization and internationalization requirements
**Your Communication Style**
- Explain technical decisions with rationale
- Provide code examples that follow project conventions
- Reference specific files and line numbers when discussing existing code
- Suggest performance optimizations proactively
- Flag potential accessibility issues early
You are proactive in identifying potential issues, suggesting improvements, and ensuring that frontend code is not just functional but exemplifies best practices in modern web development within the NuPMES architecture.
**Update your agent memory** as you discover frontend patterns, component implementations, performance optimizations, and architectural decisions. This builds up institutional knowledge across conversations. Write concise notes about what you found and where.
Examples of what to record:
- Common DevExtreme configuration patterns for grids, forms, and other components
- Knockout.js binding patterns used across Monitor and Terminal
- SignalR event handling patterns and subscription management
- Performance optimization techniques that worked well
- Reusable component locations and their purposes
- Touch interaction patterns specific to Terminal app
- IPC contract usage patterns and common data flows
- Build configuration customizations and why they exist
```

Erklärung zur Verwendung maschineller Hilfsmittel

Zu Erzeugung der Beispiele im Testsystem wurde Claude Opus 4.6 [28] und Claude Opus 4.7 [29] genutzt im Terminal in VS Studio [70].

Zur Rechtschreibprüfung wurde LanguageTool [68] verwendet.

Zur Übersetzung von einzelnen Fachbegriffen wurde DeepL[69] verwendet.



Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit in allen Teilen selbstständig angefertigt und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel benutzt habe, und dass die Arbeit in gleicher oder ähnlicher Form in noch keiner anderen Prüfung vorgelegen hat. Mir ist bewusst, dass ich Autor/in der vorliegenden Arbeit bin und volle Verantwortung für den Text trage. ☒

Ich erkläre, dass ich wörtlich oder sinngemäß aus anderen Werken – dazu gehören auch Internetquellen – übernommene Inhalte als solche kenntlich gemacht und die entsprechenden Quellen angegeben habe. ☒

Mir ist bewusst, dass meine Arbeit auf Plagiate überprüft werden kann. Mir ist bekannt, dass es sich bei der Abgabe eines Plagiats um ein schweres akademisches Fehlverhalten handelt und dass Täuschungen nach der für mich gültigen Prüfungsordnung geahndet werden. ☒

Zusätzlich versichere ich, dass ich auf künstlicher Intelligenz (KI) basierende Werkzeuge nur in Absprache mit den Prüfern verwendet habe. Dabei stand meine eigene geistige Leistung im Vordergrund, und ich habe jederzeit den Prozess steuernd bearbeitet. ☒

Diese Werkzeuge habe ich im Quellenverzeichnis in der Rubrik „Übersicht verwendeter Hilfsmittel“ mit ihrem Produktnamen und einer Übersicht des im Rahmen dieser Prüfungs-/Studienarbeit genutzten Funktionsumfangs unter Angabe der Textstelle in der Arbeit vollständig aufgeführt. ☒

Ich versichere, dass ich keine KI-basierten Tools verwendet habe, deren Nutzung die Prüfer explizit schriftlich ausgeschlossen haben. Ich bin mir bewusst, dass die Verwendung von Texten oder anderen Inhalten und Produkten, die durch KI-basierte Tools generiert wurden, keine Garantie für deren Qualität darstellt. ☒

Ich verantworte die Übernahme jeglicher von mir verwendeter maschinell generierter Passagen vollumfänglich selbst und trage die Verantwortung für eventuell durch die KI generierte fehlerhafte oder verzerrte Inhalte, fehlerhafte Referenzen, Verstöße gegen das Datenschutz- und Urheberrecht oder Plagiate. ☒

Schwarzenberg, 22.04.26

Ort, Datum

Unterschrift