

Projektdokumentation

Abschlussprüfung Fachinformatiker Anwendungsentwicklung

Entwicklung einer webbasierten Zeiterfassungs- und Aufgabenverwaltungsanwendung

ZeitmanagementTool

Prüfling	Rolands Burmberger
Ausbildungsberuf	Fachinformatiker Anwendungsentwicklung
Projektzeitraum	80 Stunden (IHK-Rahmen)
Abgabedatum	Mai 2026
Prüfende Stelle	Industrie- und Handelskammer

Inhaltsverzeichnis

1. Einleitung	1
1.1 Projektbezeichnung	1
1.2 Kurzbeschreibung des Projekts	1
1.3 Projektumfeld	1
1.4 Projektziele	1
1.5 Projektschnittstellen	2
1.6 Zeitliche Eingrenzung	2
2. Analysephase	3
2.1 Ist-Analyse	3
2.2 Anforderungsanalyse	3
2.2.1 Funktionale Anforderungen	3
2.2.2 Nichtfunktionale Anforderungen	4
2.3 Abgrenzung des Projektumfangs	4
2.4 Wirtschaftlichkeitsbetrachtung und Make-or-Buy-Entscheidung	4
2.5 Use-Case-Diagramme	7
2.5.1 Benutzer- und Rollenverwaltung	7
2.5.2 Aufgabenverwaltung	8
2.5.3 Aufgabenansicht	8
2.5.4 Statusworkflow	9
2.5.5 Zeiterfassung	10
2.5.6 Tagesbericht	11
2.5.7 Kommentare und Benachrichtigungen	11
3. Entwurfsphase	12
3.1 Soll-Konzept	12
3.2 Technische Architektur	12
3.2.1 Architekturüberblick	12
3.2.2 Verwendeter Technologie-Stack	12

3.2.3 Projektstruktur	13
3.3 Datenmodell	13
3.3.1 Datenbankdiagramm – ER-Modell	14
3.3.2 Klassendiagramm der Domänenmodelle.....	14
3.3.3 Zentrale Entitäten und ihre Bedeutung	16
3.3.4 Wichtige Beziehungen	16
3.4 Rollen- und Rechtekonzept	16
3.4.1 Rollenübersicht	17
3.4.2 Rechtemodell	17
3.4.3 Datenschutz und Verarbeitung personenbezogener Daten	17
3.4.4 Abwehr typischer Angriffsvektoren	18
3.5 Entwurf der Benutzeroberfläche	19
3.5.1 Aufgabenübersicht	19
3.5.2 Aufgaben-Detailansicht	20
3.5.3 Tagesbericht-Maske.....	20
3.5.4 Tagesbericht-Übersicht	21
3.6 Aktivitätsdiagramme der Zeiterfassung	22
3.6.1 Normaler Ablauf der Zeiterfassung.....	22
3.6.2 Behandlung unterbrochener Zeiterfassungen.....	23
3.7 Architekturentscheidungen	25
3.7.1 MVC-Architektur.....	25
3.7.2 ASP.NET Core Identity.....	25
3.7.3 Entity Framework Core mit MySQL	25
3.7.4 Soft-Archivierung statt Hard-Delete	26
3.7.5 SignalR für Benachrichtigungen	26
4. Umsetzungsphase	28
4.1 Einrichtung der Entwicklungsumgebung	28
4.2 Einrichtung der Datenbankbindung.....	28
4.3 Umsetzung der Benutzer- und Rollenverwaltung	28

4.4 Umsetzung der Kundenverwaltung	29
4.5 Umsetzung der Aufgabenverwaltung	29
4.6 Umsetzung der Tagesberichte	30
4.6.1 Kalenderlogik	30
4.6.2 Automatische Status- und Buchungslogik	30
4.7 Umsetzung der Zeiterfassung	30
4.7.1 Statuszustände	31
4.7.2 Unterbrechungslogik	31
4.7.3 Synchronisation mit Tagesbericht	31
4.7.4 Overlay und Präzisionsverbesserung	31
4.8 Umsetzung der Kommentarfunktion	31
4.9 Umsetzung der Archivierung	32
4.9.1 Doppelte Absicherung	32
4.10 Umsetzung des Benachrichtigungssystems	33
4.10.1 NotificationService	33
4.10.2 SignalR-Hub	33
4.10.3 Dropdown im Layout	33
4.11 Demo- und Vorführdaten	33
5. Testphase	34
5.1 Testkonzept	34
5.2 Testumgebung	34
5.3 Testmatrix (Auszug der wichtigsten Testfälle)	34
5.4 Testdurchführung (beispielhaftes Testprotokoll)	36
5.5 Testergebnisse - Zusammenfassung	37
5.6 Testzyklus-Abschluss	38
5.7 Gefundene Defekte und Behebung	39
5.8 Nachgelagerte QA-Findings aus dem Stage-Test	41
6. Einführung und Übergabe	44
6.1 Deployment	44

6.2 Einweisung der Nutzer.....	45
6.3 Hinweise zur Inbetriebnahme	45
7. Soll-Ist-Vergleich	46
7.1 Stundenaufwand.....	46
7.2 Zielerreichung.....	47
7.3 Abweichungen gegenüber dem Projektantrag	47
8. Fazit und Ausblick	49
8.1 Fazit	49
8.2 Herausforderungen im Projektverlauf	49
8.3 Projektstärken	49
8.4 Ausblick und mögliche Erweiterungen	50
8.5 Persönliches Resümee.....	50
9. Anhang	i
9.1 Abkürzungsverzeichnis.....	i
9.2 Abbildungsverzeichnis.....	ii
9.3 Tabellenverzeichnis.....	ii
9.4 Literatur- und Quellenverzeichnis	iii
9.5 Eidesstattliche Erklärung	iii
9.6 Quellcode-Auszüge	iii
9.6.1 NotificationService – Deduplizierung über EreignisSchlüssel	iii
9.6.2 BenachrichtigungHub – SignalR-Hub mit Rollen-Gruppenzuordnung.....	v
9.6.3 ZeiterfassungenController – Start/Pause/Stop mit PauseGesamtSekunden- Akkumulation	vi
9.6.4 ZeiterfassungKonfliktService – Verhindern paralleler Zeiterfassungen	vii
9.6.5 AufgabenAutoArchivierungHostedService – Hintergrunddienst für automatische Archivierung.....	ix
9.6.6 CanAccessTask – Rollenbasierte Zugriffsprüfung auf Aufgabenebene	xi

1. Einleitung

1.1 Projektbezeichnung

Entwicklung einer webbasierten Zeiterfassungs- und Aufgabenverwaltungsanwendung (im Folgenden ZeitmanagementTool genannt).

1.2 Kurzbeschreibung des Projekts

Das Projekt umfasst die Konzeption, Entwicklung und Testung einer prototypischen Webanwendung, welche die Zeiterfassung von Arbeitszeiten sowie die Verwaltung von Aufgaben innerhalb eines mittelständischen IT-Dienstleisters integriert. Die Anwendung basiert auf ASP.NET Core 8 mit dem MVC-Muster, Entity Framework Core als objektrelationalem Mapper und einer MySQL-Datenbank. Die Benutzerauthentifizierung erfolgt über ASP.NET Core Identity mit den Rollen Admin, Manager und Employee.

Kernfunktionen sind die zentrale Verwaltung von Aufgaben mit Kundenzuordnung, die Echtzeit-Zeiterfassung mit Start-, Pause- und Stopp-Funktion, die automatische Synchronisation in Tagesberichte sowie ein rollenbasiertes Benachrichtigungssystem mit SignalR-Integration für Live-Updates.

1.3 Projektumfeld

Das Projekt wurde als internes Entwicklungsprojekt im Rahmen eines betrieblichen Verbesserungsvorhabens bei einem mittelständischen IT-Dienstleister mit rund 50 Mitarbeitern durchgeführt. Der Einsatzbereich der geplanten Anwendung ist die unternehmensweite Arbeitszeit- und Aufgabenverwaltung.

Auftraggeber ist die interne IT-Abteilung in Abstimmung mit der Personal- und Fachabteilung. Die Entwicklung erfolgte als Prototyp, welcher nach Fertigstellung von Abteilungsleitern (Manager) und Mitarbeitern im Unternehmen getestet und intern eingesetzt werden soll.

1.4 Projektziele

Das Projekt verfolgt folgende Hauptziele:

- Integration von Zeiterfassung und Aufgabenverwaltung in einer einheitlichen Webanwendung
- Reduzierung von Medienbrüchen zwischen Excel-Zeiterfassung und E-Mail-basierter Aufgabenverteilung
- Transparenter Echtzeit-Überblick über den Arbeitsfortschritt für das Management
- Automatische Generierung von Tagesberichten aus erfassten Arbeitszeiten
- Rollenbasiertes Rechtekonzept mit sauberer Trennung der Zuständigkeiten

- Kommentar- und Benachrichtigungssystem zur Verbesserung der Team-Kommunikation

1.5 Projektschnittstellen

Folgende Schnittstellen sind im Projekt relevant:

- Fachliche Schnittstelle: Abstimmung mit Abteilungsleitern zur Anforderungsaufnahme
- Technische Schnittstelle: Anbindung an MySQL-Datenbankserver über Entity Framework Core mit Pomelo-Provider
- Benutzerschnittstelle: Webbasierte Razor-Views mit Bootstrap 5 und jQuery
- Echtzeitschnittstelle: SignalR-Hub für Live-Benachrichtigungen an angemeldete Benutzer

1.6 Zeitliche Eingrenzung

Das Projekt wurde innerhalb des von der IHK vorgegebenen Rahmens von 80 Stunden durchgeführt. Die Stundenverteilung entspricht der im Projektantrag vorgesehenen Planung:

Projektphase	Geplante Stunden
Analyse	8 Stunden
Entwurf	12 Stunden
Implementierung	36 Stunden
Test	8 Stunden
Einführung und Übergabe	4 Stunden
Projektdokumentation	12 Stunden
Summe	80 Stunden

2. Analysephase

2.1 Ist-Analyse

Vor Projektbeginn wurde der aktuelle Zustand der Zeit- und Aufgabenverwaltung im Unternehmen erhoben. Dabei wurden folgende Schwachstellen festgestellt:

- Arbeitszeiten werden dezentral in Excel-Tabellen erfasst, was zu Inkonsistenzen und fehlerhaften Einträgen führt
- Aufgaben werden per E-Mail oder mündlich an Mitarbeiter übergeben, ohne zentrale Nachverfolgbarkeit
- Rückmeldungen zum Bearbeitungsstand erfolgen unsystematisch, wodurch das Management keinen transparenten Echtzeit-Überblick besitzt
- Tagesberichte werden manuell nachgeschrieben, was einen Medienbruch und Doppelerfassung bedeutet
- Automatisierungen wie Benachrichtigungen bei neuen Kommentaren oder Statusänderungen fehlen vollständig

Die wachsende Mitarbeiterzahl hat diese Problematik verstärkt. Das Management hat daher den Bedarf für eine integrierte Lösung erkannt.

Die Erhebung der Schwachstellen erfolgte in Abstimmung mit der Geschäftsleitung sowie dem Projektbetreuer und stützt sich auf drei Quellen: erstens zwei leitfadengestützte Einzelgespräche mit Abteilungsleitern von jeweils rund 30 Minuten, zweitens eine kurze schriftliche Rückmeldung aus der Belegschaft (Rücklauf 12 von 18 angesprochenen Mitarbeitern) und drittens eine Sichtung der aktuell verwendeten Excel-Vorlagen und E-Mail-Abläufe.

Aus den Gesprächen und Rückmeldungen ergaben sich folgende, für die Anforderungsanalyse genutzte Anhaltspunkte: Mitarbeiter berichteten von durchschnittlich rund 10 Minuten pro Arbeitstag für die nachgelagerte Pflege der Excel-Zeiterfassung und das Nachtragen von Tagesberichten; Abteilungsleiter schätzten ihren Aufwand für das manuelle Zusammentragen des Bearbeitungsstandes auf etwa 2 bis 3 Stunden pro Woche. Diese Zahlen sind als grobe Selbsteinschätzungen zu verstehen, nicht als gemessene Werte, bilden jedoch den Mindestnutzen der angestrebten Lösung ab und werden in Abschnitt 2.3 zur Eingrenzung des Projektumfangs wieder aufgegriffen.

2.2 Anforderungsanalyse

2.2.1 Funktionale Anforderungen

Die funktionalen Anforderungen wurden aus den Gesprächen mit den Abteilungsleitern und den identifizierten Schwachstellen der Ist-Analyse abgeleitet:

- Benutzer- und Rollenverwaltung: Anlegen, Bearbeiten und Deaktivieren von Benutzern; Verwaltung der Rollen Admin, Manager und Employee

- Kundenverwaltung: CRUD-Funktionalität für Kunden mit Archivierungs- und Wiederherstellungsmöglichkeit
- Aufgabenverwaltung: Anlegen, Bearbeiten und Zuweisen von Aufgaben mit Status, Priorität, Fälligkeit und Kundenbezug
- Zeiterfassung: Start-, Pause-, Fortsetzen- und Stopp-Funktion mit Unterbrechungslogik
- Tagesberichte: Tägliche Dokumentation mit Kalenderübersicht, automatischer Status- und Buchungslogik
- Kommentare: Aufgabenbezogene Kommunikation mit automatischer Benachrichtigung
- Benachrichtigungen: Persistente Speicherung und Live-Anzeige über SignalR
- Archivierung: Soft-Archivierung für Aufgaben und Kunden mit Wiederherstellungsmöglichkeit

2.2.2 Nichtfunktionale Anforderungen

- Benutzerfreundlichkeit: Intuitive, selbsterklärende Oberfläche mit konsistenter Navigation
- Sicherheit: Rollenbasierte Zugriffskontrolle und zusätzliche objektbezogene Rechteprüfungen
- Wartbarkeit: Klare Schichtung nach dem MVC-Muster zur Trennung von Fachlogik, Datenmodell und UI
- Performance: Antwortzeiten bei typischen Masken unter zwei Sekunden
- Datenkonsistenz: Relationale Integrität durch Fremdschlüsselbeziehungen und transaktionales Verhalten
- Revisionssicherheit: Nachvollziehbarkeit durch Soft-Archivierung statt Hard-Delete

2.3 Abgrenzung des Projektumfangs

Um den Projektumfang im vorgegebenen Zeitrahmen zu halten, wurden bewusst folgende Funktionen ausgeschlossen:

- Mobile Anwendung: Keine separate native iOS- oder Android-App; Nutzung ausschließlich über Webbrowser
- Externe Schnittstellen: Keine Integration mit Lohnabrechnungs- oder ERP-Systemen; das System ist als geschlossenes internes System konzipiert
- Erweiterte Reportings: Keine langfristigen Auswertungen über Wochen oder Monate, keine grafischen Dashboards
- Umfangreiches Projektmanagement: Keine Urlaubsverwaltung, Schichtplanung, Budgetierung oder Kundenabrechnung
- Mandantenfähigkeit: Die Anwendung ist für einen einzelnen Mandanten ausgelegt

2.4 Wirtschaftlichkeitsbetrachtung und Make-or-Buy-Entscheidung

Vor Beginn der Umsetzung wurde die Frage abgewogen, ob der fachliche Bedarf durch eine am Markt verfügbare Standardlösung gedeckt werden könnte (Buy) oder ob eine Eigenentwicklung sinnvoller ist (Make). Geprüft wurden als Vergleichsobjekte Clockify in Kombination mit einem einfachen Projekt-Tool, Toggl Track mit Asana sowie Jira Software. Die folgende Gegenüberstellung stellt die wesentlichen Kriterien dar.

Kriterium	Standard-Software (Buy)	Eigenentwicklung (Make)
Lizenzkosten (50 Nutzer, 1 Jahr)	ca. 4.500 – 9.000 € pro Jahr (Jira Software Standard bzw. Toggl + Asana Business, je Nutzer und Monat)	keine laufenden Lizenzkosten; einmaliger Entwicklungsaufwand im Rahmen der Ausbildung
Initialer Einführungsaufwand	Einrichtung, Schulung, Anpassung der Workflows; Integration zweier Tools für Zeit und Aufgaben	Entwicklung, Test und Einweisung innerhalb des IHK-Projektrahmens von 80 Stunden
Fachliche Passung	Großer Funktionsumfang, aber keine direkte Verknüpfung von Zeiterfassung, Aufgabe und Tagesbericht mit automatischer Auto-Booking-Logik	Passgenau auf den erhobenen Prozess zugeschnitten; fachliche Besonderheiten wie Tagesbericht-Marker und Sonderstatus direkt abgebildet
Datenhoheit / Datenschutz	Cloudbasiert; Auftragsverarbeitungsvertrag und Prüfung der Datenverarbeitung außerhalb des EWR erforderlich	Vollständig im eigenen Netz betrieben; keine externe Datenverarbeitung
Wartung / Weiterentwicklung	Bereitgestellt durch Anbieter; Abhängigkeit von Produkt-Roadmap und Preispolitik	Eigenverantwortung; erfordert interne Entwicklungskapazität, aber volle Kontrolle

Rein rechnerisch ließe sich der erwartete Produktivitätsgewinn bereits im ersten Jahr darstellen: Bei 50 Mitarbeitern und einer angenommenen Entlastung von durchschnittlich 10 Minuten pro Arbeitstag ergibt sich im Jahr ein Einsparpotenzial im vierstelligen Stundenbereich. Dem stehen im Make-Szenario der einmalige Entwicklungsaufwand des Ausbildungsprojekts und ein laufender interner Wartungsaufwand gegenüber; im Buy-Szenario laufende Lizenzkosten in vergleichbarer Größenordnung sowie Aufwände für Einrichtung, Integration und Schulung.

Ausschlaggebend für die Entscheidung zur Eigenentwicklung waren weniger die reinen Kosten als drei fachliche Gründe: Erstens erfordert der betriebliche Prozess eine enge Verknüpfung von Aufgabe, Zeiterfassung und Tagesbericht inklusive Auto-Booking-Logik, die in den geprüften Standardprodukten so nicht abbildbar ist. Zweitens bleiben personenbezogene Arbeitszeitdaten vollständig im internen Netz, was im Datenschutz-Kontext ein gewichtiges Argument ist. Drittens entspricht die Umsetzung als Ausbildungsprojekt dem Ziel, die konkrete Problemstellung im Hause zu lösen und dabei den betrieblichen Lerneffekt zu nutzen. Der entwickelte Prototyp ist in diesem Sinne keine dauerhafte Festlegung gegen Standard-Software, sondern eine für den aktuellen Bedarf passgenaue Lösung, die bei wachsendem Funktionsumfang später durch eine zugekaufte Lösung abgelöst werden kann.

2.5 Use-Case-Diagramme

Die zentralen Anwendungsfälle wurden in UML-Use-Case-Diagrammen modelliert, um die Interaktion der verschiedenen Rollen mit dem System zu visualisieren.

2.5.1 Benutzer- und Rollenverwaltung

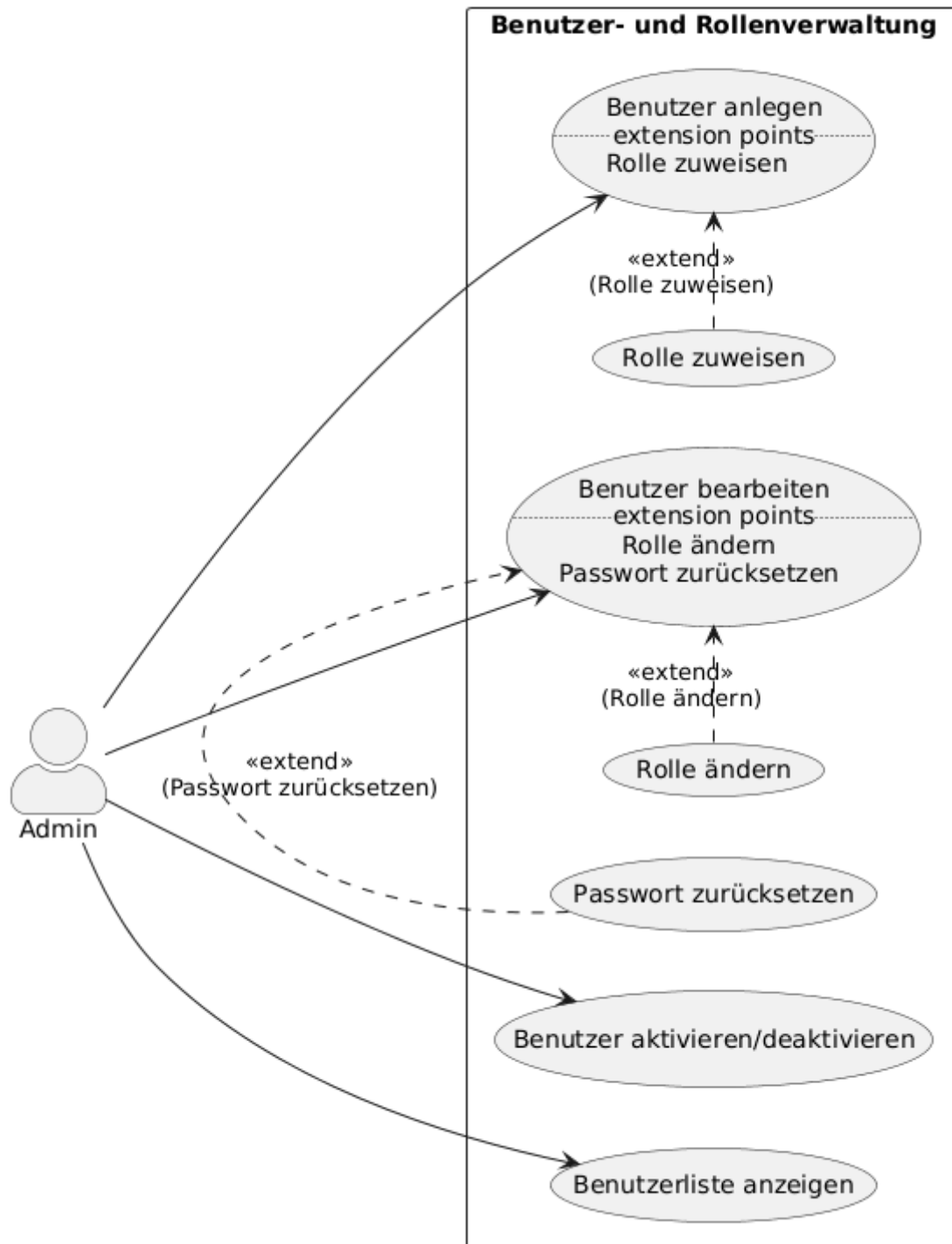


Abbildung 1: Use-Case-Diagramm Benutzer- und Rollenverwaltung

2.5.2 Aufgabenverwaltung

Use-Case-Diagramm: Aufgabenverwaltung (Entwurfsstand)

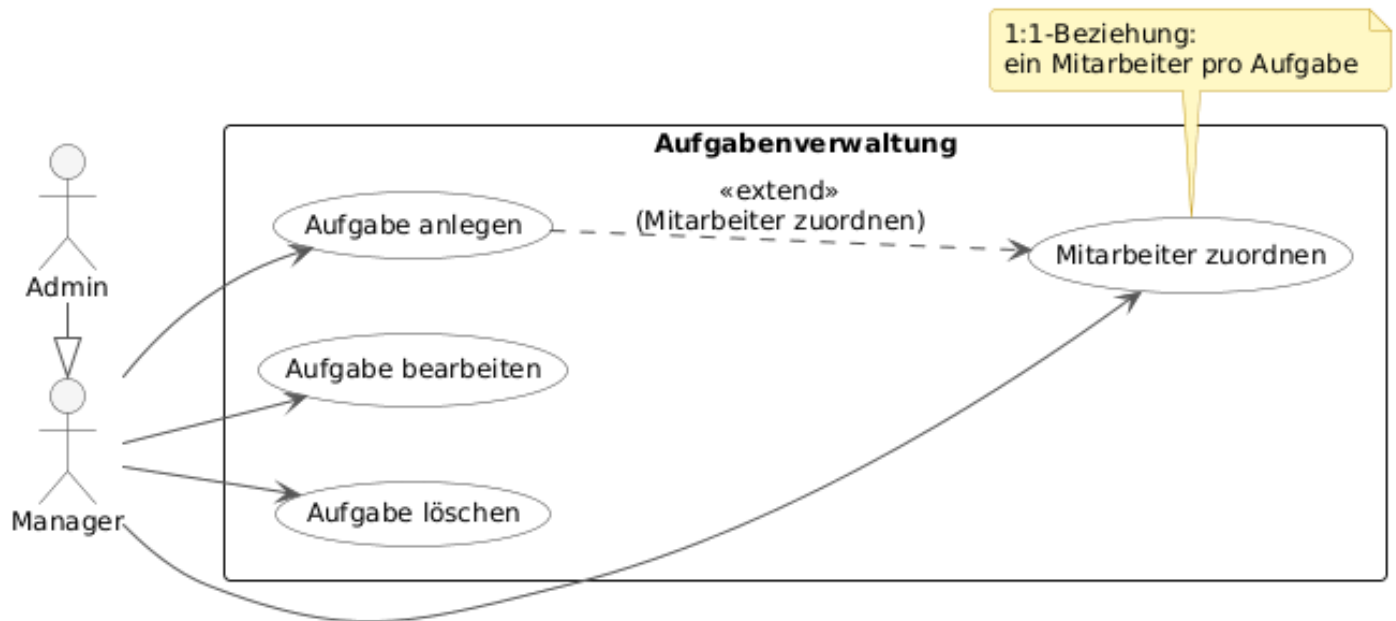


Abbildung 2: Use-Case-Diagramm Aufgabenverwaltung (Entwurfsstand)

2.5.3 Aufgabenansicht

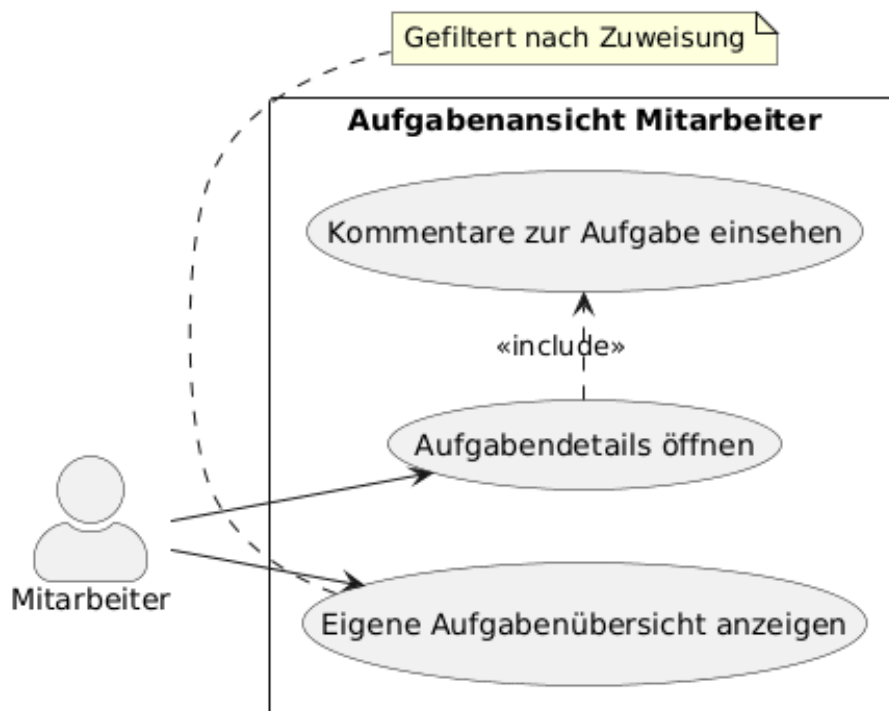


Abbildung 3: Use-Case-Diagramm Aufgabenansicht

2.5.4 Statusworkflow

Use-Case-Diagramm: Statusworkflow (Entwurfsstand)

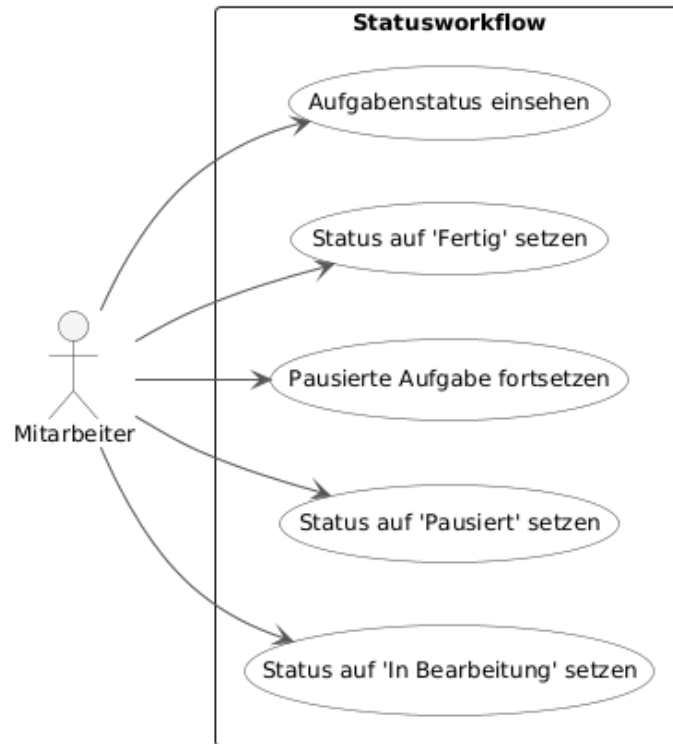


Abbildung 4: Use-Case-Diagramm Statusworkflow (Entwurfsstand)

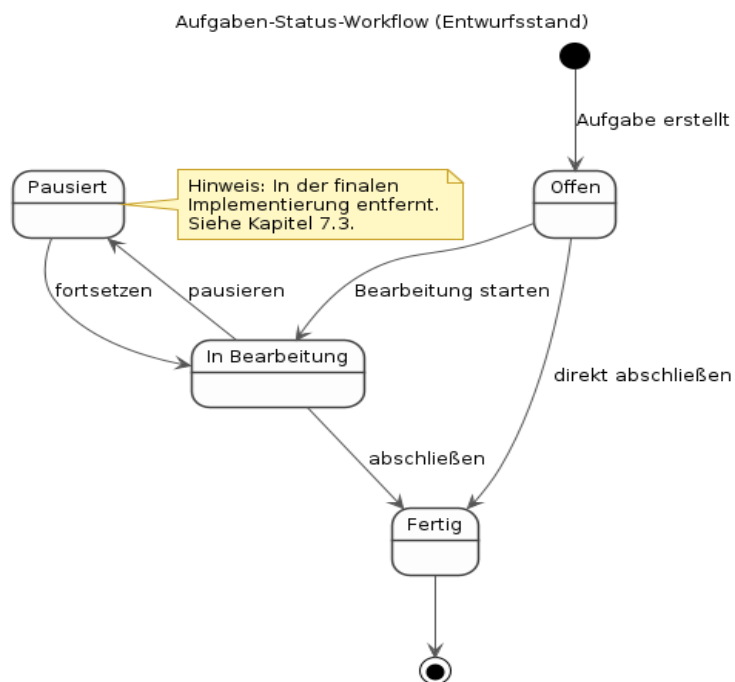


Abbildung 5: Zustandsdiagramm Aufgaben-Status-Workflow (Entwurfsstand)

2.5.5 Zeiterfassung

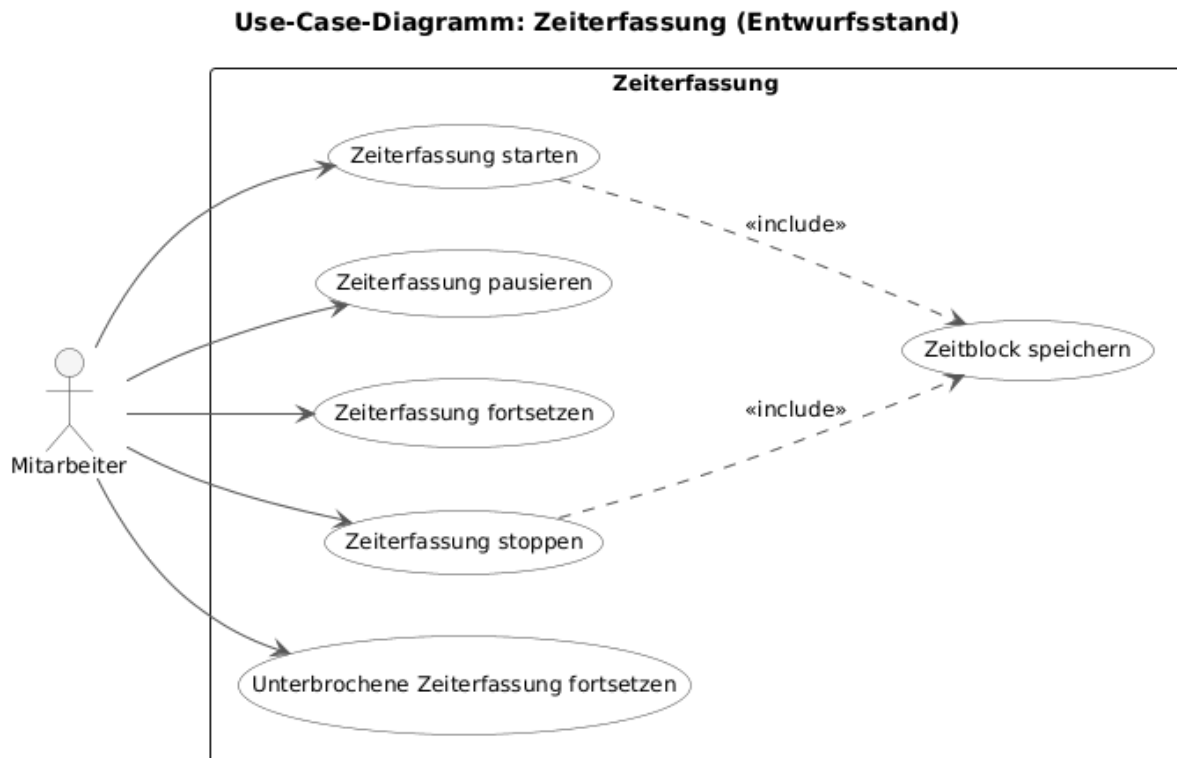


Abbildung 6: Use-Case-Diagramm Zeiterfassung (Entwurfsstand)

2.5.6 Tagesbericht

Use-Case-Diagramm: Tagesbericht (Entwurfsstand)

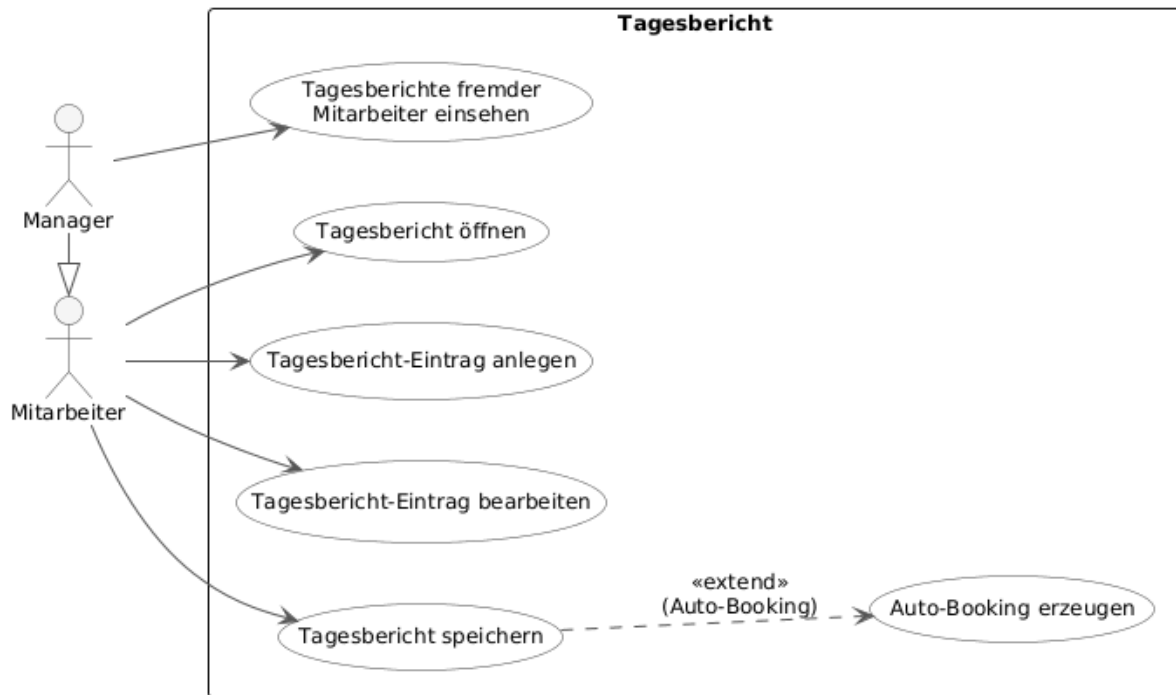


Abbildung 7: Use-Case-Diagramm Tagesbericht (Entwurfsstand)

2.5.7 Kommentare und Benachrichtigungen

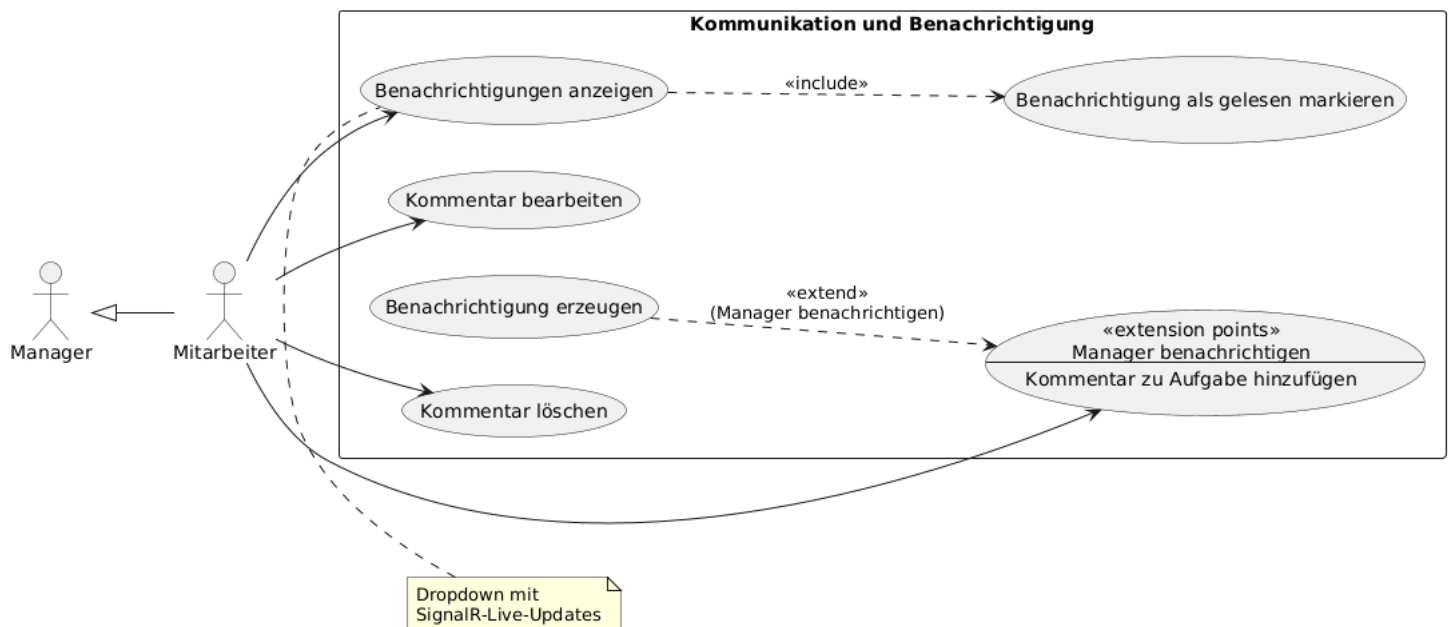


Abbildung 8: Use-Case-Diagramm Kommentare und Benachrichtigungen

3. Entwurfsphase

3.1 Soll-Konzept

Auf Basis der Ist-Analyse und der erhobenen Anforderungen wurde das Soll-Konzept der Anwendung entwickelt. Das ZeitmanagementTool soll eine integrierte Lösung bieten, in der Aufgaben, Arbeitszeiten und tägliche Dokumentation in einer einzigen Webanwendung zusammengeführt werden.

Das Zielbild gliedert sich in drei Ebenen. Erstens benötigen Mitarbeiter eine alltagstaugliche Oberfläche, in der sie ihre zugewiesenen Aufgaben sehen und bearbeiten können. Zweitens benötigt die Organisation eine belastbare Zeiterfassungs- und Nachweislogik. Drittens benötigt das Management eine strukturierte Sicht auf offene Vorgänge, Fortschritt, Prioritäten und Fälligkeiten. Diese drei Ebenen stehen nicht nebeneinander, sondern ergänzen sich gegenseitig.

3.2 Technische Architektur

3.2.1 Architekturüberblick

Die Anwendung folgt dem klassischen MVC-Muster (Model-View-Controller) und ist als serverseitig gerenderte Webanwendung konzipiert. Die Entscheidung für MVC und gegen eine Single-Page-Application wurde getroffen, weil die Anwendung klassische Fachmasken, Formulare, Listen, Detailansichten, Validierungen und rollenbasierte Navigation abbildet.

Die Architektur ist in drei Schichten aufgeteilt:

- Präsentationsschicht: Razor-Views mit Bootstrap 5 und jQuery für die Benutzeroberfläche
- Logikschicht: Controller für die Verarbeitung von Anfragen sowie Service-Klassen für Querschnittslogik
- Datenhaltungsschicht: Entity Framework Core mit MySQL-Provider Pomelo für die Persistenz

3.2.2 Verwendeter Technologie-Stack

Komponente	Technologie / Version
Laufzeitumgebung	.NET 8 (LTS)
Programmiersprache	C# 12
Web-Framework	ASP.NET Core MVC
View-Engine	Razor Views

Komponente	Technologie / Version
ORM	Entity Framework Core 8
Datenbank	MySQL 8 mit Pomelo-Provider
Authentifizierung	ASP.NET Core Identity
Echtzeit-Kommunikation	SignalR
Frontend-Framework	Bootstrap 5, jQuery
IDE	Microsoft Visual Studio 2022
Versionsverwaltung	Git
Datenbank-Tooling	MySQL Workbench

3.2.3 Projektstruktur

Das Projekt folgt dem typischen Aufbau einer ASP.NET Core MVC-Anwendung. Die wichtigsten Verzeichnisse sind:

- Controllers: Enthält die Controller-Klassen für die einzelnen Fachbereiche
- Models: Beinhaltet die Domänenmodelle wie Aufgabe, Zeiterfassung, Tagesbericht, Kommentar
- Views: Razor-Views für die Benutzeroberfläche, gegliedert nach Controllern
- ViewModels: Spezielle Models zur Datenübergabe zwischen Controller und View
- Data: Enthält den ApplicationDbContext und die Seeder-Klassen
- Services: Querschnittslogik wie NotificationService, AufgabenArchivService und der Hosted Service
- Hubs: Die SignalR-Hubs für Echtzeit-Kommunikation
- Constants: Zentrale Konstanten wie Rollennamen

3.3 Datenmodell

Das Datenmodell bildet die fachlichen Kernobjekte des Projekts ab und wurde so entworfen, dass es einen durchgängigen Geschäftsprozess unterstützt. Die Hauptentitäten sind Benutzer, Rolle, Kunde, Aufgabe, Zeiterfassung, Tagesbericht, TagesberichtEintrag, Kommentar und Benachrichtigung.

3.3.1 Datenbankdiagramm – ER-Modell

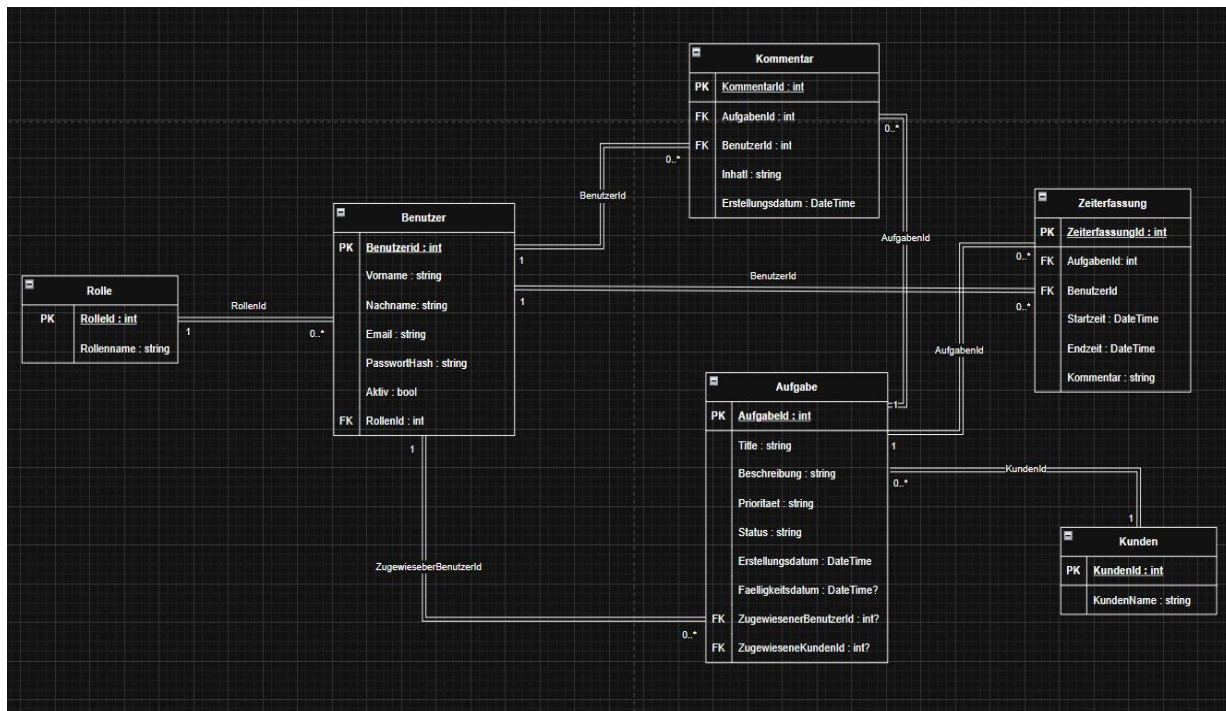


Abbildung 9: Datenbankdiagramm der Kerntabellen (Entwurfsstand)

3.3.2 Klassendiagramm der Domänenmodelle

Das folgende Klassendiagramm stellt den finalen Stand der Domänenmodelle dar. Es ergänzt das Datenbankdiagramm um Details, die in der relationalen Sicht nicht sichtbar sind: die Aufteilung in Tagesbericht und TagesberichtEintrag, die Ableitung des ApplicationUser von IdentityUser (Rollen über ASP.NET Identity), die Pausenakkumulation in der Zeiterfassung sowie die Soft-Archivierung über IsArchived und ArchivedAt bei Kunde und Aufgabe. Die Kardinalitäten der Beziehungen sind jeweils direkt am Pfeil annotiert.

Klassendiagramm - ZeitmanagementTool

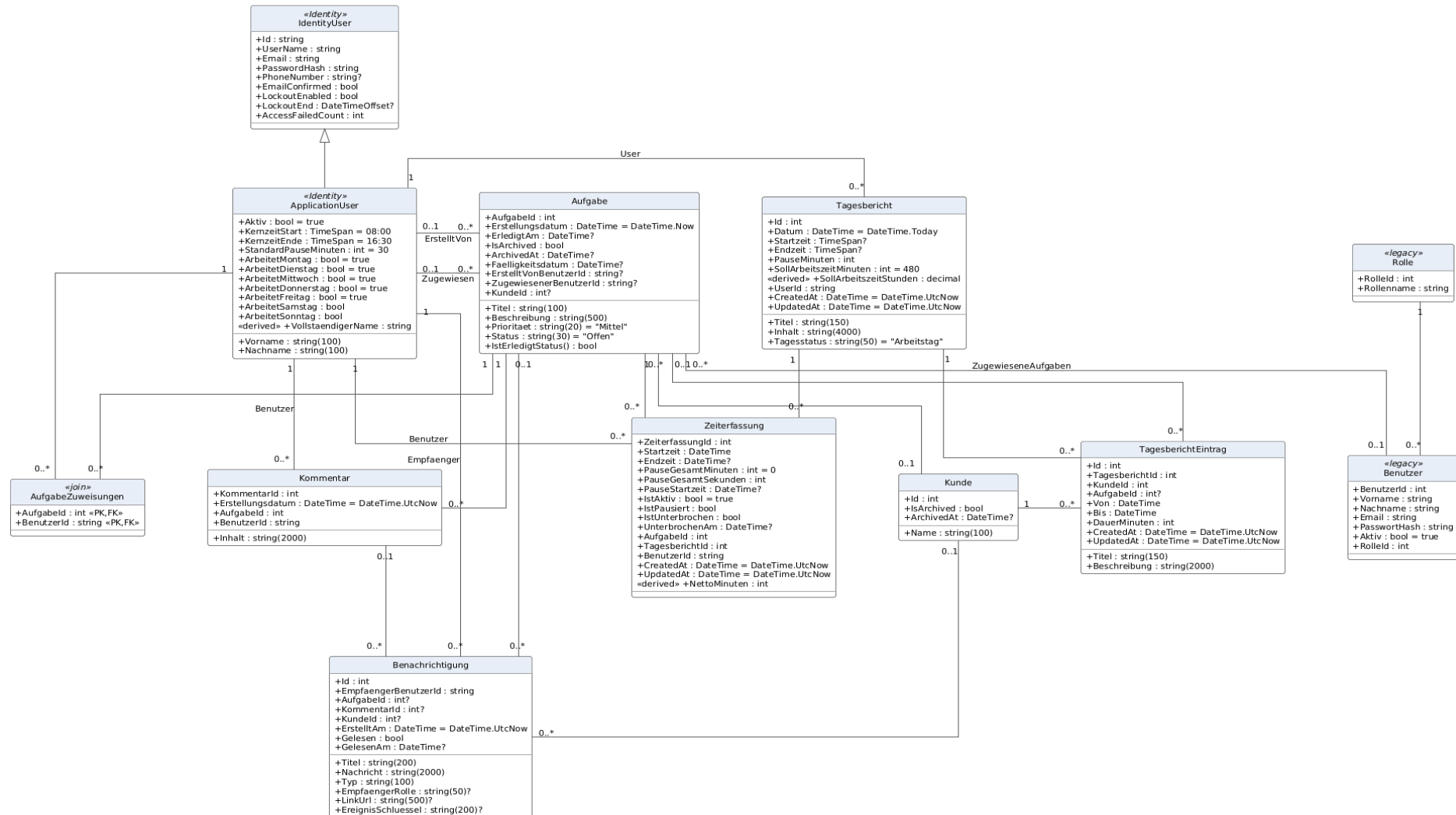


Abbildung 10: Klassendiagramm der Domänenmodelle

3.3.3 Zentrale Entitäten und ihre Bedeutung

Aufgabe: Das zentrale operative Objekt. Enthält Titel, Beschreibung, Priorität, Status, Fälligkeitsdatum, Erstellungsdatum, Kundenbezug und Zuweisung. Aufgaben sind der Haupteinstiegspunkt für Mitarbeiter in die tägliche Arbeit und bilden die Klammer für Kommentare, Statuswechsel, Zeiterfassung und Historie.

Kunde: Ermöglicht eine fachliche Zuordnung von Aufgaben zu externen oder internen Kontexten. Dadurch lässt sich eine Aufgabe nicht nur als isolierte Aktivität betrachten, sondern in einem nachvollziehbaren Geschäftsbezug führen.

Zeiterfassung: Bildet reale Arbeitszeitsegmente ab. Eng verknüpft mit Benutzer und Aufgabe, läuft über Statuszustände wie aktiv, pausiert oder unterbrochen. Zeiterfassungen sind nicht losgelöste Arbeitszeitbuchungen, sondern operative Zeiten, die direkt aus der Aufgabebearbeitung entstehen.

Tagesbericht: Das tägliche Dokumentationsobjekt. Steht für einen Kalendertag und einen bestimmten Benutzer. Enthält einzelne TagesberichtEinträge, die manuell, automatisch oder aus der Zeiterfassung synchronisiert entstehen können.

TagesberichtEintrag: Ein einzelner Arbeitsblock innerhalb eines Tagesberichts mit Kunde, optionaler Aufgabe, Start- und Endzeit sowie Beschreibung und berechneter Dauer.

ApplicationUser: Erweitert den ASP.NET Core Identity-Benutzer um fachliche Angaben wie Vorname, Nachname und Aktiv-Status.

Kommentar: Textnotiz zu einer Aufgabe mit Autorbezug über BenutzerId.

Benachrichtigung: Ereignisobjekt pro Empfänger mit Titel, Nachricht, Typ, Link, Lese-Status und Ereignisschlüssel zur Duplikatvermeidung. Optionale Referenzen auf Aufgabe, Kommentar oder Kunde.

3.3.4 Wichtige Beziehungen

- ApplicationUser 1:n Tagesbericht (Pflicht, Restrict)
- Tagesbericht 1:n TagesberichtEintrag (Cascade)
- Kunde 1:n Aufgabe (optionaler Fremdschlüssel in Aufgabe, Restrict)
- Kunde 1:n TagesberichtEintrag (Pflicht, Restrict)
- Aufgabe 1:n Zeiterfassung (Cascade)
- Tagesbericht 1:n Zeiterfassung (Cascade)
- ApplicationUser 1:n Zeiterfassung (Restrict)
- Aufgabe 1:n Kommentar (Cascade)
- Aufgabe n:m ApplicationUser über die Join-Tabelle AufgabeZuweisungen
- Benachrichtigung n:1 ApplicationUser als Empfänger (Cascade)

3.4 Rollen- und Rechtekonzept

3.4.1 Rollenübersicht

Das System arbeitet mit drei Rollen, die zentral in der Konstantenklasse RoleNames.cs definiert sind:

Rolle	Zweck	Berechtigungen
Admin	Administration	Vollständige Rechte: Benutzerverwaltung, Rollenverwaltung, Einstellungen, alle Aufgaben, alle Kunden, alle Tagesberichte
Manager	Führung / Teamleitung	Verwaltung von Kunden, Aufgaben (alle), Einsicht in alle Tagesberichte, Bearbeitung von Statusänderungen
Employee	Mitarbeiter	Nur eigene Aufgaben, eigene Zeiterfassungen, eigene Tagesberichte; Kommentare und Zeiterfassung auf zugewiesenen Aufgaben

3.4.2 Rechtemodell

Das Rechtekonzept basiert auf einer zweistufigen Absicherung. Zunächst wird auf Controller-Ebene per Authorize-Attribut eine rollenbasierte Mindestberechtigung geprüft. Anschließend erfolgt eine zusätzliche objektbezogene Prüfung, damit Mitarbeiter wirklich nur die Datensätze sehen und bearbeiten können, die ihnen zugewiesen sind. Entsprechende Hilfsmethoden wie CanAccessTask, CanAccessReport und CanAccessTimeEntry sind in den jeweiligen Controllern implementiert.

3.4.3 Datenschutz und Verarbeitung personenbezogener Daten

Die Anwendung verarbeitet personenbezogene Daten im Sinne der DSGVO, insbesondere Namen und E-Mail-Adressen der Mitarbeiter sowie Arbeitszeitdaten, aus denen sich Rückschlüsse auf das Arbeitsverhalten ziehen lassen. Im Entwurf wurden daher folgende Grundsätze berücksichtigt: Datenminimierung (es werden nur Felder erfasst, die für Zeiterfassung und Aufgabenverwaltung fachlich nötig sind), Zweckbindung (keine Übermittlung an Dritte, kein Profiling außerhalb des fachlichen Kontexts), Zugriffsbeschränkung (objektbezogene Prüfung, so dass Mitarbeiter nur eigene Arbeitszeiten sehen), sowie Speicherbegrenzung (Soft-Archivierung mit Zeitstempel als Grundlage für Löschkonzepte nach Ablauf gesetzlicher Aufbewahrungsfristen).

Die Anwendung wird im internen Firmennetz betrieben, personenbezogene Daten verlassen das Unternehmensnetz nicht. Zugriff erhalten ausschließlich authentifizierte und rollenzugeordnete Beschäftigte. Für den produktiven Einsatz ist eine Abstimmung mit dem Betriebsrat

sowie eine Verarbeitungstätigkeit im Verzeichnis gem. Art. 30 DSGVO vorgesehen; diese Schritte liegen außerhalb des technischen Projektumfangs, werden aber im Rahmen der Einführung begleitet.

3.4.4 Abwehr typischer Angriffsvektoren

Ergänzend zur rollen- und objektbezogenen Zugriffskontrolle werden die typischen Angriffsvektoren einer serverseitig gerenderten Webanwendung adressiert:

- Cross-Site Request Forgery (CSRF): ASP.NET Core MVC gibt für alle Formulare standardmäßig Antiforgery-Tokens aus („__RequestVerificationToken“), die bei jedem POST serverseitig geprüft werden. Alle schreibenden Actions sind mit dem Attribut [ValidateAntiForgeryToken] belegt beziehungsweise laufen über das Standardverhalten der Form-Helper, die das Token automatisch mitsenden.
- Cross-Site Scripting (XSS): Die Ausgabe von Nutzereingaben – insbesondere in Kommentaren und Aufgabenbeschreibungen – erfolgt über die Razor-Standardausgabe mit @model.Feld, die HTML automatisch enkodiert. Html.Raw wird im Projekt für Nutzerinhalte bewusst nicht verwendet. Dadurch werden Tags wie <script> aus Kommentartexten als Text dargestellt und nicht ausgeführt.
- SQL-Injection: Sämtliche Datenbankzugriffe erfolgen über Entity Framework Core, das die erzeugten Abfragen parametrisiert. Direkte String-Konkatenation mit Nutzereingaben in SQL-Anweisungen findet nicht statt, Methoden wie FromSqlRaw werden nicht verwendet.
- Passwortsicherheit: ASP.NET Core Identity speichert Passwörter nicht im Klartext, sondern über einen Hash mit PBKDF2 und Salt. Die Passwortregeln wurden gegenüber den Identity-Defaults gehärtet: Mindestlänge 8 Zeichen sowie Pflicht für Groß- und Kleinbuchstaben, Ziffern und Sonderzeichen. Ergänzend ist eine Kontosperrung nach fünf fehlgeschlagenen Anmeldeversuchen mit fünf Minuten Sperrzeit aktiv. Für eine produktive Nutzung ließe sich die Mindestlänge weiter auf 10 oder 12 Zeichen anheben.
- Authentifizierung und Sitzung: Die Anmeldung erfolgt über Cookie-Authentifizierung durch Identity; der Session-Cookie ist als HttpOnly gekennzeichnet und wird im produktiven Betrieb zusätzlich mit dem Secure-Flag ausgeliefert, sobald die Anwendung hinter TLS bereitgestellt wird.
- Insecure Direct Object Reference (IDOR): Die objektbezogenen Prüfmethode CanAccessTask, CanAccessReport und CanAccessTimeEntry sind genau für dieses Muster gedacht: Sie verhindern, dass ein angemeldeter Mitarbeiter durch bloßes Austauschen der Id in der URL auf fremde Datensätze zugreifen kann, auch wenn sein Authorize-Attribut ihm die Action grundsätzlich erlauben würde.

3.5 Entwurf der Benutzeroberfläche

Die Benutzeroberfläche wurde auf Übersichtlichkeit und einfache Bedienbarkeit ausgelegt. Da die Anwendung im Arbeitsalltag eingesetzt werden soll, ist eine intuitive Nutzung ein wichtiger Faktor für die Akzeptanz durch die Benutzer. Für die wichtigsten Seiten wurden Mockups erstellt.

3.5.1 Aufgabenübersicht



Abbildung 11: Mockup Aufgabenübersicht

3.5.2 Aufgaben-Detailansicht

Aufgabe 1

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et

Priorität 2

Erstellt von Manager
Max Musterman

Zugewiesene Mitarbeiter

FS SE GE RB
ER AP SP

Status:
In Bearbeitung

Max Musterman

Saved Time
DateTime: 01.01.2026 - 08:10 bis 12:45

Max Musterman

Kommentar
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy

Kommentar erstellen

Start Stop

Abbildung 12: Mockup Aufgaben-Detailansicht

3.5.3 Tagesbericht-Maske

Tagesbericht
01.01.2026

Kalender

1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20

Schnellauswahl:

Aufgabe hinzufügen Krank Feiertag Berufsschule Urlaub

Zeit: 08:00 bis 17:15 | Pause: 01:00 | Gesamte Arbeitsstunden: 08:15h

01.01.2026

Datum	Tag	Zeit	Stunden	Kunde
01.01.2026	Montag	08:00 bis 09:30	01:30h	Max Musterman GmbH
01.01.2026	Montag	09:30 bis 12:00	02:30h	Max Musterman GmbH
01.01.2026	Montag	13:00 bis 17:15	04:15h	Max Musterman GmbH

Abbildung 13: Mockup Tagesbericht-Maske

3.5.4 Tagesbericht-Übersicht

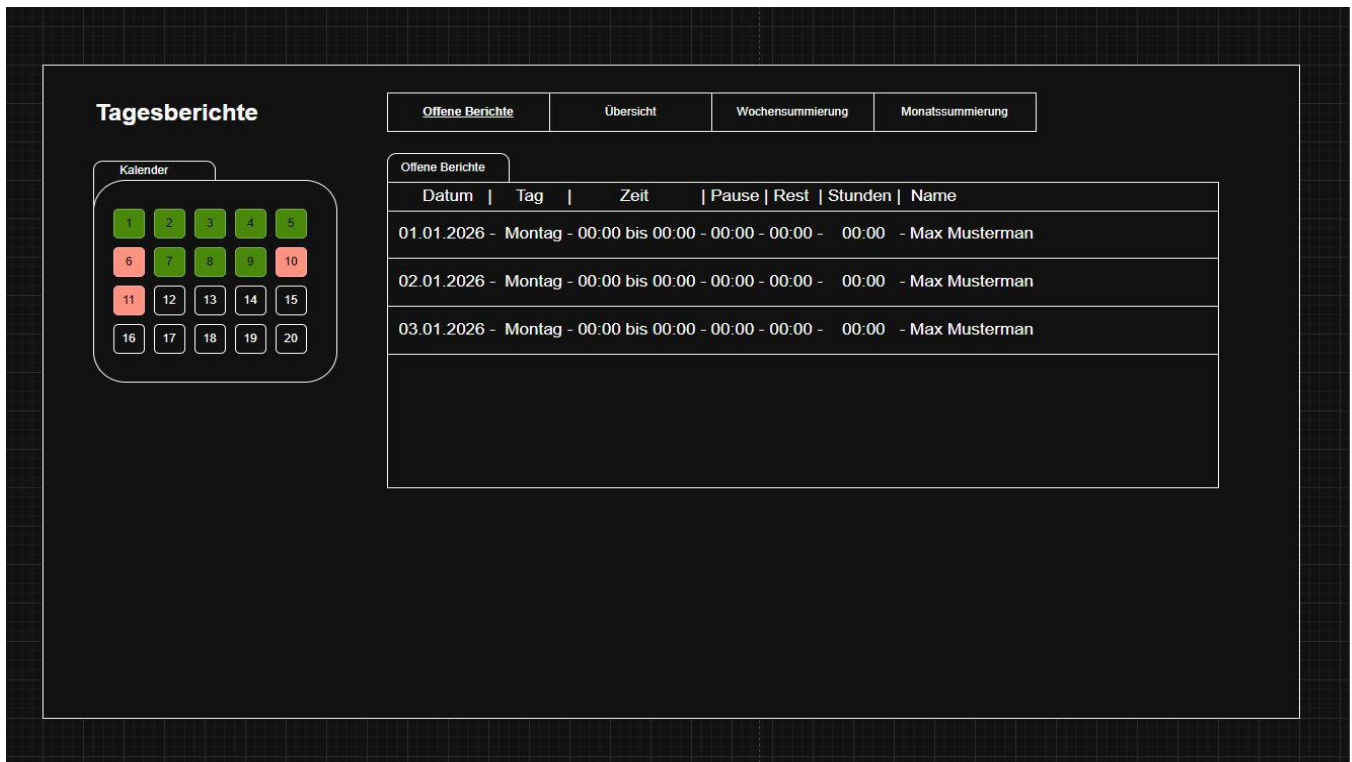


Abbildung 14: Mockup Tagesbericht-Übersicht

3.6 Aktivitätsdiagramme der Zeiterfassung

Die Zeiterfassung ist fachlich das komplexeste Modul der Anwendung. Aus diesem Grund wurden die wichtigsten Abläufe in Aktivitätsdiagrammen modelliert.

3.6.1 Normaler Ablauf der Zeiterfassung

Aktivitätsdiagramm - Normaler Ablauf der Zeiterfassung

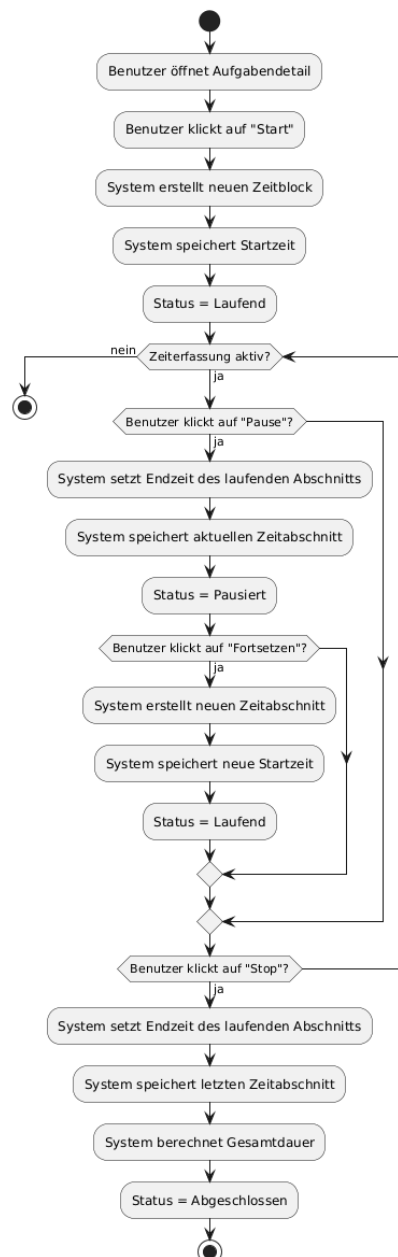


Abbildung 15: Aktivitätsdiagramm - Normaler Ablauf der Zeiterfassung

3.6.2 Behandlung unterbrochener Zeiterfassungen

Aktivitätsdiagramm - Behandlung unterbrochener Zeiterfassungen

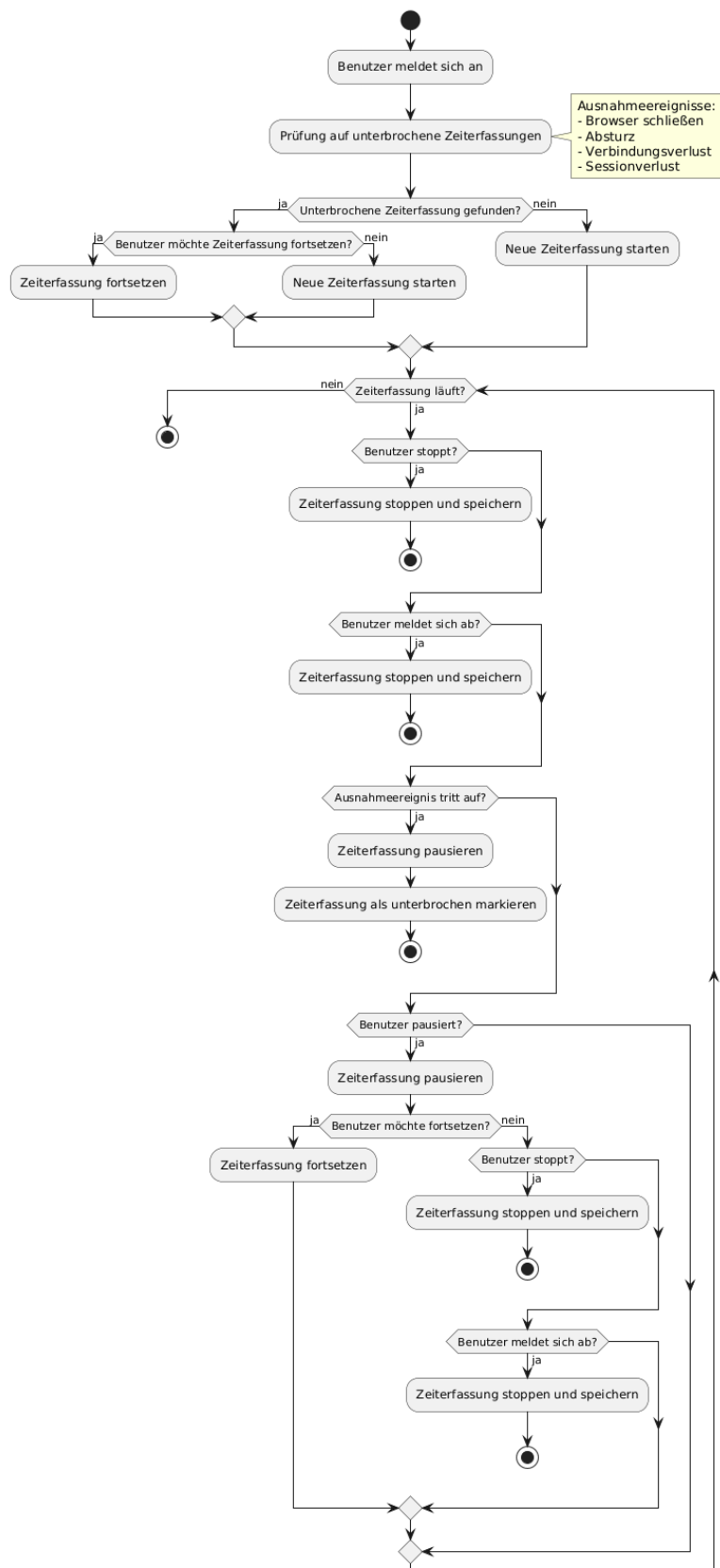


Abbildung 16: Aktivitätsdiagramm - Behandlung unterbrochener Zeiterfassungen

Ergänzend zu den Aktivitätsdiagrammen zeigt das folgende Sequenzdiagramm den Zusammenspiel der beteiligten Komponenten über die Zeit: Browser, ZeiterfassungenController, der KonfliktService zur Prüfung paralleler Erfassungen, der ApplicationDbContext sowie – für die Stop-Benachrichtigung an Vorgesetzte – NotificationService und BenachrichtigungHub.

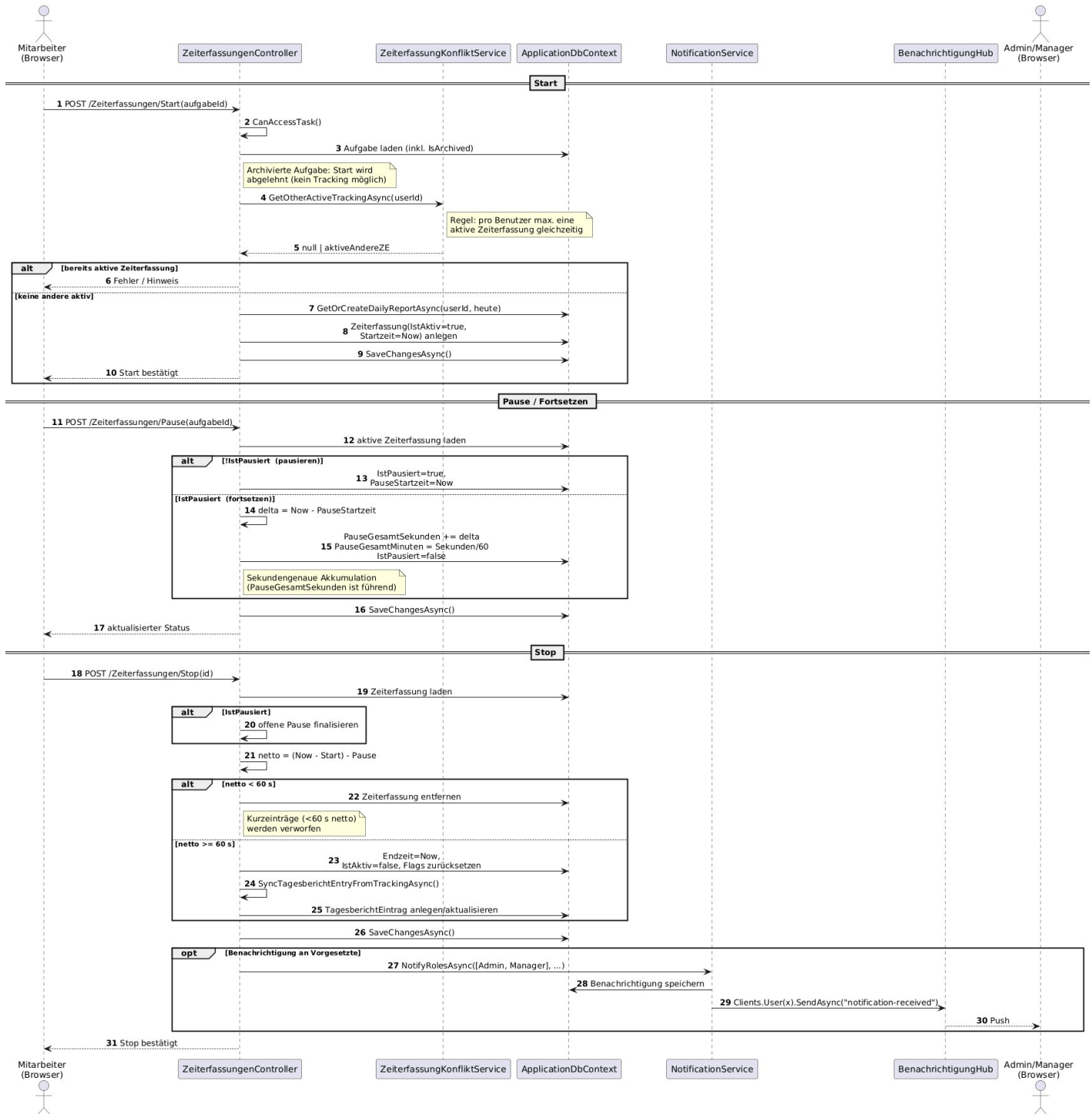


Abbildung 17: Sequenzdiagramm - Zeiterfassung Start / Pause / Stop

3.7 Architekturentscheidungen

Während der Entwurfsphase wurden mehrere grundlegende Architekturentscheidungen getroffen, welche den weiteren Projektverlauf prägen:

3.7.1 MVC-Architektur

Die Entscheidung für das MVC-Muster fiel gegen Alternativen wie Razor Pages oder eine Single-Page-Application mit Blazor beziehungsweise React. Die klassische Trennung von Controllern, Models und Views bietet für ein klassisches Enterprise-Fachsystem den besten Kompromiss aus Strukturklarheit und Umsetzungsgeschwindigkeit und ist im Fachgespräch leicht erklärbar.

Razor Pages wären für kleinere seitenbasierte Anwendungen naheliegend gewesen, werden jedoch bei einer wachsenden Zahl rollenbasierter Aktionen pro Fachobjekt unübersichtlicher, weil jede Seite ihre eigene Code-Behind-Datei führt. Im vorliegenden Fall mit mehreren CRUD-Bereichen, einer objektbezogenen Rechteprüfung und einer gemeinsamen Service-schicht erleichtert die Controller-Sicht von MVC die Wiederverwendung von Aktionen und das saubere Gruppieren verwandter Endpunkte.

Eine SPA mit Blazor oder React wurde verworfen, weil sie für klassische Fachmasken mit Formularen, Validierungen und Listen einen überproportionalen Mehraufwand bedeutet hätte: zusätzliche API-Schicht, separates Build-Tooling, eigenes State-Management, komplexere Authentifizierung über Tokens. Für den IHK-Projektrahmen von 80 Stunden ist serverseitig gerendertes MVC hier klar wirtschaftlicher. Die einzige Einschränkung – weniger hochinteraktive Client-Logik – wurde gezielt durch punktuellen Einsatz von jQuery und SignalR kompensiert.

3.7.2 ASP.NET Core Identity

Die Nutzung von ASP.NET Core Identity anstelle einer eigenen Authentifizierungslogik wurde bewusst gewählt. Das Framework bietet aus dem Stand sichere Authentifizierung, Passwort-Hashing, Rollenverwaltung und Cookie-Handling. Eine Eigenentwicklung wäre fehleranfälliger und sicherheitstechnisch deutlich riskanter.

3.7.3 Entity Framework Core mit MySQL

Entity Framework Core wurde als ORM gewählt, weil es eine starke Integration in .NET bietet, Migrationen versionssicher abgebildet und die relationale Modellierung über Code-First direkt aus den Domänenklassen heraus erfolgen kann. Als Datenbank-Provider kommt Pomelo.EntityFrameworkCore.MySql zum Einsatz, der eine robuste MySQL-Anbindung ermöglicht.

Im .NET-Umfeld wäre Microsoft SQL Server der technisch näherliegende Kandidat gewesen, wurde aber aus zwei Gründen nicht gewählt: Erstens ist MySQL im Unternehmen bereits für

andere Anwendungen etabliert und wird vom Betrieb bereits administriert, zweitens fallen für die angepeilte Mehrbenutzerumgebung keine zusätzlichen Lizenzkosten an, während beim SQL Server über die Express-Edition hinaus Lizenzgebühren zu erwarten wären. PostgreSQL wäre funktional eine gleichwertige Alternative gewesen, bringt im konkreten Umfeld aber keinen Vorteil, der den Wechsel der bestehenden Datenbank-Landschaft rechtfertigt. Der Pomelo-Provider ist für MySQL eine stabile und aktiv gepflegte Implementierung und wird in der offiziellen EF-Core-Dokumentation als Standardprovider für MySQL geführt.

3.7.4 Soft-Archivierung statt Hard-Delete

Für Aufgaben und Kunden wurde die Soft-Archivierung anstelle eines physischen Löschsens gewählt. Archivierte Datensätze werden über das Feld `IsArchived` mit zugehörigem `ArchivedAt`-Zeitstempel markiert und aus den aktiven Listen ausgeblendet, bleiben aber über eigene Archivansichten sichtbar. Dies erhält die Historie, verhindert verwaiste Fremdschlüssel und ist fachlich wesentlich realistischer.

3.7.5 SignalR für Benachrichtigungen

Für das Benachrichtigungssystem wurde SignalR als Echtzeit-Kommunikationslayer gewählt. Die Alternative wäre ein clientseitiges Polling gewesen, was aber deutlich mehr Ressourcen verbraucht hätte. Benachrichtigungen werden zusätzlich persistent in der Datenbank gespeichert, damit sie auch nach Seitenwechseln und erneuter Anmeldung sichtbar bleiben.

Als zweite Alternative wurden Server-Sent Events (SSE) betrachtet, die für ein reines Push-Szenario ausreichend gewesen wären, weil der Server in diesem Anwendungsfall die Richtung vorgibt und der Client nur empfängt. Für SignalR sprachen trotzdem drei Punkte: die tiefe Integration in ASP.NET Core (Authentifizierung, Gruppenmechanik, Reconnect-Logik sind ohne Zusatzaufwand verfügbar), die Fähigkeit, automatisch auf den jeweils besten Transport zurückzufallen (WebSockets, wenn möglich; andernfalls Long-Polling) sowie die Möglichkeit, Nachrichten rollenbasiert an SignalR-Gruppen zu senden, was für das rollenbasierte Benachrichtigungsmodell fachlich genau passt. Klassisches HTTP-Polling wurde verworfen, weil es bei 50 angemeldeten Mitarbeitern und einem typischen Intervall von wenigen Sekunden unverhältnismäßig viele Leerabfragen erzeugt hätte.

Das folgende Sequenzdiagramm zeigt exemplarisch den Ablauf einer Live-Benachrichtigung bei einem Statuswechsel durch einen Mitarbeiter: vom Verbindungsaufbau mit rollenbasierter Gruppenzuordnung über die Erzeugung und Persistierung der Benachrichtigung bis zur Push-Zustellung an die Empfänger (Admin und Manager) und die abschließende Aktualisierung des Benachrichtigungs-Dropdowns im Browser.

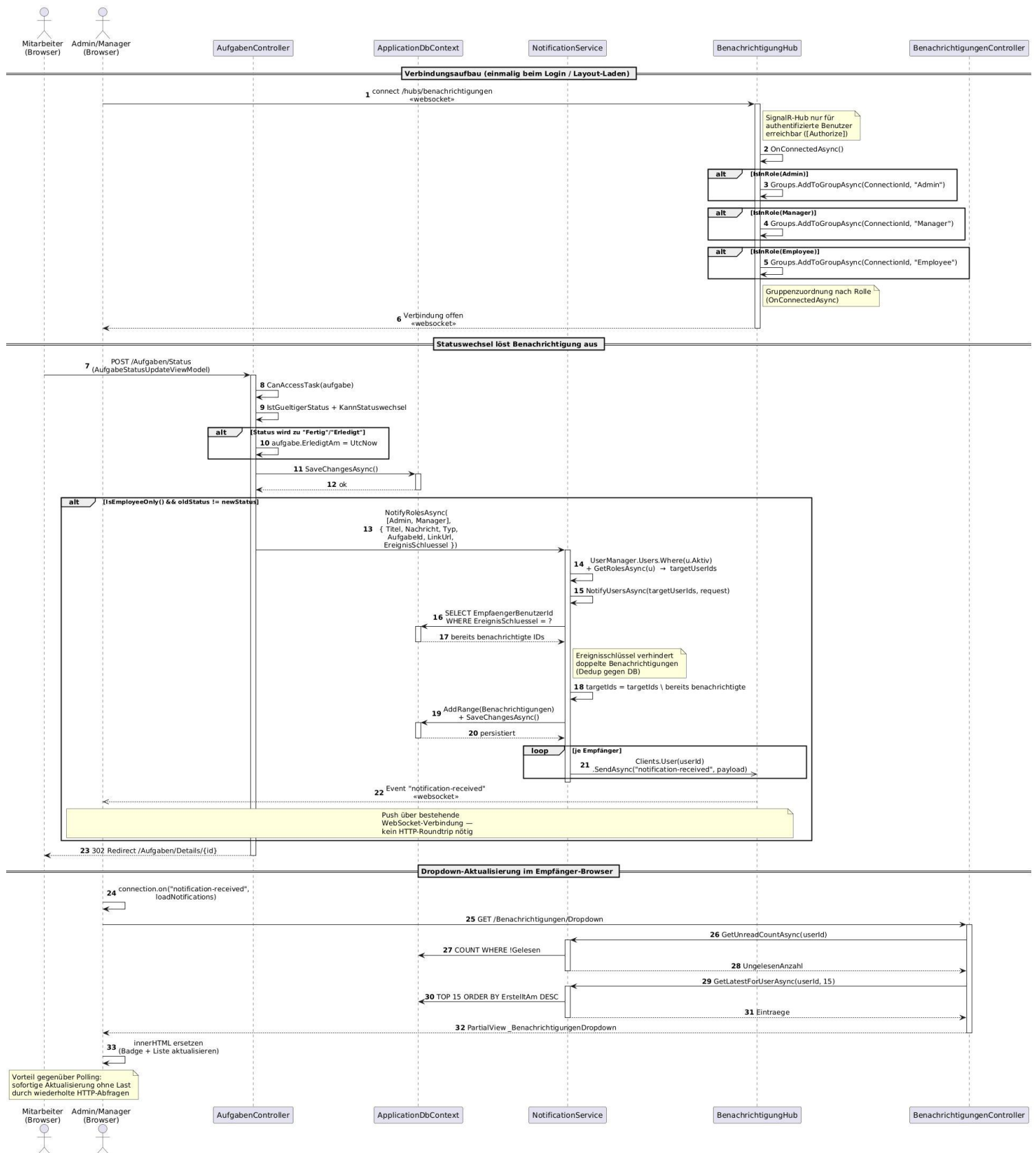


Abbildung 18: Sequenzdiagramm - SignalR-Live-Benachrichtigung

4. Umsetzungsphase

4.1 Einrichtung der Entwicklungsumgebung

Für die Umsetzung wurde zunächst die Entwicklungsumgebung eingerichtet. Als Entwicklungswerkzeug kam Microsoft Visual Studio 2022 zum Einsatz. Die Anwendung wurde als ASP.NET Core MVC-Webanwendung in C# mit Zielversion .NET 8 erstellt. Für die Datenhaltung wurde MySQL 8 eingerichtet und ergänzend MySQL Workbench zur Verwaltung und Kontrolle der Datenbank installiert.

Nach der Projekterstellung wurden die benötigten NuGet-Pakete eingebunden, insbesondere Microsoft.AspNetCore.Identity.EntityFrameworkCore, Microsoft.EntityFrameworkCore.Design, Microsoft.EntityFrameworkCore.Tools und Pomelo.EntityFrameworkCore.MySql. SignalR ist Bestandteil des ASP.NET-Core-Shared-Frameworks (Microsoft.AspNetCore.App) und muss nicht gesondert als NuGet-Paket installiert werden; es wird in Program.cs über `builder.Services.AddSignalR()` aktiviert. Die Verbindungszeichenfolge zur MySQL-Datenbank wurde in der Datei `appsettings.json` unter `DefaultConnection` hinterlegt.

4.2 Einrichtung der Datenbankanbindung

Im nächsten Schritt wurde die Datenbankanbindung umgesetzt. Hierfür wurde ein `ApplicationDbContext` erstellt, der von `IdentityDbContext` ableitet und die Verbindung zwischen Anwendung und Datenbank herstellt. Innerhalb des Kontexts wurden alle benötigten `DbSets` für die Entitäten definiert. In der Methode `OnModelCreating` wurden die Beziehungen und das Löschverhalten zwischen den Entitäten konfiguriert.

Die Registrierung des Kontexts erfolgt in `Program.cs` über die Methode `AddDbContext` mit dem Pomelo-Provider. Zusätzlich wird beim Anwendungsstart über `dbContext.Database.MigrateAsync` sichergestellt, dass ausstehende Migrationen automatisch angewendet werden. Dies vereinfacht die Inbetriebnahme im Entwicklungsumfeld erheblich.

4.3 Umsetzung der Benutzer- und Rollenverwaltung

Als erster funktionaler Baustein wurde die Benutzer- und Rollenverwaltung umgesetzt. Die zentrale Identity-Klasse `ApplicationUser` wurde mit fachlichen Feldern wie Vorname, Nachname und Aktiv erweitert. Ergänzend stellt die berechnete Eigenschaft `VollstaendigerName` eine einheitliche Namensdarstellung bereit.

Die Rollen `Admin`, `Manager` und `Employee` wurden in `Constants/RoleNames.cs` zentral definiert. Der `IdentityDataSeeder` legt beim ersten Start automatisch die Rollen und einen Standardadministrator an. Der `BenutzerverwaltungController` ist ausschließlich für die Rolle `Admin`

erreichbar und stellt eine vollständige CRUD-Funktionalität für Benutzer sowie Rollenzuweisungen bereit.

Eine zentrale Fachfunktion ist die Rollenhierarchie, die über die Methode `ExpandHierarchy` umgesetzt wurde. Der `IdentityDataSeeder` expandiert die gewählte Primärrolle automatisch in alle fachlich untergeordneten Rollen und vergibt sie in einem Schritt an den Benutzer. Ein Admin erhält dadurch technisch alle drei Rollen zugewiesen, fachlich genügt in der Oberfläche jedoch die Auswahl der höchsten Rolle.

4.4 Umsetzung der Kundenverwaltung

Die Kundenverwaltung wurde als eigener CRUD-Bereich umgesetzt. Das Model `Kunde` enthält die technische Id, den Namen sowie die Felder für die Soft-Archivierung (`IsArchived`, `ArchivedAt`). Der `KundenController` ist den Rollen Admin und Manager vorbehalten.

Anstelle eines physischen Löschsens wurde eine Soft-Archivierung implementiert. Beim Archivierungsvorgang wird der Datensatz nicht gelöscht, sondern lediglich `IsArchived` auf `true` und `ArchivedAt` auf den aktuellen Zeitpunkt gesetzt. Die aktive Kundenliste zeigt nur nicht-archivierte Einträge an. Zusätzlich wurde eine separate Archivansicht ergänzt, in der archivierte Kunden aufgelistet und bei Bedarf wiederhergestellt werden können.

4.5 Umsetzung der Aufgabenverwaltung

Die Aufgabenverwaltung ist eine der Kernkomponenten der Anwendung. Das Modell `Aufgabe` enthält die zentralen Eigenschaften Titel, Beschreibung, Priorität, Status, Erstellungsdatum, Fälligkeitsdatum, Kundenzuordnung, Zuweisung sowie `ErledigtAm` und die Archivierungsfelder.

Der `AufgabenController` implementiert die vollständige CRUD-Funktionalität. Besonderheiten sind die strenge Objektberechtigung über die Hilfsmethode `CanAccessTask`, die Mehrfachzuweisung über eine eigene Join-Tabelle `AufgabeZuweisungen` und die statusabhängigen Übergänge. Die Statuswechsel-Logik setzt beim Übergang in einen erledigten Status automatisch das Feld `ErledigtAm`. Beim Rückwechsel aus einem erledigten Status wird das Feld zurückgesetzt, damit die automatische Archivierungslogik fachlich korrekt bleibt.

Zusätzlich wurde eine Filter- und Suchfunktion ergänzt. Aufgaben können nach Status, Priorität, Kunde, Zuweisung oder Überfälligkeit gefiltert werden. Eine freie Textsuche ermöglicht das schnelle Auffinden nach Titel oder Beschreibung.

4.6 Umsetzung der Tagesberichte

Die Tagesberichte stellen fachlich den komplexesten Teil der Anwendung dar. Ein Tagesbericht gehört genau einem Benutzer und einem Kalendertag. Er enthält Felder wie Datum, Startzeit, Endzeit, PauseMinuten, SollArbeitszeitMinuten, Tagesstatus, Titel und Inhalt. Innerhalb eines Tagesberichts können mehrere TagesberichtEinträge erfasst werden, die jeweils eine einzelne Arbeitsperiode auf einer Aufgabe oder einem Kunden repräsentieren.

4.6.1 Kalenderlogik

Der Einstieg in die Tagesberichte erfolgt über einen Monatskalender. Die Kalenderlogik arbeitet mit drei Farbstufen:

- Rot: Vergangener Tag ohne gespeicherten Bericht - signalisiert einen fehlenden Nachweis
- Gelb: Gespeicherter Bericht mit Warnung, zum Beispiel mit automatischen Buchungen
- Grün: Gespeicherter Bericht ohne Warnungen - vollständig und konsistent

Heutige und zukünftige Tage werden neutral dargestellt, sind aber bereits anklickbar, um eine Vorausplanung zu ermöglichen. Der Klick auf einen Kalendertag öffnet den entsprechenden Bericht; existiert noch keiner, wird er beim Öffnen automatisch erzeugt.

4.6.2 Automatische Status- und Buchungslogik

Eine fachliche Besonderheit ist die automatische Status- und Buchungslogik. Wenn ein Tag den Sonderstatus Urlaub, Krank, Feiertag oder Berufsschule erhält, werden passende automatische Einträge über die Methode `UpsertAutomaticStatusEntryAsync` erzeugt oder aktualisiert.

Eine weitere Sonderfunktion ist die Auto-Booking-Logik. Wenn die manuell gebuchten Aufgabenzeiten geringer sind als die tatsächliche Arbeitszeit aus Startzeit, Endzeit und Pause, wird die fehlende Zeit automatisch als separater Eintrag ergänzt. Diese automatisch ergänzten Einträge werden nicht als echte gebuchte Zeit gewertet, sondern gesondert markiert und führen zu einer gelben Kalenderfarbe.

4.7 Umsetzung der Zeiterfassung

Die Zeiterfassung bildet den operativen Kern der Anwendung. Der `ZeiterfassungenController` ermöglicht das Starten, Pausieren, Fortsetzen und Stoppen von Zeiterfassungen. Jede Zeiterfassung ist an einen Benutzer, eine Aufgabe und einen Tagesbericht gebunden.

4.7.1 Statuszustände

Eine Zeiterfassung durchläuft die Statuszustände aktiv, pausiert und unterbrochen. Die Validierung stellt sicher, dass pro Benutzer immer nur eine aktive Zeiterfassung existieren kann. Beim Pausieren werden die Pausenzeiten aufsummiert, beim Fortsetzen läuft die Erfassung weiter.

4.7.2 Unterbrechungslogik

Eine Besonderheit ist die Unterbrechungslogik über die Methode `MarkInterrupted`. Laufende Zeiterfassungen können als unterbrochen markiert werden, um ungeplante Unterbrechungen sauber abzubilden. Solche Erfassungen können später fortgesetzt werden, beispielsweise nach einem Browser-Absturz oder einem Verbindungsverlust.

4.7.3 Synchronisation mit Tagesbericht

Beendete Zeiterfassungen werden automatisch als Einträge in den Tagesbericht des jeweiligen Tages übernommen. Dadurch ist die Tagesübersicht nicht von der operativen Zeiterfassung getrennt, sondern fachlich verbunden. Beim Stoppen einer Zeiterfassung wird ein entsprechender TagesberichtEintrag erzeugt oder ein vorhandener Eintrag aktualisiert.

4.7.4 Overlay und Präzisionsverbesserung

Für die sichtbare Zeiterfassung wurde ein persistentes Overlay-Widget in das Layout eingebunden. Dieses zeigt die aktuell laufende Zeiterfassung dauerhaft in der oberen Leiste an und bietet Schnellzugriffe zum Pausieren und Stoppen. In der letzten Entwicklungsphase wurde zudem eine sekundengenaue Pausenakkumulation eingeführt, um Rundungsfehler bei kurzen Pausen zu vermeiden. Hierfür wurde zusätzlich zur Minutenbasis eine Sekundenspalte `PauseGesamtSekunden` eingeführt, in welche über eine EF-Core-Migration und ein initiales `Populate-Statement` die bestehenden Werte übernommen wurden.

4.8 Umsetzung der Kommentarfunktion

Kommentare sind aufgabenbezogen und werden über den `KommentareController` verwaltet. Beim Erstellen eines Kommentars wird die zugehörige Aufgabe auf ihre Zugänglichkeit, Archivstatus und Berechtigung geprüft. Der Kommentar wird getrimmt gespeichert, um überflüssige Leerzeichen zu vermeiden.

Ein Mitarbeiter kann beim Erstellen eines Kommentars gezielt eine Benachrichtigung an die Führungsebene auslösen. Diese Option ist nur für Mitarbeiter verfügbar und führt dazu, dass Admins und Manager eine persistente Benachrichtigung erhalten, welche zusätzlich über `SignalR` live an die angemeldeten Empfänger ausgeliefert wird.

Kommentare und Zeitbuchungen werden in der Aufgaben-Detailansicht als gemeinsamer Aufgabenverlauf in chronologischer Reihenfolge dargestellt. Dadurch entsteht eine nachvollziehbare Historie der Aufgabenbearbeitung.

4.9 Umsetzung der Archivierung

Die Soft-Archivierung wurde sowohl für Kunden als auch für Aufgaben umgesetzt. Neben den Feldern `IsArchived` und `ArchivedAt` im Datenmodell sind folgende Komponenten relevant:

- Eigene Archivansichten für Kunden und Aufgaben mit Wiederherstellungsfunktion
- Schreibschutz auf archivierte Aufgaben in allen relevanten Controllern: Kommentare, Zeiterfassung und Statusänderungen sind blockiert
- Visueller Hinweis in der Aufgaben-Detailansicht, wenn eine Aufgabe archiviert ist
- Rollenabhängige Sichtbarkeit: Admins und Manager sehen alle archivierten Aufgaben, Mitarbeiter nur die ihnen zugewiesenen

4.9.1 Doppelte Absicherung

Besonders wichtig war die doppelte Absicherung der Read-only-Regeln. Sichtbare UI-Sperren allein reichen fachlich und sicherheitstechnisch nicht aus, da Formularaufrufe theoretisch direkt an den Server gesendet werden könnten. Deshalb wurden alle schreibenden Aktionen zusätzlich serverseitig gegen archivierte Aufgaben abgesichert. Dadurch ist gewährleistet, dass archivierte Datensätze unabhängig von der Oberfläche tatsächlich nicht mehr verändert werden können.

4.10 Umsetzung des Benachrichtigungssystems

Das Benachrichtigungssystem wurde mit einer Kombination aus persistenter Speicherung und Echtzeit-Push über SignalR realisiert.

4.10.1 NotificationService

Der zentrale NotificationService bietet Methoden zum Erstellen von Benachrichtigungen an einzelne Benutzer oder ganze Rollen. Beim Erstellen wird zusätzlich zum Datenbankeintrag ein SignalR-Event an die entsprechenden Empfänger gesendet. Über Ereignisschlüssel wird sichergestellt, dass identische Benachrichtigungen nicht mehrfach erzeugt werden.

4.10.2 SignalR-Hub

Der BenachrichtigungHub ist nur für authentifizierte Benutzer erreichbar. Beim Verbindungsaufbau wird die Rolle des Benutzers ausgewertet, und die Verbindung wird den passenden Gruppen (Admin, Manager, Employee) zugeordnet. Dadurch können Benachrichtigungen gezielt an einzelne Benutzer oder rollenbasiert an Gruppen gesendet werden.

4.10.3 Dropdown im Layout

Im globalen Layout wurde ein Benachrichtigungs-Dropdown mit Glockensymbol eingebunden. Es zeigt die Anzahl ungelesener Einträge als Badge an. Der JavaScript-Client verbindet sich beim Seitenaufbau mit dem SignalR-Hub und aktualisiert das Dropdown bei eingehenden Events über den Endpunkt /Benachrichtigungen/Dropdown. Eine Korrektur wurde dahingehend vorgenommen, dass sich das Dropdown nur noch per Klick und nicht mehr versehentlich per Hover öffnet.

4.11 Demo- und Vorführdaten

Für Präsentations- und Testzwecke wurde ein PresentationDataSeeder entwickelt, der die Datenbank mit realistischen Vorführdaten befüllt. Dies umfasst glaubwürdige Benutzer, Kunden, Aufgaben mit verschiedenen Status und Fälligkeiten, Kommentare, Zeiterfassungen und Tagesberichte mit unterschiedlichen Kalenderfarben. Der Seeder wurde mehrfach überarbeitet, um unkontrollierte Automatismen zu vermeiden, welche die Datenbasis für Demos unplausibel machen könnten.

5. Testphase

5.1 Testkonzept

Das Testkonzept verfolgt einen mehrstufigen Ansatz, der manuelle Fachtests, Integrationsüberlegungen und Smoke-Tests miteinander kombiniert. Ziel war es, die wichtigsten fachlichen Abläufe, Rollen- und Rechteprüfungen, Archivierung, Zeiterfassung, Tagesberichte und Kalenderlogik systematisch zu prüfen.

Für das Testprotokoll wurden 31 priorisierte Testfälle erarbeitet, welche die Kernfunktionen der Anwendung abdecken. Die Priorisierung erfolgte in Muss-Testfällen (kritische Kernfunktionen) und optionalen Testfällen (sinnvoll zusätzlich).

5.2 Testumgebung

Komponente	Wert
Betriebssystem	Windows 11
.NET-Version	8.0
Datenbank	MySQL (Pomelo EF Core)
Framework	ASP.NET Core MVC
Testarten	manueller Fachtest, Integration, UI-E2E, Smoke
Getestete Rollen	Admin, Manager, Employee
Testdaten	PresentationDataSeeder mit realistischen Vorführdaten

5.3 Testmatrix (Auszug der wichtigsten Testfälle)

Die folgende Tabelle zeigt einen Auszug der wichtigsten Muss-Testfälle aus der vollständigen Testmatrix:

ID	Bereich	Testziel	Priorität
TM-01	Authentifizierung	Login mit gültigem aktivem Benutzer	Muss
TM-02	Authentifizierung	Login mit inaktivem Benutzer wird abgelehnt	Muss
TM-03	Authentifizierung	Logout beendet Session sauber	Muss

ID	Bereich	Testziel	Priorität
TM-04	Rollen/Rechte	Unauthentifizierter Zugriff auf geschützte Actions	Muss
TM-05	Rollen/Rechte	Employee darf Admin-Bereich nicht nutzen	Muss
TM-06	Rollen/Rechte	Manager darf Kundenverwaltung, Employee nicht	Muss
TM-09	Aufgabenverwaltung	Aufgabe wird korrekt erstellt und zugewiesen	Muss
TM-10	Rollen/Rechte	Employee sieht nur eigene Aufgaben	Muss
TM-12	Aufgabenstatus	Employee darf nicht direkt auf Erledigt setzen	Muss
TM-13	Aufgabenstatus	Manager/Admin darf auf Erledigt setzen	Muss
TM-14	Aufgabenarchiv	Archivieren und Wiederherstellen funktionieren	Muss
TM-15	Auto-Archiv	Auto-Archivierung nach 7 Tagen bei Erledigt	Muss
TM-18	Zeiterfassung	Start, Pause, Fortsetzen, Stop funktionieren korrekt	Muss
TM-19	Zeiterfassung	Unterbrochene Zeiterfassung kann fortgesetzt werden	Muss
TM-21	Tagesberichte	Tagesbericht speichern setzt Save-Marker	Muss
TM-22	Tagesberichte	Auto-Booking bei fehlender Buchungszeit	Muss
TM-25	Tagesberichte	Aufgabe muss zum gewählten Kunden passen	Muss
TM-26	Kalenderlogik	Rot/Gelb/Grün korrekt angezeigt	Muss
TM-27	Benachrichtigungen	Dropdown öffnet nur per Klick, nicht per Hover	Muss
TM-28	Benachrichtigungen	Nur eigene Benachrichtigungen bearbeitbar	Muss

ID	Bereich	Testziel	Priorität
TM-31	Build/Smoke	Anwendung ist buildfähig und startet korrekt	Muss

5.4 Testdurchführung (beispielhaftes Testprotokoll)

Im Folgenden ein beispielhaftes, vollständig ausgefülltes Testprotokoll für TM-01 (Login mit gültigem aktivem Benutzer):

Feld	Inhalt
Testfall-ID	TM-01
Testbereich	Authentifizierung
Priorität	Muss
Testart	manuell
Testziel	Prüfung, ob sich ein aktiver Employee-Benutzer mit gültigen Zugangsdaten anmelden kann
Voraussetzungen	Benutzer aktiv, Passwort gültig, Vorführdaten vorhanden
Testdaten	emp1@zeitmanagement.local
Schritt 1	Login-Seite öffnen
Schritt 2	E-Mail eingeben
Schritt 3	Passwort eingeben
Schritt 4	Anmelden auslösen
Erwartetes Ergebnis	Login erfolgreich, Weiterleitung zur Aufgabenübersicht

Feld	Inhalt
Tatsächliches Ergebnis	Login erfolgreich, Weiterleitung zur Aufgabenübersicht erfolgt
Status	Bestanden
Getestet am	20.04.2026
Getestet von	Rolands Burmberger

5.5 Testergebnisse - Zusammenfassung

Die manuell durchgeführten Kern-Testfälle wurden erfolgreich absolviert. Die kompakte Ergebnis-Tabelle zeigt die wichtigsten durchgeführten Tests:

Testfall	Bereich	Status
TM-01	Login Employee	Bestanden
TM-03	Logout	Bestanden
TM-06	Rolle Manager vs Employee bei Kunden	Bestanden
TM-09	Aufgabe erstellen und zuweisen	Bestanden
TM-11	Statuswechsel Offen -> In Bearbeitung	Bestanden
TM-13	Statuswechsel auf Erledigt	Bestanden
TM-14	Archivierung und Wiederherstellung	Bestanden
TM-18	Pause / Fortsetzen / Stop	Bestanden
TM-19	Unterbrochene Zeiterfassung fortsetzen	Bestanden
TM-22	Auto-Booking mit Warnhinweis	Bestanden
TM-26	Kalenderfarben rot / gelb / grün	Bestanden
TM-27	Benachrichtigungs-Dropdown per Klick	Bestanden
TM-30	Vorführdaten plausibel	Bestanden
TM-31	Build und Smoke-Test	Bestanden

5.6 Testzyklus-Abschluss

Kennzahl	Wert
Anzahl getesteter Fälle	14 Kern-Fachtests + 7 explorative Befunde
Bestanden	11 / 14 (nach Behebung 14)
Nicht bestanden	3 (alle im Nachtest behoben)
Blockiert	0
Offene kritische Fehler	0
Freigabeempfehlung	Ja

Fachliches Gesamtfazit: Von den 14 priorisierten Kernfällen bestanden 11 im ersten Durchlauf; drei Fälle (TM-14 Archivierung, Kommentare auf archivierten Aufgaben, Rollenabsicherung Benutzerverwaltung) deckten fachliche Defekte auf, die in Abschnitt 5.7 beschrieben und in der Implementierungsphase behoben wurden. Ergänzend traten in explorativen Tests weitere sieben Befunde auf, die ebenfalls dokumentiert und behoben wurden. Im Anschluss an den strukturierten Testzyklus fand eine zusätzliche Prüfung auf einer dedizierten Stage-Umgebung statt; die dabei aufgedeckten nachgelagerten QA-Findings sind in Abschnitt 5.8 zusammengefasst. Nach erfolgreichem Nachtest aller betroffenen Fälle ist die Anwendung in den geprüften Hauptprozessen konsistent und stabil; offene kritische Fehler bestehen nicht.

5.7 Gefundene Defekte und Behebung

Die folgende Aufstellung fasst die im Testzyklus und in der Nachanalyse aufgedeckten fachlichen Defekte sowie deren Behebung zusammen. Die Kategorisierung orientiert sich an der potenziellen fachlichen Auswirkung, nicht an der technischen Komplexität.

ID	Schwere	Beschreibung	Behebung
DEF-01	Hoch	Admin-Benutzer konnten ohne Schutz gelöscht werden (bei Löschung des letzten Admins kein Zugang zur Benutzerverwaltung mehr).	Zusätzliche Prüfung in BenutzerverwaltungController.DeleteConfirmed: Löschen eines Admin-Benutzers wird abgewiesen, wenn dadurch kein Admin mehr im System verbliebe. Fehlermeldung wird in der Delete-View über ValidationSummary angezeigt.
DEF-02	Hoch	Beim Stoppen einer Zeiterfassung auf einer Aufgabe ohne Kundenzuordnung wurde die Zeit still auf den ersten Kunden in der Datenbank gebucht.	Fallback in ZeiterfassungenController.ResolveCustomerIdAsync entfernt. Ohne gültigen Kunden wird kein Tagesbericht-Eintrag erzeugt; bestehender Guard fängt den Fall ab.
DEF-03	Hoch	Archivierte Aufgaben erschienen in der Kunden-/Aufgaben-Dropdownauswahl beim Anlegen oder Bearbeiten von Tagesbericht-Einträgen.	In TagesberichteController.BuildCustomerTaskGroupsAsync Filter !c.IsArchived und !a.IsArchived ergänzt. Archivierte Objekte sind in der Auswahl nicht mehr sichtbar.
DEF-04	Mittel	Die Konstante [AUTO-TIME] war in ZeiterfassungenController und TagesberichteController unabhängig definiert. Änderung an einer Stelle hätte die Synchronisationslogik entkoppelt.	Neue Konstantenklasse Constants/TagesberichtConstants.cs eingeführt; beide Controller referenzieren AutoTimeEntryPrefix zentral.
DEF-05	Mittel	Tagesbericht konnte nicht als 'gespeichert' markiert werden, wenn keine Kunden in der Datenbank	Existenzprüfung vor MarkReportAsSavedAsync ergänzt. Fehlt ein Kunde, wird

ID	Schwere	Beschreibung	Behebung
		existieren - die interne Hilfslogik MarkReportAsSavedAsync scheiterte still.	eine verständliche Meldung zurückgegeben statt stillschweigend zu überspringen.
DEF-06	Mittel	Parallele aktive Zeiterfassungen pro Benutzer waren über den Bearbeiten- und Reaktivieren-Pfad möglich; die 'nur eine aktive'-Regel wurde nicht auf allen Pfaden geprüft.	Zentrale Prüfmethode in ZeiterfassungenController ergänzt, die andere aktive Zeiterfassungen desselben Benutzers erkennt, und in Bearbeiten- und Reaktivieren-Pfaden verankert.
DEF-07	Mittel	Beim Stoppen oder Bearbeiten einer Zeiterfassung wurden Tagesberichts-Kernzeiten (Startzeit, Endzeit, Pausenzeit) ungewollt überschrieben.	SyncTrackedTimeEntriesAsync so angepasst, dass nur passende Einträge erzeugt oder aktualisiert werden; Kernzeiten des Tagesberichts bleiben unberührt, sofern nicht gezielt bearbeitet.
DEF-08	Mittel	Archivierte Aufgaben waren über einzelne Endpunkte (Kommentieren, Zeiterfassung starten) noch beschreibbar, obwohl die UI sie als schreibgeschützt markierte.	Zusätzliche serverseitige Prüfung auf IsArchived in KommentareController und ZeiterfassungenController; Aktionen auf archivierten Aufgaben werden einheitlich mit 403/Meldung abgewiesen.
DEF-09	Mittel	Kunden konnten archiviert/gelöscht werden, obwohl noch Aufgaben auf sie referenzierten. Das führte zu inkonsistenten Zuständen in Tagesbericht-Ansichten.	In KundenController.DeleteConfirmed Prüfung auf verbundene Aufgaben ergänzt. Bestehen aktive Aufgaben, wird die Archivierung mit einer verständlichen Meldung verhindert.
DEF-10	Niedrig	Tagesbericht-Kalenderfarben waren zeitweise nicht eindeutig: automatisch ergänzte Buchungen wurden wie manuell gespeicherte Berichte behandelt.	Markerlogik vereinheitlicht: [SYSTEM-MAILED] für manuell gespeicherte Berichte, [AUTO-BOOKING] für automatisch ergänzte Zeiten. Kalenderlogik (rot/gelb/grün) basiert jetzt nachvollziehbar auf diesen Markern.

Alle in der Tabelle genannten Defekte wurden in die laufende Implementierung zurückgespielt, unmittelbar behoben und im Nachtest gegen den ursprünglichen Testfall beziehungsweise gegen einen neu ergänzten expliziten Nachtest geprüft. Die Nachtests sind im Testprotokoll separat mit dem Status „bestanden nach Fix“ gekennzeichnet.

5.8 Nachgelagerte QA-Findings aus dem Stage-Test

Im Anschluss an den strukturierten Testzyklus wurde die Anwendung auf einer dedizierten Stage-Umgebung erneut geprüft. Dabei traten weitere fachliche Findings sowie Verbesserungspotenziale zu Bedienführung, Validierung und Konfigurierbarkeit zutage. Die folgende Aufstellung dokumentiert die identifizierten Punkte und ihre konkrete Umsetzung. Alle Findings wurden noch vor der Abnahme des Prototyps behoben und im Anschluss erneut gegen den auslösenden Prüffall verifiziert.

ID	Modul	Beschreibung	Behebung / Umsetzung
QA-01	Auth/Benutzer	Header zeigte nur die E-Mail-Adresse, der Abmelden-Button war optisch nicht eindeutig rechtsbündig platziert.	_Layout.cshtml überarbeitet; ApplicationUserClaimsPrincipalFactory ergänzt, sodass Vor- und Nachname als Claims verfügbar sind und im Header zusammen mit der E-Mail angezeigt werden.
QA-02	Auth/Benutzer	„Benutzer löschen“ erzeugte Fremdschlüssel-Fehler, sobald der Benutzer Zeiterfassungen oder Tagesberichte besaß; zudem konnte sich ein Admin selbst entfernen.	Delete-View und DeleteConfirmed-Action aus BenutzerverwaltungController entfernt; ausschließlich das vorhandene Deaktivieren über IsAktiv ist nutzbar.
QA-03	Auth/Benutzer	Bezeichnung und Validierungsmeldung des Feldes „PasswortBestaetigung“ im Registrierungs- und Passwort-ändern-Flow waren uneinheitlich.	Property-Bezeichnung und Validierungsattribute angeglichen; Fehlermeldung verweist nun konsistent auf das Bestätigungsfeld.
QA-04	Aufgaben	Statusbezeichnung „Fertig“ war fachlich missverständlich, da der Status den Übergang zur Testphase markiert.	UI-Beschriftung in allen Views, Filtern, Dropdowns und Notification-Texten in „Bereit zum Testen“ umbenannt; interne Enum-Werte unverändert (keine Migration).
QA-05	Aufgaben	Kundenfilter auf der Aufgaben-Index-Seite enthielt auch Kunden ohne aktive Aufgaben und führte zu leeren Filterergebnissen.	Dropdown so angepasst, dass nur Kunden mit mindestens einer aktiven (nicht-archivierten) Aufgabe angezeigt werden.

ID	Modul	Beschreibung	Behebung / Umsetzung
QA-06	Aufgaben	Archivierung erledigter Aufgaben gab nur eine generische Meldung aus; bei Blockade war für den Anwender nicht erkennbar, welche Aufgaben das Problem verursachten.	TempData-Erfolgsmeldung nach erfolgreicher Archivierung ergänzt; bei blockierter Archivierung wird die konkrete Liste der betroffenen Aufgaben in der View eingeblendet.
QA-07	Aufgaben	Auf eine bereits erledigte Aufgabe konnte über das UI eine neue Zeiterfassung gestartet werden; laufende Erfassungen wurden beim Setzen auf „Erledigt“ nicht zuverlässig abgeschlossen.	Start einer neuen Zeiterfassung im Controller validiert und im UI gesperrt; bestehende laufende Erfassungen werden beim Statuswechsel auf „Erledigt“ sauber beendet.
QA-08	Aufgaben	Manager ohne Aufgabenbezug konnten für beliebige Aufgaben Zeit erfassen; eine fachliche Zuordnung von Mit-Verantwortlichen fehlte.	Neues Feld Mit-Ersteller auf Aufgaben mit Migration AddAufgabeMitErsteller; Zeiterfassung ist nur noch durch Ersteller oder Mit-Ersteller möglich.
QA-09	Aufgaben	Die Frist für die Auto-Archivierung erledigter Aufgaben war hartcodiert (14 Tage) und ohne Eingriff in den Quellcode nicht änderbar.	Frist über Schlüssel Aufgaben.ErledigtArchivierungsTage in SystemEinstellungen konfigurierbar; Caching über IMemoryCache (TTL 5 Minuten); Fallback auf 7 Tage mit Logging-Warnung bei fehlendem oder ungültigem Eintrag.
QA-10	Tagesbericht	Im Tagesbericht-Formular war das Aufgaben-Dropdown nicht an die Kundenauswahl gekoppelt, wodurch eine Aufgabe versehentlich dem falschen Kunden zugeordnet werden konnte.	Aufgaben-Dropdown wird dynamisch nach gewähltem Kunden gefiltert.
QA-11	Tagesbericht	Bei Zeitraum-Überschneidungen erschienen mehrere redundante Validierungsmeldungen; Einträge außerhalb der erfassten Arbeitszeit wurden nicht abgefangen.	Validierung „von – bis“ konsolidiert; zusätzliche Prüfung gegen die für den Tag erfasste Arbeitszeit ergänzt.
QA-12	Tagesbericht	Bisheriger Auto-Fill-Mechanismus erzeugte bei unvollständig erfasster Zeit eine künstliche Nachbuchung und	Auto-Fill entfernt; der Tagesbericht wird mit den exakt eingegebenen Werten gespeichert.

ID	Modul	Beschreibung	Behebung / Umsetzung
		führte zu nicht nachvollziehbaren Tageseinträgen.	
QA-13	Tagesbericht	Manuell im Tagesbericht eingetragene Zeiten erschienen nicht in der Zeiterfassungs-Historie der zugehörigen Aufgabe.	AufgabeDetailsViewModel um manuelle Tagesbericht-Einträge erweitert; diese werden in der Aufgabenhistorie korrekt angezeigt.
QA-14	Berichtsliste	Manager und Admins hatten keine Möglichkeit, die Tagesberichte-Index-Seite nach einem einzelnen Mitarbeiter zu filtern.	Dropdown-Filter „Mitarbeiter“ in der Index-Seite ergänzt; vereinfacht die Auswertung einzelner Personen ohne separate Datenbankabfrage.
QA-15	Notifications	Statusänderungen an Aufgaben lösten Benachrichtigungen an alle Manager aus, was bei größeren Teams zu Notification-Spam führte.	Empfängerkreis auf Aufgaben-Ersteller und Admins eingeschränkt.
QA-16	Kunden	Kunden konnten archiviert werden, obwohl noch aktive Aufgaben referenziert waren; der interne Systemkunde war nicht sauber geschützt.	Archivierung blockiert, solange aktive Aufgaben am Kunden hängen; betroffene Aufgaben werden bei Blockade direkt eingeblendet; der Systemkunde ist generell schreibgeschützt.
QA-17	Stage-Test	Stabilität von „Remember Me“ über Browserneustart und korrekte Rückkehr-Navigation aus der Aufgabendetailansicht in die Archivliste waren noch nicht produktionsnah verifiziert.	Auf dem Stage-Server verifiziert: Session bleibt gemäß Cookie-Lifetime erhalten; Aufgabendetails → Zurück führt zuverlässig zur Archivliste statt zur aktiven Liste.

Die Findings aus diesem Stage-Test bestätigten den Wert eines zusätzlichen, möglichst produktionsnahen Prüflaufs jenseits des regulären Testprotokolls: Insbesondere konfigurationsbezogene Aspekte (z. B. die Ausgliederung der Auto-Archivierungsfrist in die SystemEinstellungen) sowie Empfängerlogiken bei Benachrichtigungen treten erst unter realistischen Mehrbenutzer-Bedingungen klar zutage. Sämtliche Punkte wurden im laufenden Implementierungszyklus zurückgespielt und sind im aktuellen Prototyp wirksam.

6. Einführung und Übergabe

6.1 Deployment

Das Deployment des prototypischen Systems erfolgte in einer unternehmensinternen Testumgebung. Die Anwendung wurde hierzu lokal über den integrierten Webserver Kestrel bereitgestellt, sodass ein browserbasierter Zugriff ohne zusätzliche Infrastruktur möglich war.

Zur Vorbereitung wurde die Anwendung über den Befehl `dotnet publish` kompiliert und in einer lauffähigen Version bereitgestellt. Die Anbindung an die MySQL-Datenbank erfolgt über die in der Konfigurationsdatei hinterlegte Verbindungszeichenfolge. Beim ersten Start werden ausstehende Migrationen automatisch angewendet sowie die initialen Rollen und ein Standardadministrator angelegt.

Die gewählte Testumgebung ermöglicht eine realitätsnahe Ausführung der Anwendung und dient als Grundlage für Demonstration, Einweisung und fachliche Bewertung.

6.2 Einweisung der Nutzer

Nach erfolgreichem Deployment wurde eine strukturierte Einweisung in die Anwendung durchgeführt. Diese erfolgte im Rahmen einer kompakten Demonstration mit den Projektbetreuer und einem Mitarbeiter im Betrieb.

Ziel der Einweisung war es, die grundlegenden Funktionen der Anwendung praxisnah zu vermitteln und die eigenständige Nutzung zu ermöglichen.

Die Demonstration erfolgte anhand eines vorbereiteten Beispielszenarios und umfasste folgende Inhalte:

Anmeldung und rollenbasierte Navigation innerhalb der Anwendung

Erstellung und Verwaltung von Aufgaben inklusive Zuweisung und Statusänderung

Nutzung der Zeiterfassung mit Start-, Pause- und Stopp-Funktion sowie Overlay-Anzeige

Erstellung und Bearbeitung von Tagesberichten einschließlich Kalenderlogik und Statusanzeige

Verwendung der Kommentarfunktion zur Kommunikation innerhalb von Aufgaben

Einsicht und Nutzung des Benachrichtigungssystems

Während der Demonstration konnten Rückfragen direkt geklärt werden, wodurch ein besseres Verständnis der Anwendung erreicht wurde.

6.3 Hinweise zur Inbetriebnahme

Für die Inbetriebnahme in einer neuen Umgebung sind folgende Schritte notwendig:

- MySQL 8 installieren und eine leere Datenbank anlegen
- Connection String in appsettings.json anpassen
- Anwendung über dotnet run beziehungsweise hinter einem IIS starten
- Beim ersten Start werden Migrationen automatisch ausgeführt und der Standardadministrator angelegt
- Anmeldung mit dem Standardadministrator und Anlegen weiterer Benutzer über die Benutzerverwaltung

7. Soll-Ist-Vergleich

7.1 Stundenaufwand

Die folgende Tabelle stellt die im Projektantrag geplanten Stunden den tatsächlich aufgewendeten gegenüber. Alle Phasen konnten im Rahmen der IHK-Vorgaben abgeschlossen werden.

Projektphase	Soll (h)	Ist (h)	Differenz
Analyse	8	10	+ 2
Entwurf	12	7	- 5
Implementierung	36	40	+ 4
Test	8	6	- 2
Einführung und Übergabe	4	4	+/- 0
Projektdokumentation	12	13	+ 1
Summe	80	80	+/- 0

Die Summe wurde im Rahmen der 80 Stunden gehalten, die Verteilung zwischen den Phasen weicht aber bewusst von der ursprünglichen Planung ab. Die Analyse brauchte mit 10 Stunden rund 2 Stunden länger als geplant, weil neben den Gesprächen mit Abteilungsleitern zusätzlich eine schriftliche Kurzumfrage sowie eine Sichtung der bestehenden Excel-Vorlagen und der Mail-Abläufe durchgeführt wurden. Dieser Mehraufwand hat sich im Entwurf ausgezahlt, der deutlich zügiger abgeschlossen werden konnte (7 statt 12 Stunden), weil das Datenmodell und das Rollenkonzept direkt aus der Ist-Aufnahme abgeleitet werden konnten und weniger Iterationen nötig waren.

In der Implementierung fielen 4 Stunden mehr an als geplant. Der Mehraufwand verteilt sich auf drei Bereiche: die Behebung der im Test gefundenen Defekte (siehe Abschnitt 5.5), die Integration von SignalR inklusive rollenbasierter Gruppenzuordnung sowie die sekundengenaue Pausen Summierung, die erst in einer späteren Iteration ergänzt wurde. Die Testphase konnte entsprechend um 2 Stunden gekürzt werden, weil ein Teil der funktionalen Prüfungen überlappend zur Entwicklung lief und das strukturierte Testprotokoll die gezielte Abarbeitung der Muss-Fälle erlaubte. Die Dokumentation fiel mit 13 Stunden knapp über Plan, insbesondere durch die Nacharbeit am Sicherheits- und Datenschutzkapitel.

7.2 Zielerreichung

Alle zentralen Projektziele aus der Analysephase konnten erreicht werden:

Ziel	Erreicht
Integration von Zeiterfassung und Aufgabenverwaltung	Ja
Reduzierung von Medienbrüchen gegenüber Excel und E-Mail	Ja
Echtzeit-Überblick für das Management	Ja (Dashboard, Filter, Benachrichtigungen)
Automatische Generierung von Tagesberichten	Ja (inkl. Auto-Booking)
Rollenbasiertes Rechtekonzept	Ja (Admin/Manager/Employee mit Objektprüfungen)
Kommentar- und Benachrichtigungssystem	Ja (inkl. SignalR)

7.3 Abweichungen gegenüber dem Projektantrag

Im Laufe des Projekts wurden gegenüber dem ursprünglichen Projektantrag einige sinnvolle Anpassungen vorgenommen, ohne den Projektrahmen zu sprengen:

- Wechsel der Zielversion von .NET 6 auf .NET 8 (LTS): Im Projektantrag war noch .NET 6 vorgesehen, dessen LTS-Support jedoch im November 2024 ausläuft. Da die Implementierung danach erfolgte, wurde auf .NET 8 (LTS bis November 2026) gewechselt; der fachliche Funktionsumfang bleibt davon unberührt.
- Erweiterung des Rollenmodells um eine Admin-Rolle zusätzlich zu Manager und Employee, um Benutzerverwaltungsaufgaben klar von fachlicher Führung abzugrenzen
- Einführung der Soft-Archivierung anstelle eines einfachen Löschrmechanismus, um Revisionssicherheit zu erhöhen
- Implementierung eines Hosted Services für die automatische Archivierung erledigter Aufgaben
- Ergänzung einer sekundengenauen Pausenakkumulation (zusätzliche Spalte PauseGesamtSekunden mit Migration), um Rundungsfehler bei kurzen Pausen zu vermeiden
- Vereinheitlichung der Marker-Logik für Tagesberichte (Einführung von [SYSTEM-SAVED], [AUTO-TIME] und [AUTO-BOOKING]) inklusive Auslagerung der AutoTimeEntryPrefix-Konstante in eine gemeinsame Konstantenklasse

- Nachschärfung der Schreibsperre auf archivierte Aufgaben: durchgängig serverseitige Prüfung in sämtlichen schreibenden Actions (Kommentare, Zeiterfassung, Statuswechsel)
- Anpassung des Aufgaben-Status-Workflows: Der ursprünglich vorgesehene Status „Pausiert“ wurde im Laufe der Umsetzung aus dem aktiven Workflow entfernt und nur noch als Legacy-Wert auf „In Bearbeitung“ normalisiert, da die Pausierung fachlich besser über die Zeiterfassung abgebildet wird. Im Gegenzug wurde der finale Status „Erledigt“ eingeführt, dessen Setzen und Zurücksetzen ausschließlich Manager und Admin vorbehalten ist (Endabnahme).
- Erweiterung der Mitarbeiterzuweisung von einer 1:1-Beziehung auf eine n:m-Beziehung über die Join-Tabelle `AufgabeZuweisungen`, um Mehrfachzuweisungen einer Aufgabe an mehrere Mitarbeiter zu ermöglichen. Das ursprüngliche Feld `ZugewiesenerBenutzerId` bleibt aus Kompatibilitätsgründen im Modell erhalten.
- Ergänzung einer Kurzzeit-Verwerfungsregel in der Zeiterfassung: Erfassungen mit einer Netto-Dauer von unter 60 Sekunden werden beim Stoppen, Bearbeiten oder Auflösen einer Unterbrechung nicht gespeichert, sondern verworfen. Damit werden versehentliche Mini-Buchungen (z. B. durch Doppelklicks) zuverlässig vermieden.
- Auslagerung der Archivierungsfrist aus dem hartcodierten Wert in die Tabelle `SystemEinstellung` (Schlüssel `Aufgaben.ErledigtArchivierungTage`, Standardwert 7 Tage). Der Wert wird zur Laufzeit gelesen und gecacht, sodass Anpassungen ohne Neudeployment möglich sind.
- Ergänzung eines Live-Benachrichtigungssystems mit SignalR und persistenter Speicherung in der Tabelle `Benachrichtigungen`. Im ursprünglichen Antrag war keine Echtzeit-Komponente vorgesehen; sie wurde aufgenommen, um Statusereignisse und Kommentare ohne Seitenneuladen sichtbar zu machen.
- Ergänzung eines Schutzes gegen das Löschen des letzten Admin-Benutzers sowie gegen die Archivierung von Kunden mit noch aktiven Aufgaben

8. Fazit und Ausblick

8.1 Fazit

Das Projekt ZeitmanagementTool konnte erfolgreich innerhalb des IHK-Rahmens von 80 Stunden abgeschlossen werden. Die entwickelte Webanwendung erfüllt die zentralen Anforderungen und bildet den definierten Kernprozess rund um Aufgabe, Zeit, Tagesbericht und teambezogene Kommunikation ab.

Die fachlich stärksten Elemente der Anwendung sind die enge Verknüpfung von Aufgabe und Zeiterfassung, die automatische Überführung in Tagesberichte, das mehrstufige Rollen- und Rechtekonzept sowie die sichtbare Kalender- und Markerlogik. Diese Bereiche bieten einen konkreten Mehrwert gegenüber der vorherigen Lösung mit Excel und E-Mail und adressieren die in der Ist-Analyse identifizierten Schwachstellen zielgerichtet.

Technisch zeigt die Anwendung eine klare Schichtung: Controller, Views, Models, ViewModels und Services sind sauber getrennt. Die Rechteprüfung erfolgt auf zwei Ebenen, was eine robuste Sicherheitsgrundlage bildet. Die Soft-Archivierung sorgt für Revisionssicherheit, und das SignalR-basierte Benachrichtigungssystem bietet einen modernen Komfort.

8.2 Herausforderungen im Projektverlauf

Im Projektverlauf traten einige Herausforderungen auf, die den Lerneffekt deutlich erhöht haben:

- Die fachliche Komplexität der Tagesbericht-Logik mit Kalenderfarben, automatischen Buchungen und Sonderstatus erforderte mehrere Iterationen
- Die Synchronisation zwischen Zeiterfassung und Tagesberichten musste an mehreren Stellen **vereinheitlicht** werden
- Kleinere Präzisionsprobleme bei der Pausen Summierung wurden durch die Einführung einer zusätzlichen Sekundenspalte behoben
- Die sauberen Schichten der Rechteprüfung (Authorize-Attribute plus Objektmethoden) mussten durchgängig etabliert werden

8.3 Projektstärken

- Sauberes Rollen- und Zugriffskonzept durch Kombination aus Attributen und Objektprüfungen
- Gute fachliche Tiefe bei Tagesbericht und Zeiterfassung durch Auto-Status, Auto-Buchung und Markerlogik
- Nachvollziehbare Datenmodellierung mit klaren Beziehungen im ApplicationDbContext
- Archivierungs- und Restore-Logik statt Datenverlust
- Live-Benachrichtigungen mit SignalR für bessere Benutzererfahrung

8.4 Ausblick und mögliche Erweiterungen

Die aktuelle Anwendung ist als Prototyp konzipiert und kann sinnvoll in mehreren Richtungen weiterentwickelt werden:

- Auslagerung der Fachlogik aus den Controllern in dedizierte Application Services zur Erhöhung der Testbarkeit
- Einführung eines einheitlichen Audit-Loggings zur vollständigen Nachvollziehbarkeit von Änderungen
- Zusätzliche Dashboard-Auswertungen pro Mitarbeiter, Kunde oder Monat
- Integration von E-Mail- oder Push-Benachrichtigungen für Ereignisse außerhalb der aktiven Sitzung
- Produktionsreifes Secret-Management über User Secrets, Environment Variables oder einen Key Vault
- Anbindung an ein externes Lohnabrechnungs- oder ERP-System über definierte Schnittstellen
- Ergänzung einer nativen mobilen Anwendung beziehungsweise einer Progressive Web App

8.5 Persönliches Resümee

Das Projekt bot die Gelegenheit, einen realistischen Business-Prozess vollständig zu durchdringen und technisch in einer modernen .NET-Umgebung umzusetzen. Besonders wertvoll war die ganzheitliche Betrachtung von Analyse über Entwurf bis Test und Übergabe. Die enge Verbindung zwischen fachlicher Anforderung und technischer Lösung hat gezeigt, wie wichtig saubere Architektur, Rechte-Management und benutzerfreundliches Design im Enterprise-Umfeld sind. Der Lerngewinn aus Themen wie SignalR, Hosted Services und den Feinheiten von Entity Framework Core Migrationen ist für zukünftige Projekte eine hervorragende Grundlage.

9. Anhang

9.1 Abkürzungsverzeichnis

Abkürzung	Bedeutung
API	Application Programming Interface
ASP.NET	Active Server Pages .NET (Web-Framework)
CRUD	Create, Read, Update, Delete
CSRF	Cross-Site Request Forgery
DB	Datenbank
EF Core	Entity Framework Core (ORM)
ERP	Enterprise Resource Planning
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IHK	Industrie- und Handelskammer
IIS	Internet Information Services
MVC	Model-View-Controller
ORM	Object-Relational Mapper
SPA	Single Page Application
SQL	Structured Query Language
UI	User Interface
UML	Unified Modeling Language

9.2 Abbildungsverzeichnis

- Abbildung 1: Use-Case-Diagramm Benutzer- und Rollenverwaltung
- Abbildung 2: Use-Case-Diagramm Aufgabenverwaltung (Entwurfsstand)
- Abbildung 3: Use-Case-Diagramm Aufgabenansicht
- Abbildung 4: Use-Case-Diagramm Statusworkflow (Entwurfsstand)
- Abbildung 5: Zustandsdiagramm Aufgaben-Status-Workflow (Entwurfsstand)
- Abbildung 6: Use-Case-Diagramm Zeiterfassung (Entwurfsstand)
- Abbildung 7: Use-Case-Diagramm Tagesbericht (Entwurfsstand)
- Abbildung 8: Use-Case-Diagramm Kommentare und Benachrichtigungen
- Abbildung 9: Datenbankdiagramm der Kerntabellen (Entwurfsstand)
- Abbildung 10: Klassendiagramm der Domänenmodelle
- Abbildung 11: Mockup Aufgabenübersicht
- Abbildung 12: Mockup Aufgaben-Detailansicht
- Abbildung 13: Mockup Tagesbericht-Maske
- Abbildung 14: Mockup Tagesbericht-Übersicht
- Abbildung 15: Aktivitätsdiagramm Normaler Ablauf Zeiterfassung
- Abbildung 16: Aktivitätsdiagramm Behandlung unterbrochener Zeiterfassungen
- Abbildung 17: Sequenzdiagramm - Zeiterfassung Start / Pause / Stop
- Abbildung 18: Sequenzdiagramm - SignalR-Live-Benachrichtigung

9.3 Tabellenverzeichnis

- Tabelle 1: Stundenverteilung der Projektphasen
- Tabelle 2: Verwendeter Technologie-Stack
- Tabelle 3: Rollenübersicht
- Tabelle 4: Testumgebung
- Tabelle 5: Auszug der wichtigsten Muss-Testfälle
- Tabelle 6: Beispielhaftes Testprotokoll TM-01
- Tabelle 7: Testergebnisse Zusammenfassung
- Tabelle 8: Testzyklus-Abschluss
- Tabelle 9: Soll-Ist-Vergleich Stundenaufwand
- Tabelle 10: Zielerreichung
- Tabelle 11: Wirtschaftlichkeits-Gegenüberstellung Make-or-Buy
- Tabelle 12: Gefundene Defekte und Behebung
- Tabelle 13: Nachgelagerte QA-Findings aus dem Stage-Test

9.4 Literatur- und Quellenverzeichnis

- Microsoft Corporation: ASP.NET Core Dokumentation - <https://learn.microsoft.com/aspnet/core>
- Microsoft Corporation: Entity Framework Core Dokumentation - <https://learn.microsoft.com/ef/core>
- Microsoft Corporation: ASP.NET Core SignalR Dokumentation - <https://learn.microsoft.com/aspnet/core/signalr>
- Pomelo Foundation: Pomelo.EntityFrameworkCore.MySql - GitHub Repository
- Oracle Corporation: MySQL 8.0 Reference Manual - <https://dev.mysql.com/doc/refman/8.0/en/>
- Bootstrap Team: Bootstrap 5 Dokumentation - <https://getbootstrap.com/docs/5.0/>

9.5 Eidesstattliche Erklärung

Hiermit erkläre ich, Rolands Burmberger, an Eides statt, dass ich die vorliegende Projektdokumentation selbstständig und ohne fremde Hilfe verfasst habe. Andere als die angegebenen Quellen und Hilfsmittel wurden nicht verwendet. Alle wörtlichen und sinngemäß übernommenen Stellen aus anderen Werken sind entsprechend gekennzeichnet.

Ort, Datum, Unterschrift

9.6 Quellcode-Auszüge

Dieses Kapitel enthält sechs ausgewählte Code-Auszüge, die fachlich besonders relevante Stellen des Projekts dokumentieren. Die Auszüge wurden gegenüber dem Original teilweise gekürzt; Einrückung und Bezeichner wurden beibehalten. Die vollständigen Quellen liegen dem Projekt im Repository bei.

9.6.1 NotificationService – Deduplizierung über EreignisSchlüssel

Pfad: ZeitmanagementTool/Services/NotificationService.cs (Z. 27–67)

Die Methode `NotifyUsersAsync` verteilt Benachrichtigungen an mehrere Empfänger und persistiert sie sowohl in der Datenbank als auch via SignalR-Push. Fachlich entscheidend ist die Deduplizierung über den EreignisSchlüssel: Bevor neue Einträge angelegt werden, wird gegen bereits bestehende Benachrichtigungen mit gleichem Schlüssel gefiltert. Dadurch wird verhindert, dass derselbe Empfänger mehrfach zum gleichen Ereignis (z.B. Aufgabe überfällig) benachrichtigt wird.

```
public async Task NotifyUsersAsync(IEnumerable<string> userIds, NotificationCreateRequest request, CancellationToken cancellationToken = default)
```

```

{
    var targetIds = userIds
        .Where(id => !string.IsNullOrEmpty(id))
        .Distinct(StringComparer.Ordinal)
        .ToList();

    if (targetIds.Count == 0)
    {
        return;
    }

    if (!string.IsNullOrEmpty(request.EreignisSchluessel))
    {
        var existingTargetIds = await _context.Benachrichtigungen
            .Where(n => n.EreignisSchluessel == request.EreignisSchluessel && targetIds.Contains(n.EmpfaengerBenutzerId))
            .Select(n => n.EmpfaengerBenutzerId)
            .Distinct()
            .ToListAsync(cancellationToken);

        targetIds = targetIds.Except(existingTargetIds, StringComparer.Ordinal).ToList();
    }

    if (targetIds.Count == 0)
    {
        return;
    }

    var notifications = targetIds.Select(userId => new Benachrichtigung
    {
        EmpfaengerBenutzerId = userId,
        Titel = request.Titel,
        Nachricht = request.Nachricht,
        Typ = request.Typ,
        EreignisSchluessel = request.EreignisSchluessel,
        ErstelltAm = DateTime.UtcNow
    }).ToList();

    _context.Benachrichtigungen.AddRange(notifications);
    await _context.SaveChangesAsync(cancellationToken);
}

```

9.6.2 BenachrichtigungHub – SignalR-Hub mit Rollen-Gruppenzuordnung

Pfad: ZeitmanagementTool/Hubs/BenachrichtigungHub.cs (komplett)

Der SignalR-Hub übernimmt die Echtzeit-Verteilung von Benachrichtigungen. Bei jeder neuen Verbindung wird der Benutzer automatisch in eine SignalR-Gruppe entsprechend seiner Rolle (Admin, Manager, Employee) eingeordnet. So können rollenbasierte Broadcasts (z.B. „neuer Tagesbericht eingereicht“ an alle Manager) effizient über `Groups.AddToGroupAsync` zugestellt werden, ohne dass der Aufrufer einzelne UserIds ermitteln muss. Das `[Authorize]`-Attribut stellt sicher, dass nur authentifizierte Clients verbinden.

`[Authorize]`

```
public class BenachrichtigungHub : Hub
{
    public override async Task OnConnectedAsync()
    {
        if (Context.User?.IsInRole(RoleNames.Admin) == true)
        {
            await Groups.AddToGroupAsync(Context.ConnectionId, RoleNames.Admin);
        }

        if (Context.User?.IsInRole(RoleNames.Manager) == true)
        {
            await Groups.AddToGroupAsync(Context.ConnectionId, RoleNames.Manager);
        }

        if (Context.User?.IsInRole(RoleNames.Employee) == true)
        {
            await Groups.AddToGroupAsync(Context.ConnectionId, RoleNames.Employee);
        }

        await base.OnConnectedAsync();
    }
}
```

9.6.3 ZeiterfassungenController – Start/Pause/Stop mit PauseGesamtSekunden-Akkumulation

Pfad: ZeitmanagementTool/Controllers/ZeiterfassungenController.cs (Auszug Z. 386–453)

Der Auszug zeigt den Kern der Pausen- und Stopp-Logik. Beim Fortsetzen einer Pause wird die seit PauseStartzeit vergangene Zeit sekundengenau auf PauseGesamtSekunden addiert; PauseGesamtMinuten wird daraus abgeleitet. Beim Stop wird eine evtl. noch laufende Pause analog akkumuliert und anschließend die Netto-Arbeitszeit als Bruttodauer minus PauseGesamtSekunden berechnet. Dieser sekundengenaue Ansatz vermeidet Rundungsfehler, die bei minutenweiser Akkumulation entstehen würden.

```
// --- Pause / Fortsetzen ---
if (entry.IstPausiert)
{
    if (entry.PauseStartzeit.HasValue)
    {
        var pauseSek = (int)Math.Max(0, (DateTime.Now - entry.PauseStartzeit.Value).TotalSeconds);
        entry.PauseGesamtSekunden += pauseSek;
        entry.PauseGesamtMinuten = entry.PauseGesamtSekunden / 60;
    }

    entry.IstPausiert = false;
    entry.PauseStartzeit = null;
    TempData["TrackingMessage"] = "Zeiterfassung fortgesetzt.";
}
else
{
    entry.IstPausiert = true;
    entry.PauseStartzeit = DateTime.Now;
    TempData["TrackingMessage"] = "Zeiterfassung pausiert.";
}

// --- Stop: laufende Pause schliessen, Netto-Dauer berechnen ---
var now = DateTime.Now;

if (entry.IstPausiert && entry.PauseStartzeit.HasValue)
{
    var pauseSek = (int)Math.Max(0, (now - entry.PauseStartzeit.Value).TotalSeconds);
    entry.PauseGesamtSekunden += pauseSek;
    entry.PauseGesamtMinuten = entry.PauseGesamtSekunden / 60;
}
```

```

var nettoSek = Math.Max(0, (int)(now - entry.Startzeit).TotalSeconds - entry.PauseGesamtSe-
kunden);
if (nettoSek < 60)
{
    _context.Zeiterfassungen.Remove(entry);
    await _context.SaveChangesAsync();
    return RedirectToLocal(returnUrl, "Details", "Aufgaben", new { id = aufgabeId });
}

entry.Endzeit = now;
entry.IstAktiv = false;

```

9.6.4 ZeiterfassungKonfliktService – Verhindern paralleler Zeiterfassungen

Pfad: ZeitmanagementTool/Services/ZeiterfassungKonfliktService.cs (Auszug Z. 16–65)

Beim manuellen Bearbeiten einer Zeiterfassung muss verhindert werden, dass sich der neue Zeitraum mit bestehenden Einträgen überlappt. `PrueferAsync` lädt alle anderen Zeiterfassungen des Benutzers am gleichen Tag und unterscheidet vier Fälle: keine Überlappung (übersprungen), neuer Zeitraum vollständig innerhalb eines bestehenden (Speichern verboten), bestehender Eintrag vollständig innerhalb des neuen (Löschung) sowie linke/rechte Kürzung (Zeitfenster wird zugeschnitten). Das Resultat wird vor dem Persistieren in einer Bestätigungsmaske angezeigt.

```

public async Task<KonfliktPreviewResult> PrueferAsync(
    string benutzerId,
    DateTime neuVon,
    DateTime neuBis,
    int excludeZeiterfassungId)
{
    var result = new KonfliktPreviewResult();

    var kandidaten = await _context.Zeiterfassungen
        .Include(z => z.Aufgabe)
        .Where(z => z.BenutzerId == benutzerId)
        .Where(z => z.ZeiterfassungId != excludeZeiterfassungId)
        .Where(z => z.Endzeit != null)
        .Where(z => z.Startzeit.Date == neuVon.Date)
        .ToListAsync();

    foreach (var z in kandidaten)
    {
        var von = z.Startzeit;
        var bis = z.Endzeit!.Value;
    }
}

```

```

var titel = z.Aufgabe?.Titel ?? "Aufgabe";

// Keine Überlappung
if (neuBis <= von || neuVon >= bis)
    continue;

// Sonderfall: neuer Zeitraum komplett innerhalb eines bestehenden -> Speichern
verboten
if (neuVon >= von && neuBis <= bis)
{
    result.IstSpeichernErlaubt = false;
    result.Fehlermeldung = $"Der neue Zeitraum liegt vollständig innerhalb von
'{titel}' - bitte die andere Zeiterfassung zuerst anpassen.";
    return result;
}

// Vollständige Löschung: bestehender Eintrag komplett innerhalb des neuen Zeit-
raums
if (von >= neuVon && bis <= neuBis)
{
    result.Loeschungen.Add(new KonfliktEintrag { ZeiterfassungId = z.Zeiterfassung-
gId, AufgabeTitel = titel });
    continue;
}

// Linke / rechte Kürzung: bestehender Eintrag wird zugeschnitten
// ... (Berechnung von NeuVon/NeuBis, Pause-Anpassung)
}

result.HatKonflikt = result.Kuerzungen.Count > 0 || result.Loeschungen.Count > 0;
return result;
}

```

9.6.5 AufgabenAutoArchivierungHostedService – Hintergrunddienst für automatische Archivierung

Pfad: ZeitmanagementTool/Services/AufgabenAutoArchivierungHostedService.cs (komplett)

Der BackgroundService startet einmal pro Tag (ab 09:00 Uhr) und ruft den AufgabenArchivService auf, der erledigte Aufgaben nach Ablauf der konfigurierten Frist archiviert. Die Wieder-Ausführung am selben Tag wird über `_lastRunDate` verhindert. Da BackgroundService ein Singleton ist, werden Datenbankzugriffe über einen `IServiceScopeFactory` in einem eigenen Scope ausgeführt, um den Scoped DbContext nicht über die Lebenszeit eines Singleton zu teilen.

```
public class AufgabenAutoArchivierungHostedService : BackgroundService
{
    private readonly IServiceScopeFactory _scopeFactory;
    private readonly ILogger<AufgabenAutoArchivierungHostedService> _logger;
    private DateOnly? _lastRunDate;

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            var now = DateTime.Now;
            var hasRunToday = _lastRunDate.HasValue && _lastRunDate.Value == DateOnly.FromDateTime(now);
            var shouldRunNow = now.TimeOfDay >= TimeSpan.FromHours(9) && !hasRunToday;

            if (!shouldRunNow)
            {
                var nextRun = GetNextRun(now);
                var delay = nextRun - now;
                if (delay <= TimeSpan.Zero) delay = TimeSpan.FromMinutes(1);
                await Task.Delay(delay, stoppingToken);
                continue;
            }

            try
            {
                using var scope = _scopeFactory.CreateScope();
                var archiveService = scope.ServiceProvider.GetRequiredService<IAufgabenArchivService>();
                var archivedCount = await archiveService.RunAutomaticArchivingAsync(DateTime.Now, stoppingToken);
                _lastRunDate = DateOnly.FromDateTime(DateTime.Now);
            }
            catch { }
        }
    }
}
```

```
        _logger.LogInformation("Automatische Archivierung ausgeführt. Archivierte  
Aufgaben: {Count}", archivedCount);  
    }  
    catch (OperationCanceledException) { break; }  
    catch (Exception ex)  
    {  
        _logger.LogError(ex, "Fehler bei der automatischen Aufgaben-Archivie-  
rung.");  
    }  
}  
}
```

9.6.6 CanAccessTask – Rollenbasierte Zugriffsprüfung auf Aufgabenebene

Pfad: ZeitmanagementTool/Controllers/AufgabenController.cs (Z. 917–938)

Die zentrale Methode CanAccessTask kapselt die fachlichen Zugriffsregeln auf Aufgabenebene und wird in mehreren Controllern (Aufgaben-, Zeiterfassungs-, Kommentar-Controller) wiederverwendet. Manager und Admins erhalten Zugriff auf alle Aufgaben; reine Mitarbeiter nur dann, wenn sie als Hauptzugewiesener oder als zusätzlicher Zugewiesener erfasst sind. Damit wird die Trennung zwischen rollenbasierter (IsInRole) und ressourcenbasierter Autorisierung sauber umgesetzt.

```
private bool IsEmployeeOnly()
{
    return User.IsInRole(RoleNames.Employee) && !IsManagerOrAdmin();
}

// Zugriffsschutz: Employee nur zugewiesene Aufgabe, Management alle.
private bool CanAccessTask(Aufgabe aufgabe)
{
    if (IsManagerOrAdmin())
    {
        return true;
    }

    if (!IsEmployeeOnly())
    {
        return false;
    }

    var currentUserId = _userManager.GetUserId(User);
    return !string.IsNullOrEmpty(currentUserId)
        && (aufgabe.ZugewiesenerBenutzerId == currentUserId
            || aufgabe.ZugewieseneBenutzer.Any(u => u.Id == currentUserId));
}
```