HTL - Perg
Höhere Abteilung für Informatik

Diplomarbeit

XCALC

Projektteam: Simon Brunner
Daniel Matzinger
Projektbetreuer: Prof. Ing. Dominik Raffetseder, MSc

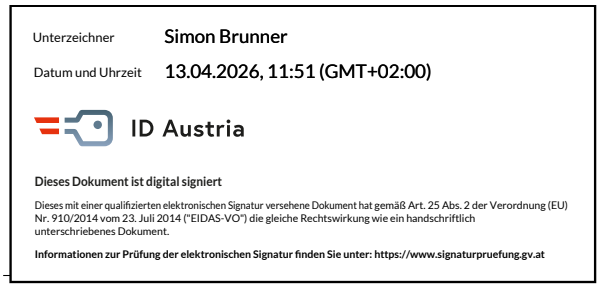
In Zusammenarbeit mit X Architekten GesmbH
Betreuer: Arch DI Frau Bettina Brunner-Krenn

Bearbeitungszeitraum: 07.07.2025 – 26.03.2026

Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Bei der Erstellung der Arbeit haben wir die generativen KI-Tools Gemini, Claude und QuillBot zu folgendem Zweck verwendet: zur Rechtschreib- und Grammatikprüfung, zur sprachlichen und stilistischen Überarbeitung der selbst verfassten Rohtexte sowie zur Rechercheunterstützung.

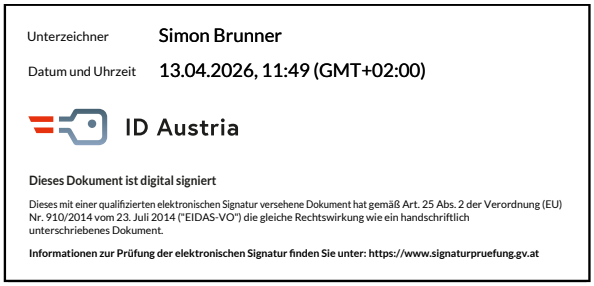


Simon Brunner

Daniel Matzinger

Gendererklärung

Im Rahmen dieser Diplomarbeit wird aus Gründen der besseren Lesbarkeit auf die gleichzeitige Verwendung der weiblichen sowie der männlichen Form verzichtet. Alle Personenbezeichnungen beziehen sich gleichermaßen auf alle Geschlechter. Diese Vereinfachungen dienen ausschließlich der Lesbarkeit und sollen in keiner Weise Diskriminierung oder Benachteiligung bestimmter Geschlechter implizieren.


Unterzeichner Simon Brunner
Datum und Uhrzeit 13.04.2026, 11:49 (GMT+02:00)
ID Austria
Dieses Dokument ist digital signiert
Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr. 910/2014 vom 23. Juli 2014 ("EIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.
Informationen zur Prüfung der elektronischen Signatur finden Sie unter: https://www.signaturpruefung.gv.at

Simon Brunner


Unterzeichner Daniel Matzinger
Datum und Uhrzeit 13.04.2026, 11:56 (GMT+02:00)
ID Austria
Dieses Dokument ist digital signiert
Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr. 910/2014 vom 23. Juli 2014 ("EIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.
Informationen zur Prüfung der elektronischen Signatur finden Sie unter: https://www.signaturpruefung.gv.at

Daniel Matzinger

Danksagung

Wir möchten allen, die uns während der Erstellung dieser Diplomarbeit unterstützt und zum Erfolg des Projekts beigetragen haben, unseren Dank aussprechen.

Ein besonderer Dank geht an unsere Auftraggeberin, Arch. DI Bettina Brunner-Krenn, für das Vertrauen, die tolle Zusammenarbeit und die wertvolle Hilfe bei der Umsetzung des Projekts.

Ein weiterer Dank geht an unseren Betreuungslehrer, Herrn Prof. Ing. Dominik Raffetseder, MSc. Seine fachliche Expertise, die konstruktiven Ratschläge und seine Hilfe bei organisatorischen Fragen waren eine enorme Unterstützung für uns.

Auch möchten wir allen Lehrkräften der HTL Perg danken, die uns in den fünf Jahren unserer Ausbildung das notwendige Fundament für diese Arbeit vermittelt haben.

Abstract

Task

The objective of this diploma thesis is to develop a centralized calculation tool for the architectural firm X Architekten ZT GmbH. The application consolidates construction cost estimates, fee calculations and fee distributions into a single unified system. The final result of these calculations indicates how many hours an employee should spend on a given project. The results can be exported as a PDF document whose layout conforms to the company's corporate identity. The solution is primarily implemented as a web application and is additionally available as a desktop application.

Implementation

The implementation began with a requirements analysis aimed at consolidating calculations that were previously spread across multiple Excel files into one central system. After logging in, the application provides a dashboard for managing existing projects and creating new ones. A structured process guides the user through three calculation steps, culminating in a final summary with PDF export. In addition, an administrative interface enables the management of users and system-wide calculation parameters.

Result

The application was fully developed as part of this diploma thesis and will be handed over to the client, X Architekten ZT GmbH, upon completion of the project. Following the handover, selected employees will be trained in a presentation, and an electronic manual will document the application's usage as well as the calculation workflow. Simon Brunner will remain available for future maintenance and support. No additional costs will be incurred by the company, as the software was developed free of charge within the scope of this thesis.

Kurzfassung

Aufgabenstellung

Ziel dieser Diplomarbeit ist die Entwicklung eines zentralen Kalkulationstools für das Architektenbüro X Architekten ZT GmbH. Die Anwendung soll Baukostenermittlungen, Honorarberechnungen und Honorarverteilungen in einem einheitlichen System zusammenführen. Das Endergebnis dieser Berechnungen gibt an, wie viele Stunden ein Mitarbeiter an einem Projekt aufwenden soll. Die Ergebnisse sollen als PDF exportiert werden können, deren Gestaltung der Corporate Identity des Unternehmens entspricht. Die Lösung wird primär als Web-Anwendung umgesetzt und steht zusätzlich als Desktop-Anwendung zur Verfügung.

Realisierung

Die Umsetzung begann mit einer Anforderungsanalyse, um die zuvor in verschiedenen Excel-Tabellen verstreuten Berechnungen in einem zentralen System zu bündeln. Nach der Anmeldung bietet die Anwendung ein Dashboard zur Projektverwaltung sowie zur Neuanlage von Projekten. Ein strukturierter Prozess führt die Benutzer durch die drei Berechnungsschritte bis hin zur finalen Zusammenfassung mit PDF-Export. Zusätzlich ermöglicht eine Administrationsebene die effiziente Verwaltung von Benutzern und Basisdaten.

Ergebnis

Die Anwendung wurde im Rahmen der Diplomarbeit vollständig entwickelt und wird nach Abschluss des Projektes dem Auftraggeber X Architekten übergeben. Simon Brunner steht nach der Übergabe für zukünftige Fragen und Probleme zur Verfügung. Für das Unternehmen fallen keine zusätzlichen Kosten an, da die Anwendung kostenfrei entwickelt wurde.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

1.1.1 Geschäftlicher Hintergrund

Den Hintergrund dieser Diplomarbeit bildet der zunehmende Bedarf an effizienten, einheitlichen und nachvollziehbaren digitalen Lösungen im Architektur- und Planungsbereich. Insbesondere die Honorarermittlung, die Baukostenschätzung sowie die Stundenkalkulation stellen wichtige Grundlagen für eine wirtschaftliche Projektabwicklung dar. Der Kooperationspartner X Architekten ZT GmbH bearbeitet Projekte unterschiedlicher Größenordnungen und Schwierigkeitsklassen, weshalb konsistente Berechnungen von großer Bedeutung sind. Die aktuell beim Kooperationspartner eingesetzten Werkzeuge stellen jedoch keine durchgehende Gesamtlösung dar. Die Honorar- und Baukostenermittlung erfolgt über uneinheitliche Excel-Tabellen, während die Stundenkalkulation über ein externes Online-Tool abgewickelt wird. Diese Trennung führt zu Inkonsistenzen, redundanter Datenhaltung und eingeschränkter Nachvollziehbarkeit der Berechnungen. Der daraus resultierende Mehraufwand wirkt sich negativ auf die Produktivität aus. An dieser Stelle setzt XCalc an, das im Rahmen dieser Diplomarbeit entwickelt wurde. Durch die Zusammenführung aller drei Bereiche in einer zentralen Anwendung können Arbeitsschritte reduziert und die Berechnungsgrundlagen vereinheitlicht werden.

1.1.2 Projektauslöser

Die in Abschnitt 1.1.1 beschriebene Trennung der Werkzeuge führte zu dem Wunsch des Kooperationspartners nach einer zentralen Lösung. Daher entstand in Abstimmung mit X Architekten der Bedarf nach einer internen, webbasierten Anwendung, die alle relevanten Kalkulationsbereiche vereint, auf den Firmenrechnern verfügbar ist und die bestehenden Werkzeuge vollständig ablöst. Dieser Bedarf bildete den Ausgangspunkt für die Entwicklung von XCalc im Rahmen dieser Diplomarbeit.

1.2 Zielsetzung

1.2.1 Geschäftsziel

Das Geschäftsziel dieser Diplomarbeit besteht darin, die wirtschaftliche Projektabwicklung beim Kooperationspartner durch konsistente, nachvollziehbare und effiziente Kalkulationsprozesse zu verbessern. Durch die zentrale Durchführung von Honorarermittlung, Baukostenschätzung und Stundenkalkulation soll der Aufwand reduziert, die Konsistenz der Berechnungsergebnisse erhöht und die Produktivität gesteigert werden.

1.2.2 Projektvorgehensziele

Zu den wichtigsten Zielen bei der erfolgreichen Entwicklung von *XCalc* gehören:

1. **Anforderungsdefinition**
   Klare Definition der funktionalen und nicht-funktionalen Anforderungen an das Kalkulationstool in Abstimmung mit dem Kooperationspartner.
2. **Technologieauswahl**
   Auswahl der geeigneten Technologien für Backend, Datenbank und Frontend zur Umsetzung einer webbasierten Anwendung.
3. **Implementierung der Baukostenermittlung**
   Umsetzung einer strukturierten Baukostenschätzung auf Basis von Flächen- und Kostenansätzen.
4. **Implementierung der Honorarermittlung**
   Entwicklung einer konsistenten Honorarberechnung auf Basis der Bauwerkskosten und Leistungsparameter.
5. **Implementierung der Stundenkalkulation**
   Berechnung des Stundenaufwands pro Mitarbeiterklasse .
6. **Implementierung des PDF-Exports**
   Export der wesentlichen Informationen aus den Berechnungen in ein PDF.
7. **Testung und Qualitätssicherung**
   Testung der Anwendung zur Sicherstellung korrekter Berechnungsergebnisse.

1.3 Projektinhalt - Überblick

XCalc vereint drei Kalkulationsbereiche in einer zentralen, webbasierten Anwendung. Die folgende Übersicht beschreibt die wesentlichen Funktionsbereiche des Systems.

1.3.1 Baukostenermittlung

Die Baukostenermittlung bildet das Fundament für die nachfolgenden Berechnungen. Bauwerkskosten werden auf Basis definierter Flächenarten und Kostenansätze erfasst und berechnet. Die strukturierte Eingabe der Parameter gewährleistet eine nachvollziehbare Ermittlung der Baukosten, die als Ausgangspunkt für die Honorarberechnung dient.

1.3.2 Honorarermittlung

Aufbauend auf den ermittelten Bauwerkskosten erfolgt die Honorarermittlung. Projektspezifische Parameter wie Schwierigkeitsklasse und Leistungsumfang fließen in die Berechnung ein und ermöglichen eine einheitliche und transparente Honorarberechnung.

1.3.3 Stundenkalkulation

Auf Basis der ermittelten Honorarsumme wird der Stundenaufwand eines Projekts berechnet. Dies unterstützt die interne Ressourcenplanung und ermöglicht eine realistische Einschätzung des Projektaufwands.

1.3.4 Ergebnisdarstellung und Export

Die Ergebnisse aller drei Kalkulationsbereiche werden am Ende des Berechnungsworkflows zusammengefasst dargestellt. Eine integrierte Exportfunktion gibt die relevanten Ergebnisse als strukturiertes, Corporate-Identity-konformes (CI-konformes) PDF aus.

1.4 Projektumfeld

1.4.1 Projektteam

Unser Team besteht aus zwei Schülern der HTL Perg - Abteilung für Höhere Informatik.

Simon Brunner

Projektleiter

Daniel Matzinger

Projektmitglied

1.4.2 Auftraggeber

Unser Auftraggeber ist die X Architekten ZT GmbH mit Hauptsitz in Linz sowie weiteren Standorten in Wien und Lambach. Das lösungsorientierte Unternehmen wurde ursprünglich 1996 gegründet und agiert als staatlich befugtes sowie beeidetes Ziviltechnikerbüro.

Das Gebiet von X Architekten umfasst ein breites Spektrum der Architektur, das von Wohn- und Bürobau über Industrie- und Gewerbeprojekte bis hin zu Städtebau und Design reicht.

Abbildung 1: Logo X Architekten

Quelle: Von Auftraggeber erhalten

1.4.3 Betreuung

Unsere Diplomarbeit wird von Herrn Professor Raffetseder betreut. Seit 2019 unterrichtet er an der HTL Perg.

Professor Raffetseder begleitet uns bereits seit der 3. Klasse an der HTL Perg. Er unterrichtete uns in der 3. und 4. Klasse im Fach Software Engineering (Java) und unterrichtet uns aktuell im Fach Webprogrammierung. Durch den langjährigen Unterricht kennt er unsere Arbeitsweise, sowie unsere Stärken und Schwächen, sehr gut.

Mit seiner Erfahrung in den Bereichen Webprogrammierung, Datenbanken und Projektentwicklung unterstützte er uns bei Fragen und bei der Umsetzung unserer Diplomarbeit.

Abbildung 2: Prof. Ing. Dominik Raffetseder, MSc
Quelle: https://www.htl-perg.ac.at

2 Grundlagen und Methoden

Für die Umsetzung der Diplomarbeit werden bewährte Technologien der web-basierten Softwareentwicklung genutzt. Das Hauptziel ist es, das User Interface, die Anwendungslogik und die Datenhaltung klar zu entkoppeln. Diese Architektur sorgt dafür, dass die Anwendung wartbar ist und eine transparente Struktur hat.

2.1 Verwendete Technologien

Die Wahl der Technologien wurde durch die Aufgabenstellung, die Vorgaben des Auftraggebers und die im Schulunterricht behandelten Grundlagen beeinflusst. Das Gesamtbild der Anwendung und die Kommunikationswege zwischen den verschiedenen Schichten sind in Abbildung 3 dargestellt.

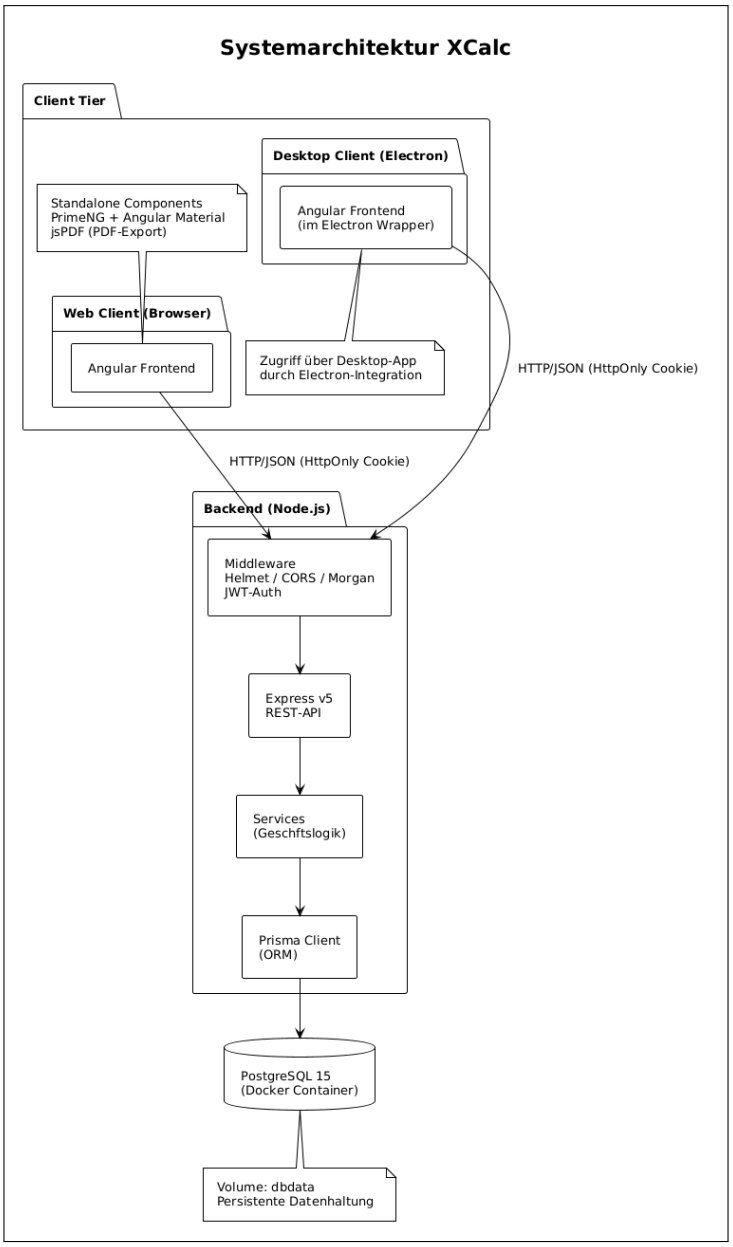


Abbildung 3: Systemarchitektur von XCalc

Quelle: eigene Darstellung, gerendert mit PlantUML [1]

2.1.1 Angular

Angular ist ein auf TypeScript basierendes Open-Source-Framework von Google zur Entwicklung von Single-Page Applications [2]. Es basiert auf einer komponentenbasierten Architektur, bei der jede Komponente eine TypeScript-Klasse, ein HTML-Template und ein optionales Stylesheet vereint. Angular verfügt zudem über ein integriertes Dependency-Injection-System, das die Verwaltung von Abhängigkeiten zwischen Komponenten und Services vereinfacht.

2.1.2 Typescript

TypeScript ist eine typisierte Obermenge von JavaScript und ermöglicht statische Typprüfungen zur Entwicklungszeit [3]. Dadurch können viele Fehler bereits vor der Laufzeit erkannt und behoben werden, was die Codequalität und Wartbarkeit erhöht.

2.1.3 jsPDF

Für die clientseitige Erzeugung von PDF-Dokumenten kommt die JavaScript-Bibliothek jsPDF zum Einsatz [4]. Sie ermöglicht es, PDF-Dateien direkt im Browser zu generieren, ohne dass serverseitige Infrastruktur erforderlich ist.

2.1.4 Electron

Für die Desktop-Variante der Anwendung kommt Electron zum Einsatz [5]. Electron ermöglicht es, Webanwendungen als native Desktop-Programme für Windows, macOS und Linux bereitzustellen. Die Angular-Anwendung wird dabei in einem eingebetteten Chromium-Browser ausgeführt. Auf diese Weise kann die bestehende Webapplikation ohne grundlegende Anpassungen als installierbare Desktop-Anwendung bereitgestellt werden.

2.1.5 Node.js

Die serverseitige Anwendungslogik wird mit Node.js umgesetzt [6]. Node.js ist eine asynchrone, ereignisgesteuerte Laufzeitumgebung, die auf der V8-JavaScript-Engine von Google basiert. Durch das nicht-blockierende I/O-Modell eignet sich Node.js besonders für den gleichzeitigen Umgang mit mehreren API-Anfragen.

2.1.6 Express

Für die Strukturierung des Backends wird Express genutzt [7]. Das Framework vereinfacht die Erstellung von Webservern und REST-APIs über ein Middleware-Konzept: Eingehende HTTP-Anfragen durchlaufen eine Abfolge von Verarbeitungsschritten, bevor eine Antwort erfolgt.

2.1.7 PostgreSQL

Für die dauerhafte Speicherung der Daten kommt PostgreSQL zum Einsatz [8]. Es bietet hohe Zuverlässigkeit, strenge Datenintegrität und hält sich konsequent an den SQL-Standard. Darüber hinaus unterstützt PostgreSQL komplexe Abfragen, Transaktionen und referenzielle Integrität

2.1.8 Prisma

Für den Datenbankzugriff wird Prisma als Object Related Mapping (ORM) für Node.js und TypeScript eingesetzt [9]. Im Gegensatz zu herkömmlichen ORMs legt Prisma das Datenbankschema in einer separaten, deklarativen Schemadatei (`schema.prisma`) fest, aus der ein vollständig typisierter Client generiert wird. Dieser Client kann direkt in TypeScript verwendet werden und bietet so typsichere Datenbankabfragen ohne manuell geschriebenes SQL.

2.1.9 Docker

Docker wird verwendet, um die Datenbankumgebung bereitzustellen [10]. Docker ermöglicht es, Anwendungen in Containern zu betreiben, die zusammen mit allen notwendigen Abhängigkeiten verpackt sind.

2.2 Verwendete Entwicklungssysteme

Neben den zuvor beschriebenen Technologien sind während der Entwicklung von XCalc mehrere Werkzeuge zum Einsatz gekommen, die den Arbeitsablauf im Team unterstützen.

2.2.1 Visual Studio Code

Als Code-Editor für die Frontend- und Backend-Entwicklung wurde Visual Studio Code (VS Code) verwendet [11]. Der quelloffene Editor von Microsoft überzeugt mit einer schnellen Startzeit, einer integrierten Terminalumgebung und der IntelliSense-Autovervollständigung für TypeScript. Zusätzlich haben wir in VS Code direkt mit Git gearbeitet und die Angular Language Service Extension für die Autovervollständigung in HTML-Templates installiert.

2.2.2 GitHub

Für die Versionskontrolle und das Teamwork wurde GitHub genutzt [12]. Frontend und Backend des Projekts lagen in einem gemeinsamen Repository. Die beiden Teamkollegen haben auf unterschiedlichen Branches gearbeitet und ihre Änderungen über Pull Requests in den Hauptbranch eingefügt [13].

2.2.3 Swagger

Die Dokumentation der REST-API des Backends erfolgt automatisch über Swagger [14]. Swagger generiert eine interaktive Dokumentation basierend auf den JSDoc-Kommentaren der Routen-Dateien, die unter dem Pfad `/docs` des laufenden Servers zu finden ist, und zwar gemäß der OpenAPI-Spezifikation.

2.2.4 Postman

Postman wurde zum Testen der REST-API verwendet [15]. Während der Entwicklung des Backends wurden alle API-Endpunkte zuerst in Postman getestet, bevor sie in das Angular-Frontend eingebunden wurden. Collections ermöglichen es, Anfragen zu gruppieren und sie mit Variablen zu parametrisieren.

2.2.5 Docker Desktop

Zur Verwaltung der PostgreSQL-Datenbank auf lokaler Ebene kam Docker Desktop zum Einsatz [16]. Mit dem Befehl `docker-compose up` wird der PostgreSQL-Container gestartet. Es war praktisch für die Entwicklung, weil die Datenbank mit einem einzigen Befehl gestartet und bei Bedarf zurückgesetzt werden konnte, ohne dass man PostgreSQL direkt auf dem Rechner installieren musste.

3 Projektplanung

3.1 Projektorganisation

Das XCalc-Team besteht aus zwei Schülern der HTL Perg. Dabei sind die Teilbereiche des Projektes klar aufgeteilt, wobei sich gegenseitig ausgeholfen und beraten wird, damit keine einseitigen Rückstände entstehen. Zur Entwicklungszeit im Sommer 2025 im Büro des Auftraggebers ist jede Woche in kurzen Worten besprochen worden, was zu erledigen ist, und es sind sonstige Bemerkungen bekannt gemacht worden. Nach vier Wochen wurde das bisherige Ergebnis dem Auftraggeber gezeigt und ab dann von zu Hause aus weiterentwickelt.

Die Planung erfolgte mit dem Online-Tool "Trello".[17]

3.2 Projektzeitplan

Die Aufgaben wurden zeitlich geplant und umgesetzt, indem wir die Projektmanagement-Methode *Scrum [5]* genutzt haben. Das Tool *Trello* wurde dabei eingesetzt, um den Fortschritt transparent zu dokumentieren und die Aufgaben effizient zu koordinieren [17]. So konnten der aktuelle Status jedes Arbeitspakets jederzeit überwachen und flexibel auf neue Anforderungen reagieren.

3.2.1 Aufgabenverwaltung mit Trello

Unser Trello-Board wurde in sechs spezifische Spalten unterteilt, um den gesamten Lebenszyklus einer Aufgabe abzubilden:

1. **Backlog:** Hier wurden alle langfristig geplanten Features und Anforderungen gesammelt.
2. **Sprint Backlog:** Enthält jene Aufgaben, die für den aktuellen Arbeitszeitraum ausgewählt wurden.
3. **Doing:** Aufgaben, die sich aktuell in aktiver Bearbeitung befinden.
4. **Code Review:** Sobald die Programmierung abgeschlossen war, wurde der Code von einem anderen Teammitglied geprüft, um die Qualität sicherzustellen.
5. **Testing:** Nach dem Review wurden die Funktionen in einer Testumgebung auf Fehler und Korrektheit geprüft.
6. **Done:** Erst nach erfolgreichem Test wurden die Karten in diese Spalte verschoben.

Abbildung 4 zeigt den Aufbau unseres Projektboards während der Entwicklungsphase.

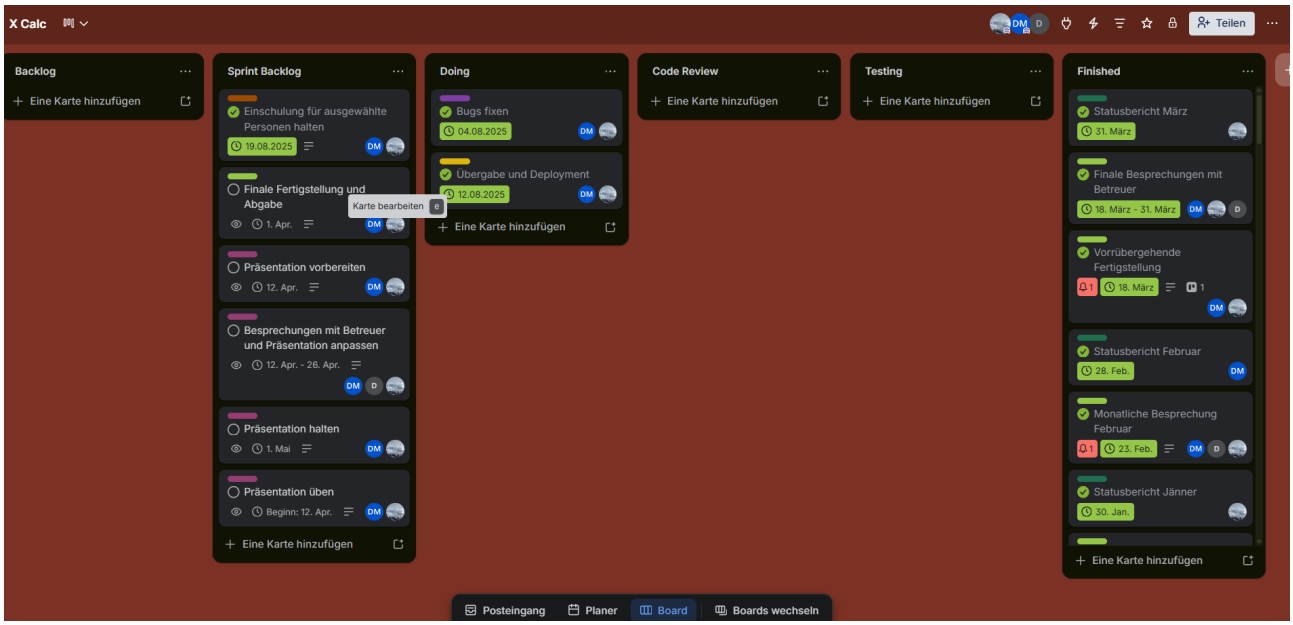


Abbildung 4: Trello-Board zur Verwaltung der Arbeitspakete im Scrum-Prozess

3.2.2 Vorgehensweise und Feedbackschleifen

Im Gegensatz zum klassischen Wasserfallmodell setzen wir auf kurze Iterationen und einen kontinuierlichen Austausch mit unserer Auftraggeberin, Frau Bettina Brunner.

Wöchentliche Springs Unsere Arbeitsphasen waren in einwöchige Intervalle unterteilt. Zu Beginn jeder Woche wurden die Ziele im Sprint Backlog definiert.

Jour Fixe mit dem Auftraggeber Ein zentraler Bestandteil unseres Prozesses war das wöchentliche Meeting mit der Auftraggeberin. In diesen Terminen präsentierten wir die Ergebnisse der vergangenen Woche direkt am System.

Feedback und Anpassung Das unmittelbare Feedback aus den Präsentationen floss direkt in das Backlog für die darauffolgende Woche ein. Dadurch konnten Fehlentwicklungen frühzeitig vermieden und das Design des Tools laufend an die Corporate Identity (CI) des Unternehmens angepasst werden.

4 Implementierung

4.1 Technischer Überblick

XCalc wird als webbasierte Full-Stack-Anwendung auf Basis eines konsistenten TypeScript-Stacks umgesetzt. Das Frontend basiert auf dem Framework `Angular`, das eine modulare und reaktive Benutzeroberfläche ermöglicht. Die serverseitige Logik wird durch ein `Node.js`-Backend mit dem Web-Framework `Express` realisiert, das über eine REST-Schnittstelle mit dem Frontend kommuniziert. Für die persistente Datenhaltung kommt `PostgreSQL` zum Einsatz; der Datenbankzugriff sowie die Typsicherheit werden durch das ORM `Prisma` gewährleistet. Um eine einheitliche Ausführungsumgebung sicherzustellen, wird die Datenbankumgebung mit `Docker` containerisiert [10]. Für den Einsatz auf den Firmenrechnern des Kooperationspartners steht zusätzlich eine Desktop-Variante zur Verfügung, die mithilfe von `Electron` realisiert wird.

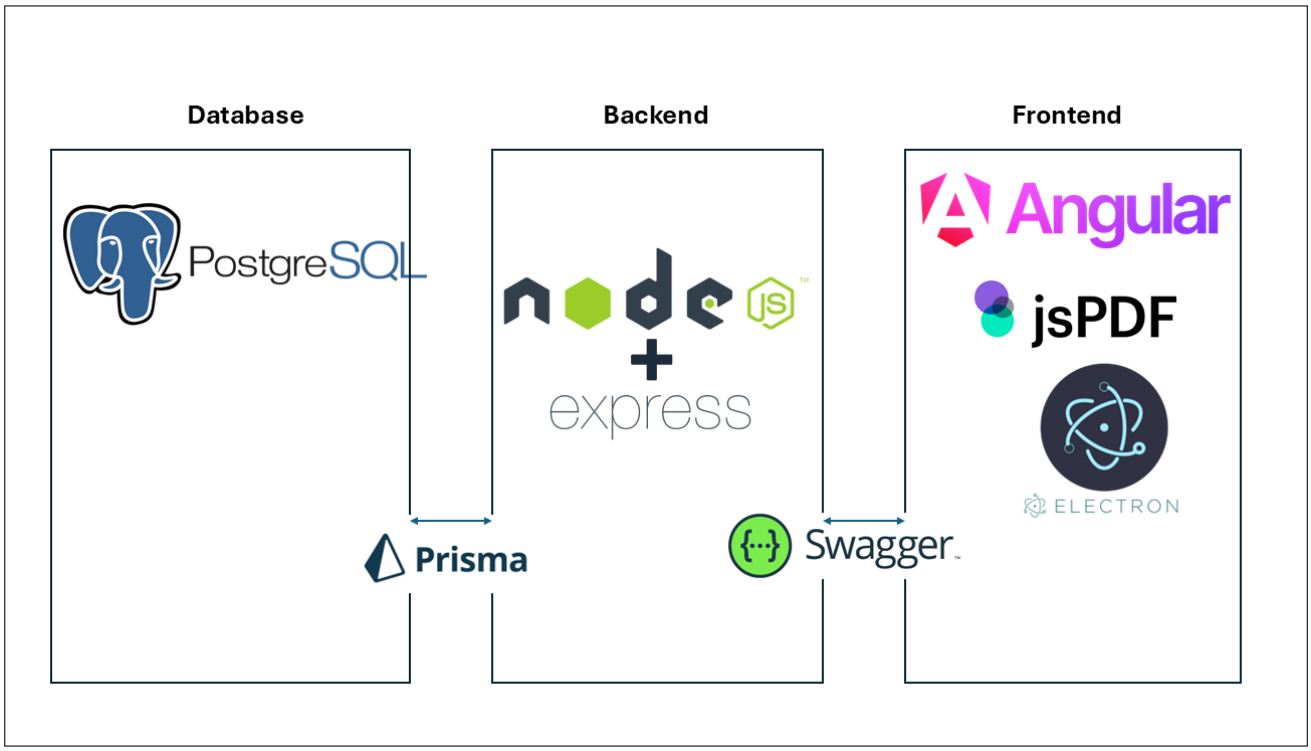


Abbildung 5: Technischer Überblick über das System XCalc

4.2 Datenbank

XCalc nutzt eine PostgreSQL-Datenbank zur Datenhaltung, die über das ORM-Framework Prisma angebunden ist. Alle projektbezogenen Berechnungsdaten, Benutzerdaten und systemweiten Basisdaten werden in der Datenbank gespeichert. Das Schema ist in drei Abschnitte gegliedert, die den Berechnungsbereichen der Anwendung entsprechen: Baukostenermittlung, Honorarberechnung und Honorarverteilung.

4.2.1 Schema der Datenbank

In der Datei `schema.prisma` ist das Schema mit acht Modellen festgelegt. Prisma erstellt daraus den typsicheren Datenbankclient und die TypeScript-Typen, die im gesamten Backend Anwendung finden. Das Entity-Relationship-Diagramm in Abbildung 6 zeigt die Modelle und ihre Beziehungen.

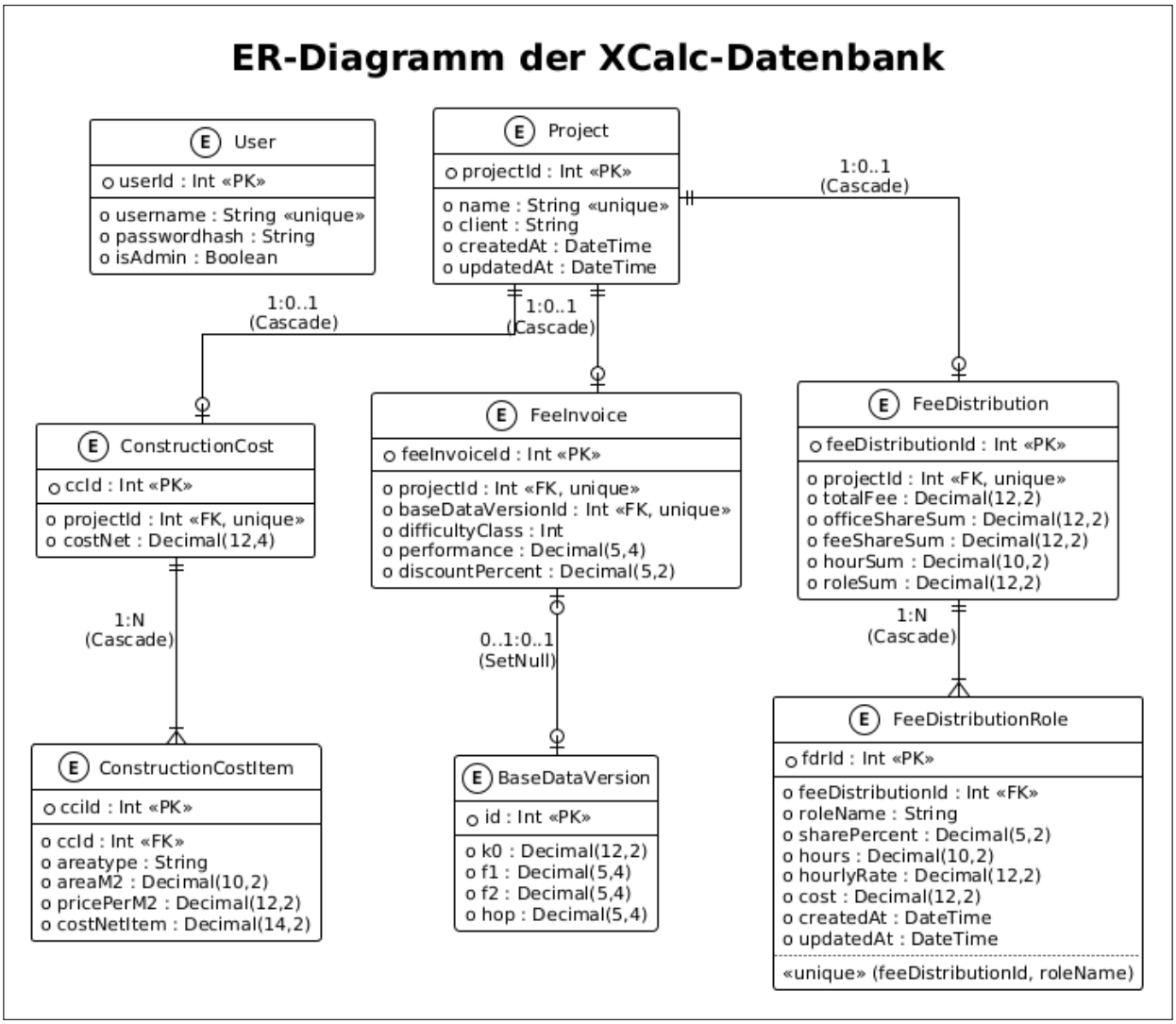


Abbildung 6: ER-Diagramm XCalc-Datenbank

User

Die Benutzerverwaltung wird durch das `User`-Modell abgebildet. Jeder Benutzer hat einen eindeutigen Benutzernamen (`username`) sowie einen gehashten Passwortwert (`passwordhash`). Das boolesche Feld `isAdmin` unterscheidet zwischen normalen Benutzern und Administratoren. Der Primärschlüssel `userId` wird automatisch erhöht.

Project

Das zentrale Modell der Anwendung ist `Project`. Es speichert den Projektnamen sowie den Auftraggeber (`client`). Einem Projekt sind genau eine Baukostenermittlung, eine Honorarberechnung und eine Honorarverteilung zugeordnet. Diese drei Beziehungen sind als optionale

1:1-Relationen modelliert, da ein neu angelegtes Projekt zunächst keine Berechnungsdaten enthält. Die Felder `createdAt` und `updatedAt` werden von Prisma automatisch verwaltet.

ConstructionCost und ConstructionCostItem

Zur Ermittlung der Baukosten dienen zwei Modelle. `ConstructionCost` ist über einen Fremdschlüssel mit genau einem Projekt verknüpft und speichert die Gesamtnettosumme (`costNet`) als `Decimal(12,4)`. Die einzelnen Baukostenpositionen werden im Modell `ConstructionCostItem` abgebildet. Jeder Eintrag enthält eine Flächenart (`areatype`), die Fläche in Quadratmetern (`areaM2`), den Preis pro Quadratmeter (`pricePerM2`) sowie die resultierende Nettosumme der Position (`costNetItem`). Die Beziehung zwischen beiden Modellen ist 1:N (eine Baukostenermittlung kann mehrere Positionen umfassen).

BaseDataVersion und FeeInvoice

Die Honorarberechnung wird im Modell `FeeInvoice` abgebildet. Es enthält die Schwierigkeitsklasse (`difficultyClass`), den beauftragten Leistungsanteil (`performance`) sowie einen optionalen Nachlass in Prozent (`discountPercent`), der standardmäßig den Wert 0 hat. Die für die Honorarberechnung erforderlichen Basisdaten – der Grundwert `k0`, die Faktoren `f1` und `f2` sowie der Honorarordnungsparameter `hop` – sind in einem separaten Modell `BaseDataVersion` ausgelagert. Dieses steht in einer optionalen 1:1-Beziehung zu `FeeInvoice`. Die Auslagerung ermöglicht es, Basisdaten zentral zu verwalten und bei Bedarf anzupassen, ohne bestehende Honorarberechnungen zu beeinflussen.

FeeDistribution und FeeDistributionRole

Das berechnete Gesamthonorar wird durch die Honorarverteilung auf die einzelnen Mitarbeiterrollen aufgeteilt. Das `FeeDistribution` Modell speichert Summenwerte: Gesamthonorar (`totalFee`), Büroanteil (`officeShareSum`), Honoraranteil (`feeShareSum`), Gesamtstunden (`hourSum`) und Rollensumme (`roleSum`). Die einzelnen Rollen werden im Modell `FeeDistributionRole` abgebildet, das je Rolle den Namen, den prozentualen Anteil, die zugewiesenen Stunden, den Stundensatz sowie die resultierenden Kosten enthält. Ein Unique-Constraint auf der Kombination aus `feeDistributionId` und `roleName` stellt sicher, dass innerhalb einer Verteilung keine zwei Rollen denselben Namen tragen können. Das Feld `createdAt` wird beim Anlegen automatisch mit dem aktuellen Zeitstempel befüllt, `updatedAt` wird bei jeder Änderung aktualisiert.

4.2.2 Unversehrtheit der Referenzen

Kaskadierende Löschregeln (`onDelete: Cascade`) kommen für alle Beziehungen zwischen Projekten und ihren Berechnungsdaten zum Einsatz, die diese Beziehungen betreffen. Wird ein Projekt aus der Datenbank entfernt, so werden auch die dazugehörige Baukostenermittlung, die Honorarberechnung und die Honorarverteilung automatisch gelöscht, ebenso wie alle untergeordneten Datensätze, wie Baukostenpositionen und Rollenzuweisungen. Auf diese Weise können keine verwaisten Datensätze entstehen. Die Beziehung zwischen `FeeInvoice` und `BaseDataVersion` ist hier die einzige Ausnahme: Die Löschregel ist auf `SetNull` eingestellt, da Basisdatenversionen unabhängig von einzelnen Projekten existieren und nicht gelöscht werden sollen.

4.2.3 Dezimalgenauigkeit

In PostgreSQL sind Dezimaltypen mit fester Genauigkeit für alle Arten von Geldwerten und Prozentwerten gedacht. So wird die Gesamtsumme der Baukosten (costNet) im Format Decimal(12,4) gespeichert, während die einzelnen Baukostenpositionen die Felder pricePerM2 als Decimal(12,2) und costNetItem als Decimal(14,2) verwenden. Prozentsätze werden als Decimal(5,2) oder Decimal(5,4) abgelegt, Stunden als Decimal(10,2) und Stundensätze als Decimal(12,2). Um Rundungsfehler zu vermeiden, die zu falschen Ergebnissen führen können, sind Dezimaltypen die bessere Wahl im Vergleich zu Gleitkommazahlen.

4.2.4 Deployment (Bereitstellung)

Die PostgreSQL-Datenbank kann über Docker betrieben werden. Im Backend-Verzeichnis legt die Datei `docker-compose.yml` einen Service fest. Die Zugangsdaten (Benutzername, Passwort, Datenbankname) werden durch Umgebungsvariablen definiert. Ein benanntes Volume (`dbdata`) sorgt dafür, dass die Daten beim Neustart des Containers erhalten bleiben. Die Datenbank ist über Port 5432 erreichbar. Während der Entwicklung wird das Schema mit dem Befehl `prisma migrate dev` angelegt, der neue Migrationen erzeugt und anwendet. Im Produktivbetrieb kommt stattdessen `prisma migrate deploy` zum Einsatz (siehe `scripts/migrate.sh`), das ausschließlich bestehende Migrationen ausführt, ohne neue zu generieren.

4.3 REST-API

Das Angular-Frontend nutzt eine REST-API, die in TypeScript mit Express 5 implementiert wurde, um mit dem Backend zu kommunizieren. Die API ist nach einer dreischichtigen Architektur aufgebaut: Routen empfangen HTTP-Anfragen und validieren die Eingabe, Services beherbergen die Geschäftslogik, und der Prisma Client ist für die Datenbankzugriffe zuständig.

4.3.1 Middleware-Kette

Bevor eine eingehende Anfrage den Route-Handler erreicht, durchläuft sie folgende Middleware-Funktionen: Der Request-Body wird durch `express.json()` als JSON geparst, `helmet()` fügt sicherheitsrelevante HTTP-Header hinzu, `cors()` erlaubt domainübergreifende Anfragen vom Frontend, und `morgan()` protokolliert jede Anfrage und sendet sie an den Logger Winston.

4.3.2 Authentifizierung und Autorisierung

Die Authentifizierung läuft über JSON Web Tokens (JWT). Nach dem erfolgreichen Login über `POST /auth/login` wird ein signiertes Token erzeugt, das zwei Stunden lang gültig ist und die Benutzer-ID sowie den Admin-Status enthält. Das Token wird als HttpOnly-Cookie an den Client übermittelt . Die Middleware `authenticate` überprüft das Token bei jeder Anfrage und weist ungültige Tokens mit einem Statuscode von 401 zurück. Für Admin-Endpunkte kontrolliert die zusätzliche Middleware `requireAdmin` den Status als Administrator und liefert bei fehlender Berechtigung den Statuscode 403. Passwörter werden mit *bcrypt* und einem Kostenfaktor von 12 sicher gehasht.

Die serverseitige Middleware `authenticate` entnimmt das Token aus dem HTTP-Header, überprüft es und fügt die Benutzerdaten dem Request-Objekt hinzu. Die Implementierung ist im Quellcode 1 zu sehen.

Listing 1: JWT-Authentifizierungs-Middleware (auth.ts)

```
export const authenticate = (
  req: AuthRequest, res: Response, next: NextFunction
) => {
  const auth = req.headers.authorization;
  if (!auth?.startsWith("Bearer ")) {
    res.status(401).json({ message: "Token fehlt" });
    return;
  }
  try {
    const payload = jwt.verify(auth.substring(7), JWT_SECRET) as {
      userId: number;
      isAdmin: boolean;
    };
    req.user = { userId: payload.userId, isAdmin: payload.isAdmin };
    next();
  } catch {
    res.status(401).json({ message: "Ungueltiges Token" });
  }
};
```

4.3.3 Eingabevalidierung

Zur Validierung von eingehenden Daten wird die Bibliothek *express-validator* eingesetzt [18]. Die Login-Endpunkte überprüfen, ob Benutzername und Passwort ausgefüllt sind. Bei der Registrierung über den Endpunkt POST /users/register wird die Länge des Benutzernamens auf 3 bis 50 Zeichen begrenzt. Die Validierung der Berechnungsdaten erfolgt in den Routen direkt im Route-Handler: Die erwarteten Felder werden auf ihren Typ überprüft (z.B. ob `difficultyClass` eine Zahl ist), und bei ungültigen Daten erfolgt eine Rückgabe des Statuscodes 400.

4.3.4 Service-Schicht

Die Datenverarbeitung findet in der Service-Schicht statt. Für jeden fachlichen Bereich existiert eine Service-Datei (z. B. `projectService.ts`, `constructionCostService.ts`, `feeInvoiceService.ts`). Die Services sind dafür zuständig, den Prisma Client zu initialisieren und die Datenbankoperationen auszuführen. Die Routen bedienen nur Service-Funktionen und geben deren Rückgabewerte als JSON an den Client zurück.

Ein Vorteil dieser Trennung ist, dass die Geschäftslogik unabhängig von HTTP getestet werden kann. Eine Service-Funktion kann über mehrere Routen aufgerufen werden, ohne dass der Code mehrfach geschrieben werden muss.

4.3.5 Codestruktur und Aufbau

Die API gliedert sich in neun Routen-Module, die jeweils einen fachlichen Bereich kapseln. Jedes Modul definiert seine Endpunkte und delegiert die Verarbeitung an eine zugehörige Service-Datei.

Die drei Berechnungsendpunkte (Baukosten, Honorar, Verteilung) verwenden ein Upsert-Muster über den `PUT`-Endpunkt: Existiert der Datensatz noch nicht, wird er angelegt; andernfalls wird er aktualisiert.

Die Implementierung am Beispiel der Baukostenermittlung ist in Quellcode 2 dargestellt. Der Ablauf läuft vollständig innerhalb einer Prisma-Transaktion ab: Zunächst wird der übergeordnete Datensatz angelegt oder abgerufen, anschließend werden alle bestehenden Positionen gelöscht und durch die neuen ersetzt. Abschließend wird die Gesamtsumme per Aggregationsabfrage neu berechnet.

Listing 2: Upsert der Baukostenermittlung in einer Transaktion (constructionCostService.ts)

```
export async function upsertConstructionCost(
  projectId: number,
  items: ConstructionCostItemInput[]
) {
  return prisma.$transaction(async (tx) => {
    const cc = await tx.constructionCost.upsert({
      where: { projectId },
      create: { projectId, costNet: new Prisma.Decimal(0) },
      update: {},
    });

    await tx.constructionCostItem.deleteMany({
      where: { ccId: cc.ccId },
    });

    if (items.length) {
      await tx.constructionCostItem.createMany({
        data: items.map((i) => {
          const price = new Prisma.Decimal(i.pricePerM2);
          return {
            ccId: cc.ccId,
            areatype: i.areatype,
            areaM2: i.areaM2,
            pricePerM2: price,
            costNetItem: price.mul(i.areaM2),
          };
        }),
      });
    }

    const agg = await tx.constructionCostItem.aggregate({
      where: { ccId: cc.ccId },
      _sum: { costNetItem: true },
    });

    return tx.constructionCost.update({
      where: { ccId: cc.ccId },
      data: { costNet: new Prisma.Decimal(agg._sum.costNetItem ?? 0) },
      include: { items: true },
    });
  });
}
```

4.3.6 Fehlerbehandlung

Die API verwendet einen zentralen Fehlerhandler, der am Ende der Middleware-Kette sitzt. Wenn ein Route-Handler oder eine Service-Funktion eine Exception wirft, fängt sie der globale Error-Handler ab, loggt sie über Winston und sendet dem Client eine JSON-Antwort mit Statuscode 500. Auf vorhersehbare Fehler, wie den Prisma-Fehlercode P2025 (Record not found), wird bereits in den Route-Handlern gezielt mit einem Statuscode von 404 reagiert, bevor der globale Handler greifen muss.

4.3.7 API-Dokumentation

JSDoc-Kommentare im OpenAPI-Format sind allen Endpunkten beigefügt. Die Pakete `swagger-jsdoc` und `swagger-ui-express` erzeugen daraus eine interaktive Dokumentation, die unter `/docs` zu finden ist. Bei jedem Serverstart wird die Spezifikation im Projektverzeichnis unter `swagger.json` abgelegt.

4.4 Login und Administratorisches

Über die Login-Seite muss sich ein Benutzer anmelden, bevor er die Berechnungsfunktionen von XCalc nutzen kann. Sobald sich dieser erfolgreich eingeloggt hat, unterscheidet die Anwendung zwischen normalen Benutzern und Administratoren, die zusätzliche Verwaltungsfunktionen nutzen können.

Anmerkung Registrierung Im Backend ist ein Endpunkt zur Benutzerregistrierung implementiert. Während der Entwicklung kam von Seiten des Auftraggebers jedoch die Anforderung, ausschließlich mit zwei vorgefertigten Benutzerkonten zu arbeiten, ohne dass weitere Konten angelegt oder gelöscht werden können. Da der Endpunkt weiterhin in der Codebasis vorhanden ist und Simon Brunner auch nach der Übergabe in Kontakt mit dem Auftraggeber bleibt, kann eine vollständige Registrierungsfunktion bei Bedarf nachträglich aktiviert werden. Abbildung 7 zeigt die zugehörigen API-Endpunkte in der Swagger-Dokumentation.

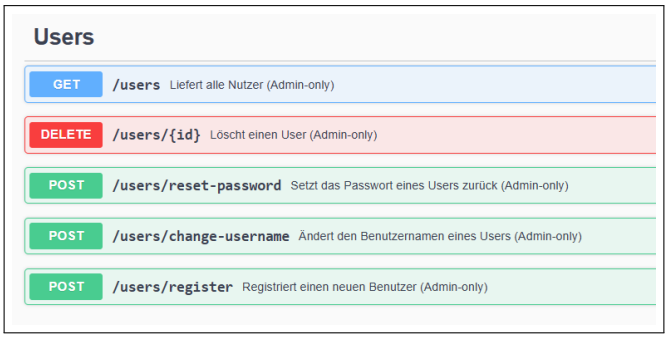


Abbildung 7: Screenshot Swagger API Users

4.4.1 Login-Prozess

Die Login-Seite ist die erste Seite der Anwendung und erscheint, wenn der Anwender die Root-URL besucht. Der Benutzer sieht die Ansicht in Abbildung 35, sobald er die Anwendung öffnet. Sie besteht aus einem Formular mit zwei Eingabefeldern (Nutzername und Passwort) sowie einer Schaltfläche für die Anmeldung. Das Formular verwendet Angulars `FormsModule` sowie Two-Way Data Binding (`ngModel`), was zur Folge hat, dass die Eingaben direkt mit den Variablen der Komponente verknüpft sind.

Abbildung 8: Screenshot Login-Seite XCalc

Frontend-seitig schützt ein `AuthGuard` alle Routen der Anwendung. Er kontrolliert über den `LoginService`, ob der Benutzer angemeldet ist; falls nicht, leitet er ihn zur Login-Seite um. Die Implementierung ist im Quellcode 3 zu finden.

Listing 3: Route Guard für geschützte Bereiche (auth.guard.ts)

```
export const authGuard: CanActivateFn = () => {
  const loginService = inject(LoginService);
  const router = inject(Router);

  if (loginService.isLoggedIn()) {
    return true;
  }
  return router.createUrlTree(['']);
};
```

Ein HTTP-Interceptor (sessionExpiryInterceptor) ergänzt bei jedem ausgehenden Request automatisch die Option withCredentials: true, sodass der Browser das HttpOnly-Cookie bei jeder Anfrage an das Backend mitsendet. So muss das Token nicht manuell im Frontend verwaltet werden. Den Mechanismus zeigt der Quellcode 4.

Listing 4: HTTP Interceptor für Cookie-Übermittlung (session-expiry.interceptor.ts)

```
export const sessionExpiryInterceptor: HttpInterceptorFn = (req, next) => {
    const sessionExpiryService = inject(SessionExpiryService);
    const isLoginRequest = req.url.includes('/auth/login');
    const authReq = req.clone({ withCredentials: true });

    return next(authReq).pipe(catchError((error: HttpErrorResponse) => {
        const isAuthExpired = error.status === 401 || error.status === 403;

        if (!isLoginRequest && isAuthExpired) {
            sessionExpiryService.handleExpiredSession();
        }
            return throwError(() => error);
        })
    );
};
```

Die Komponente verwendet den `LoginService` und ruft dessen `login()`-Methode auf, sobald das Formular eingereicht wird. Er sendet einen `POST-Request` an den Backend-Endpunkt `/auth/login`, der die Anmeldedaten (Nutzername und Passwort) im JSON-Format umfasst. Zunächst prüft das Backend, ob ein Benutzer mit dem angegebenen Nutzernamen in der Datenbank existiert. Ist dies der Fall, vergleicht bcrypt das eingegebene Passwort mit dem gespeicherten Hash. Wenn die Werte übereinstimmen, generiert das Backend ein JWT (JSON Web Token) mit einer Gültigkeit von zwei Stunden, das die Benutzer-ID und den Admin-Status enthält. Das Token wird als HttpOnly-Cookie in der HTTP-Response gesetzt, während die Benutzerdaten als JSON-Antwort zurückgegeben werden.

Im Frontend setzt der LoginService nach erfolgreicher Anmeldung den internen Authentifizierungsstatus und speichert das User-Objekt im Arbeitsspeicher der Anwendung. Das vom Backend gesetzte HttpOnly-Cookie wird automatisch bei allen nachfolgenden Anfragen vom

Browser mitgesendet, ohne dass das Frontend das Token direkt verwalten muss. Danach wird der Nutzer über den Angular-Router zum Dashboard geleitet. Wenn die Anmeldung fehlschlägt, zum Beispiel weil das Passwort nicht stimmt, erscheint die Nachricht „Ungültige Anmeldedaten“ unter dem Formular.

Beim Initialisieren der Anwendung wird über einen `APP_INITIALIZER` die Methode checkExistingAuth() des LoginService aufgerufen. Diese sendet einen GET-Request an den Endpunkt `/auth/me`. Der HttpOnly-Cookie wird dabei automatisch vom Browser mitgesendet. Ist die serverseitige Session noch gültig, stellt das Backend die Benutzerdaten bereit und der Authentifizierungsstatus wird wiederhergestellt, ohne dass der Benutzer seine Zugangsdaten erneut eingeben muss.

Das Sequenzdiagramm in Abbildung 31 zeigt den gesamten Prozess einer Anmeldung.

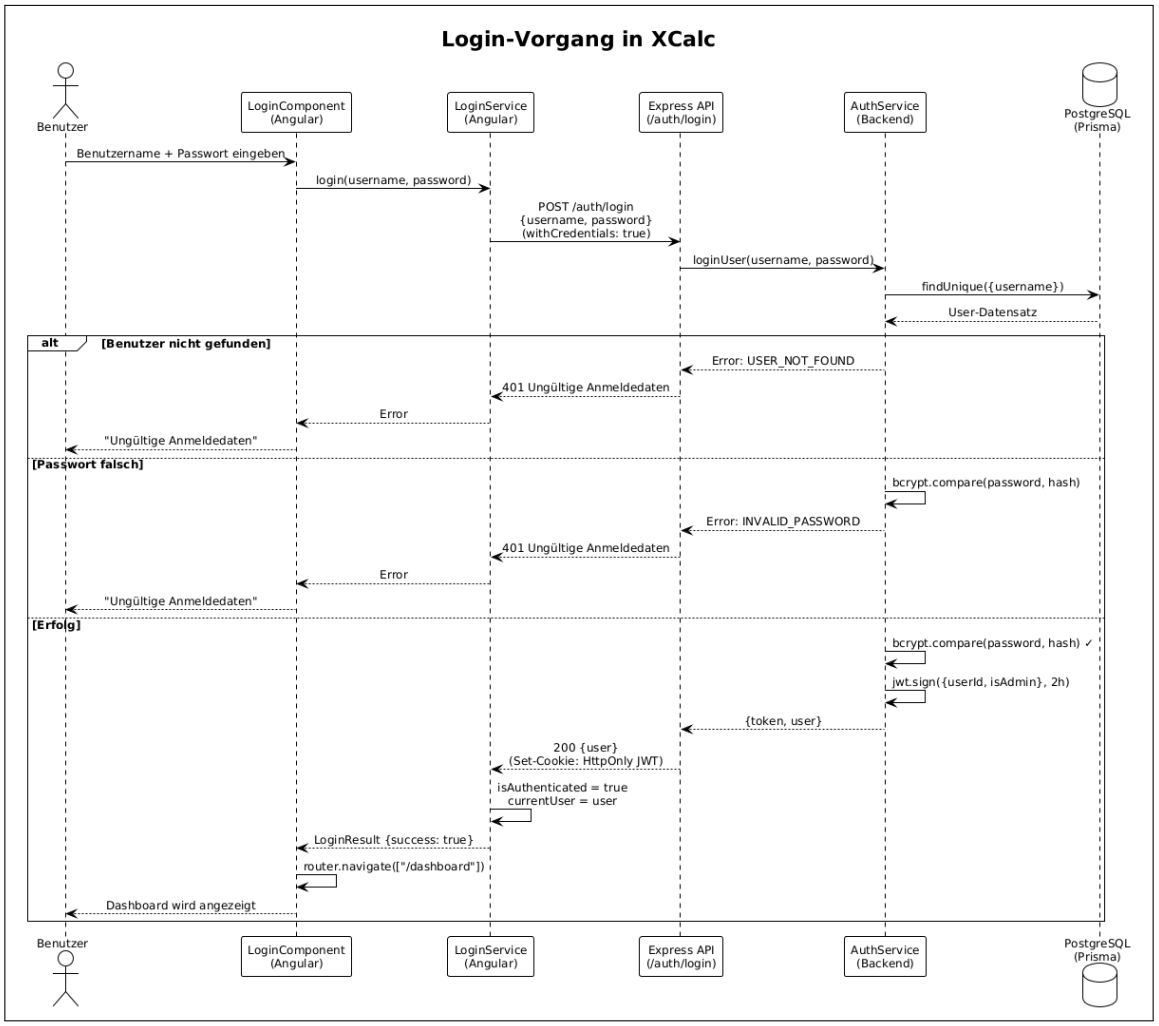


Abbildung 9: Sequenzdiagramm Login-Vorgang

Quelle: eigene Darstellung, gerendert mit PlantUML [1]

4.4.2 Logout und Session-Management

Der Authentifizierungszyklus endet mit dem regulären Abmeldevorgang und dem automatisierten Session-Management. Die Mechansimen sorgen dafür, dass unbefugte Zugriffe nach Beendigung der aktiven Nutzung zuverlässig verhindert werden. Dabei wurden entsprechende Sicherheitsstandards beachtet um potenzielle Schwachstellen, wie Cross-Site Scripting (XSS) zu verhindern.

Token-Management Das JSON Web Token wird nicht im lokalen Webbrowser abgelegt, sondern das Backend verfolgt beim Senden und Speichern das `HttpOnly`-Cookie-Prinzip. Wenn sich der Benutzer erfolgreich angemeldet hat, wird das Token mit dem Attribut `sameSite: ßtrict"` an den Client übergeben [19]. Somit wird sichergestellt, dass clientseitige Skripte keinen Zugriff auf das Token erhalten und keine XSS-Angriffe möglich sind [20].

Logout Der Benutzer kann über die Seiten Startseite, Basisdatenverwaltung und Nutzerverwaltung über einen Logout-Button den Backend-Endpunkt `POST /auth/logout` aufrufen, um sich abzumelden. Hier wird das Authentifizierungs-Cookie überschrieben und serverseitig über die Methode `res.clearCookie` invalidiert. Clientseitig wird dann noch über den `LoginService` der Zustand durch `isAuthenticated = false` bereinigt und der Anwender wird über das Angular-Routing zurück zur Login-Maske navigiert.

Abbildung 10: Screenshot Abmelde-Button im Footer

Automatisierter Session-Ablauf (Expiration) Um die Gültigkeitsdauer einer aktiven Sitzung zu beschränken, wird serverseitig ein sogenannter „Sliding Expiry“-Mechanismus verwendet [21]. Bei jeder gültigen Anfrage an eine gesicherte Adresse verifiziert die `authenticate`-Middleware, das bereits bestehende Token und dem Client wird sofort ein erneuertes Token mit einer Laufzeit von 15 Minuten aus. Hiermit wird so gut es geht das automatische Abmelden nach 15-minütiger Inaktivität realisiert.

Falls diese Inaktivität erreicht wird, verweigert das Backend weitere Anfragen strikt mit den HTTP-Statuscodes `401 Unauthorized` oder `403 Forbidden`. Um dies elegant und benutzerfreundlich im Frontend umzusetzen, wurde ein spezieller HTTP-Interceptor (`sessionExpiryInterceptor`) implementiert. Dieser fängt fehlerhafte Antworten ab und löst den `SessionExpiryService` aus. Hier wird durch ein Status-Flag (`isHandlingExpiry`) der Ablauf gesteuert und verhindert, dass parallele asynchrone Anfragen mehrere Warnungen gleichzeitig generieren. Weiters wird der Benutzer über ein modales Dialogfenster (`SessionExpiredDialogComponent`) über den Ablauf seiner Sitzung informiert. Nachdem der Benutzer den Dialog bestätigt, wird die lokale Sitzung bereinigt und zum Login umgeleitet.

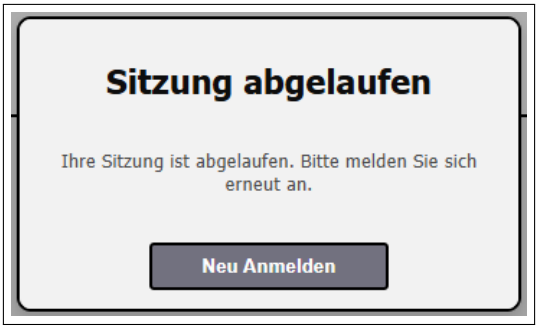


Abbildung 11: Screenshot Dialogfenster zur Bekanntgabe des Session-Ablaufs

Listing 5: Session-Expiry-Interceptor (session-expiry.interceptor.ts)

```
export const sessionExpiryInterceptor: HttpInterceptorFn =
  (req, next) => {
  const sessionExpiryService = inject(SessionExpiryService);
  const isLoginRequest = req.url.includes('/auth/login');

  return next(req).pipe(
    catchError((error: HttpErrorResponse) => {
      const isAuthExpired =
        error.status === 401 || error.status === 403;

      if (!isLoginRequest && isAuthExpired) {
        sessionExpiryService.handleExpiredSession();
      }
      return throwError(() => error);
    })
  );
};
```

4.4.3 Navigation und Rollenunterscheidung

Es wird auf allen Seiten eine Navigationsleiste angezeigt. Diese umfasst neben den Links zu den Seiten (Startseite, Basisdatenverwaltung, Nutzerverwaltung) auch das Logo des Auftraggebers und den Namen der Anwendung. Die Navigationskomponente (`NavigationComponent`) wird als eigenständige Angular-Komponente in die Seitenkomponenten integriert. Befindet sich der Benutzer in einer aktiven Berechnung verschwindet die Navigationsleiste und wird durch eine selbst erstellte Grafik ersetzt. Diese befindet sich auch auf der Login-Seite.

Abbildung 12: Screenshot Navigation XCalc

Abbildung 13: Screenshot Header-Grafik während Berechnung

Die Methode `isAdmin()` im `LoginService` unterscheidet zwischen normalen Nutzern und Administratoren, indem sie den Admin-Status aus dem gespeicherten User-Objekt überprüft. Die Links in der Navigationsleiste sind durch eine `*ngIf="isAdmin()"`-Direktive geschützt. Sie sind nur sichtbar, wenn der Benutzer, der angemeldet ist, ein Administrator ist. Reguläre Benutzer sehen keine dieser Links, sondern lediglich das Logo und den Namen der Anwendung.

4.4.4 Startseite (Dashboard)

Die `DashboardComponent` ist die Startseite der Anwendung. Nach erfolgreichem Login wird sie als zentraler Eintiegspunkt verwendet und dient zur Verwaltung der Projekte. Wenn die Komponente initialisiert wird, werden über den `XCalcService` alle Metadaten der Projekte abgerufen (Name, Kunde, Erstellungsdatum) und in einer Tabellenstruktur angezeigt. Falls keine Einträge vorhanden sind wird in der Tabelle eine Zeile mit dem Text "Keine Projekte gefunden" angezeigt.

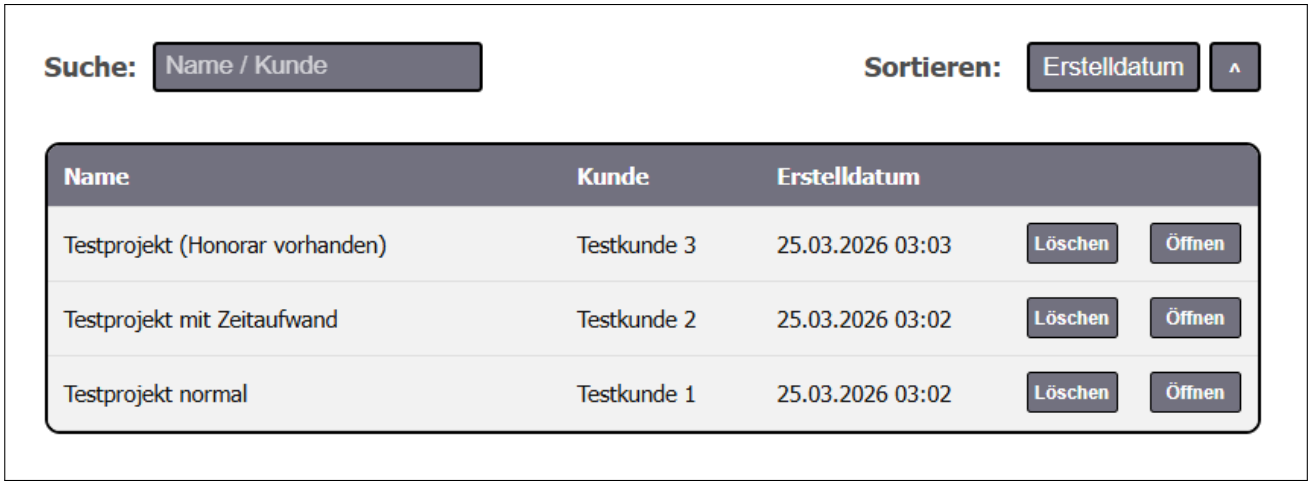


Abbildung 14: Schcreenshot Tabelle der Projekte

Datenverwaltung und Suchfunktionen Da die Anzahl der Datensätze stetig wachsen wird und trotzdem eine hohe Benutzerfreundlichkeit gewährleistet werden muss, werden clientseitige Filter- sowie Sortierfunktionen implementiert. Benutzer können nach den eindeutigen Projektnamen oder dem Kunden (nicht eindeutig) suchen. Hier wird der entsprechende Suchbegriff über das `ngModel`-Binding (`filterTerm`) dynamisch erfasst und in Echtzeit auf das Array angewandt. Bei der Sortierfunktion kann der Benutzer zwischen allen Projekt-Metadaten auswählen und entscheiden ob die Anordnung aufsteigend oder absteigend sein soll. Initial erfolgt die Sortierung aufsteigend nach dem Erstellungsdatum. Diese Funktionen werden direkt im Client ausgeführt, um zusätzliche Anfragen an das Backend zu ersparen und die Reaktionszeit der Benutzeroberfläche zu minimieren.

Ressourcenschonende Projekterstellung und Workflow-Restriktionen

Eine Besonderheit der Anwendung liegt in der Logik zur Erstellung neuer Projekte. Wenn der Benutzer auf den Button ”Neues Projekt erstellen” klickt, öffnet sich ein mehrstufiger modaler Dialog. Im ersten Schritt kann der Benutzer Projektname und Kunde definieren und im zweiten Schritt wird der Berechnungsworkflow ausgewählt. Die Auswahlmöglichkeit ”Honorar nicht vorhanden” startet eine reguläre Berechnung beginnend mit der Baukostenermittlung, während die zweite Auswahlmöglichkeit ”Honorar bereits vorhanden” direkt zur Honorarverteilung springt, da die Ermittlung der Nettoherstellungskosten und dem Honorar nicht mehr nötig ist. Hier ist die Seite dann entsprechend modifiziert, damit der Benutzer das Honorar in einen eigenen Eingabefeld angeben kann.

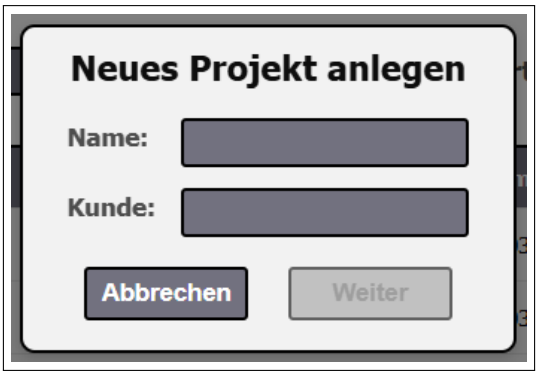


Abbildung 15: Screenshot Eingabefenster Metadaten

Falls der Nutzer diesen verkürzten Workflow wählt, so wird bei der Navigation ein Status (`state` mit dieser Information (`feeAlreadyExists`) vom Angular-Router übergeben. Damit wird verhindert, dass der Benutzer beim erneuten Öffnen des Projekt nicht weiter als bis zur Honorarverteilung zurücknavigieren kann. Der ”Zurück”-Button wird dabei über die Angular-Direktive `*ngIf="!feeAlreadyExists"` ausgeblendet.

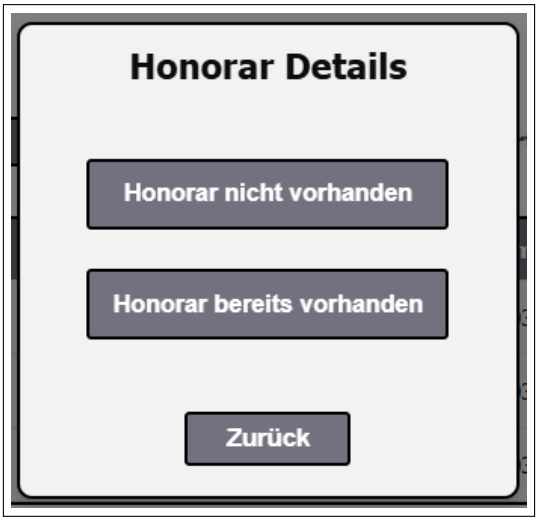


Abbildung 16: Screenshot Auswahl des Berechnungsworkflows

Da der Benutzer zum Beispiel die Berechnung über einen Button ”Projekt verwerfen” jederzeit abbrechen kann, muss vor inkonsistente Datenbestände geschützt werden. Das Projekt wird nicht nach dem Eingeben der Metadaten und der Auswahl des Berechnungsworkflows persistiert, sondern die Metadaten werden zuerst über die Methode `setProjectMeta` temporär im lokalen `CalculationStoreService` gespeichert. Der Angular-Router navigiert anschließend zum Beginn der ausgewählten Berechnung und es wird eine virtuelle Platzhalter-ID `-1` gesetzt. Erst durch das Klicken auf den Button ”Zurück zur Startseite” in der Zusammenfassung wird das Projekt tatsächlich im Backend angelegt und in die PostgreSQL-Datenbank über den Endpunkt `POST /projects/id/complete` gespeichert.

Projektverwaltung und Löschvorgang

Für jeden einzelnen Datensatz in der Tabelle stehen dem Benutzer zwei Aktionen zur Verfügung. Wenn er den Button ”Öffnen” auswählt, navigiert der Router direkt zur entsprechenden Zusammenfassungseite (`/summary/projectId`) und im Hintergrund werden sämtliche Daten des Projekts abgerufen und angezeigt. Hier kann er auch zurücknavigieren um die Berechnungen zu ändern und im Anschluss über den Button ”Zurück zur Startseite” wieder zu persistieren. Falls er den Button ”Löschen” auswählt wird er über die `ConfirmationDialogComponent` zur Bestätigung aufgerufen. Wenn er zustimmt wird der Löschbefehl über den `XCalcService` and die REST-API gesendet. Durch Cascade Delete wird in der Datenbank garantiert, dass nicht nur die Metadaten sondern auch sämtliche verknüpfte Daten (Baukosten, Honorarberechnung, Honorarverteilung, usw.) konsistent gelöscht werden. Im Anschluss wird die Tabelle im Frontend automatisch aktualisiert.

4.4.5 Nutzerverwaltung

In der `UserManagementComponent` implementiert die Nutzerverwaltung. Dies ist ein geschützter Bereich der Anwendung, der ausschließlich für Administratoren zugänglich ist. Wenn die Komponente geladen wird, wird über den `XCalcService` ein HTTP-GET-Request an den Endpunkt `/users`. Allerdings wird die Anfrage nur verarbeitet, wenn die `requireAdmin`-Middleware den Admin-Status des Benutzers anhand des JWT-Payload erfolgreich verifiziert. Als Antwort kommt eine Liste aller Systembenutzer, bestehenden aus `userId`, `username` und dem `isAdmin`-Flag.

Die Benutzer werden in der Benutzeroberfläche in einer Listenansicht präsentiert. Für jeden Datensatz stehen zwei Aktionen zur Verfügung. Durch den Button ”Nutzername ändern” kann der Nutzername geändert werden und über den Button ”Passwort ändern” kann Der Benutzer das Passwort anpassen. Die Aktionen werden nicht direkt in der Listenansicht durchgeführt, sondern es öffnen sich jeweils spezifische modale Dialogfenster, wo der Anwender die entsprechenden Daten ändern kann. Die Fenster basieren auf das `MatDialog`-Modul der Angular Material-Bibliothek.

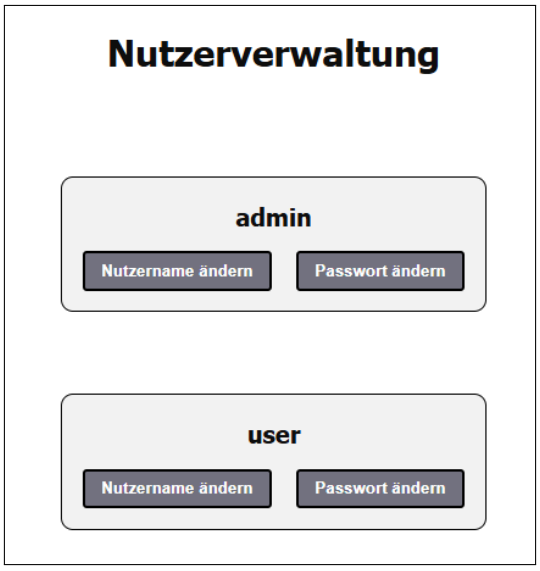


Abbildung 17: Screenshot Listenansicht der Nutzerverwaltung

Nutzername ändern (`UsernameComponent`)

Durch das Klicken auf den Button ”Nutzername ändern” öffnet sich ein Dialog, um den neuen Namen einzugeben. Die Frontend-Komponente überprüft dann, ob das Feld einen gültigen, nicht-leeren String enthält. Ist dies der Fall und der Vorgang wird durch den Dialog `UsernameConfirmationComponent` bestätigt, werden die Daten über den `XCalcService` per `POST /users/change-username` an das Backend übermittelt.

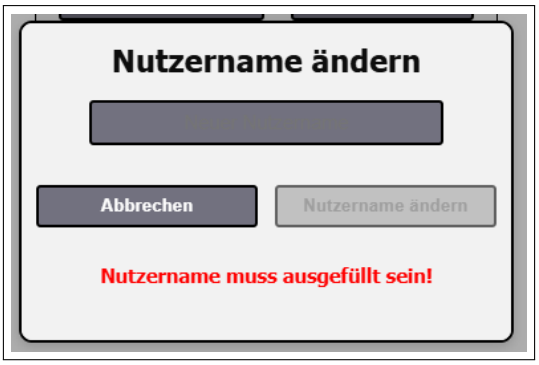


Abbildung 18: Screenshot Eingabefenster Nutzername

Im Backend (`userService.ts`) wird als erstes geprüft ob der neue Nutzername bereits in der PostgreSQL-Datenbank existiert. Ist dies der Fall, wird die Transaktion abgebrochen und ein HTTP-Statuscode `400 Bad Request` mit dem Prisma-Fehlercode `P2002` (Unique Constraint Violation) [9] an das Frontend zurückgesendet. Falls die Änderung erfolgreich war wird der Datensatz aktualisiert und der Nutzername wird dynamisch in der lokalen `users`-Liste angepasst. Es wird ein vollständiger Seiten-Reload verhindert und der Benutzer wird über die `FeedbackDialogComponent` über den Erfolg informiert.

Passwort ändern (`PasswordComponent`)

Die Änderung eines Passworts erfolgt nach dem gleichen strukturellen Prinzip wie beim Ändern des Nutzernamens. Der Administrator wird zur doppelten Eingabe des neuen Passworts (`newPassword` und `confirmPassword`) aufgefordert. Die Validierung erfolgt clientseitig und stellt sicher, dass beide Eingaben exakt übereinstimmen und nicht leer sind.

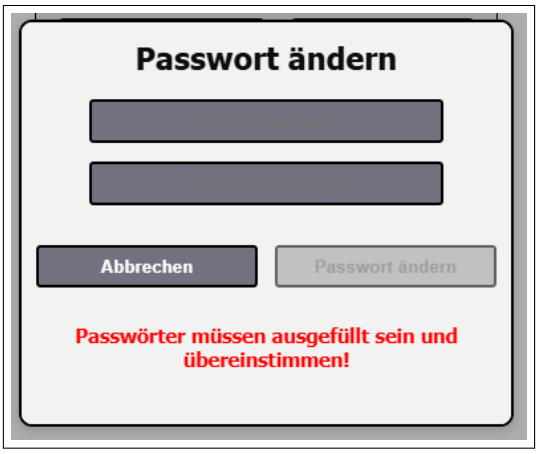


Abbildung 19: Screenshot Eingabefenster Passwort

Nach der Zustimmung im Bestätigungsdialog wird der API-Endpunkt `POST /users/reset-password` aufgerufen. Daraufhin wird im Backend (`userService` mittels der `bcrypt`-Bibliothek [22] und festgelegten Salt Rounds kryptografisch gehasht. Der alte Hash-Wert wird mit dem neu generierten Wert überschrieben und das Klartextpasswort wird nie im Backend persistiert. Auch hier wird der Administrator über ein visuelles Feedback-Fenster über dem Vorgang informiert.

4.4.6 Basisdatenverwaltung

In der Basisdatenverwaltung (`BaseDataManagementComponent`) kann der Administrator, die systemweiten Berechnungsparameter welche für die Honorarberechnung essenziell sind, anpassen. Es sind vier Zahlwerte:

- `k0` – Ein Grundwert für die Honorarberechnung
- `h(op)` – Der Honorarordnungsparameter
- `f1` – Berechnungsfaktor 1
- `f2` – Berechnungsfaktor 2

Beim Laden der Seite ruft die Komponente über den `XCalcService` die aktuellste Version der Basisdaten ab, indem eine Anfrage an den Backend-Endpunkt (`GET /base-data/latest`) gesendet wird und die aktuellen Werte in den Eingabefeldern anzeigt.

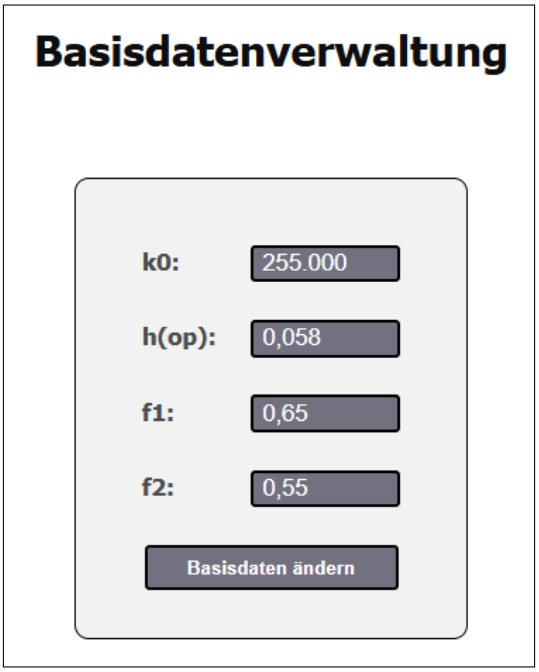


Abbildung 20: Screenshot Eingabefeld Basisdatenverwaltung

Bevor die finale Speicherung erfolgt, erscheint ein Bestätigungsdialog (`ConfirmationComponent`), der die eingegebenen Werte übersichtlich darstellt und den Administrator fragt, ob die Änderung wirklich durchgeführt werden soll. Die Anfrage wird erst nach der Bestätigung an das Backend gesendet (`POST /base-data`).

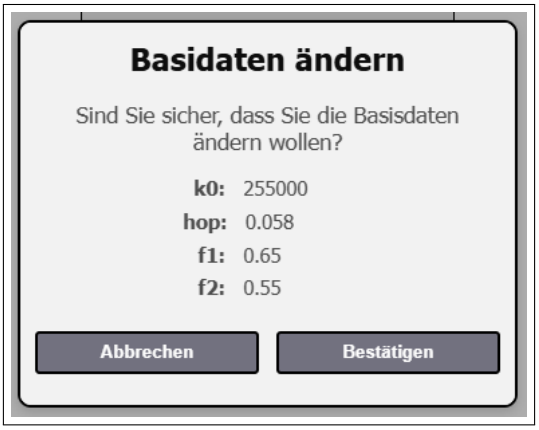


Abbildung 21: Screenshot Bestätigungsfenster Basisdaten

Es wird bewusst keine bestehende Version überschrieben, sondern stattdessen wird eine neue `BaseDataVersion` erstellt. Honorarberechnungen, die mit einer älteren Version verknüpft sind, behalten über dem Fremdschlüssel ihre Verknüpfung und werden durch die Änderung nicht beeinflusst. Dieses Prinzip der Versionierung sorgt dafür, dass alle abgeschlossenen Berechnungen nachvollziehbar bleiben, selbst wenn die Parameter für zukünftige Projekte angepasst werden.

4.5 Berechnungen und PDF-Export

Die Durchführung der Honorarkalkulation und deren Export in ein CI-konformes PDF-Dokument wurden strikt im Frontend realisiert. Dadurch verzögert sich nicht die Rückmeldung an den Benutzer und die Serverlast wird minimiert. Damit der Benutzer nicht während des Berechnungsvorgangs navigieren kann, wird die Navigation lediglich durch eine CI-konforme Grafik ersetzt, und die Fußzeile mit dem Abmelde-Button wird komplett ausgeblendet. Mit Ausnahme der anfänglichen Baukostenermittlung verfügt jede Seite über einen ”Zurück”-Button zur Korrektur von Eingaben. Unterhalb der eigentlichen Kalkulation befinden sich die Buttons ”Übermitteln” zum lokalen Zwischenspeichern und Fortfahren sowie ”Projekt verwerfen” zum vollständigen Abbruch des Berechnungsvorgangs.

4.5.1 Baukostenermittlung

Nachdem der Benutzer über das Dashboard erfolgreich ein Projekt erstellt hat und die Option ”Honorar nicht vorhanden” ausgewählt hat, wird er direkt in die Baukostenermittlung geleitet, um mit dem Berechnungsvorgang zu beginnen. In diesem Schritt wird eine Kostenschätzung des Bauvorhabens durchgeführt, um die Nettoherstellungskosten zu ermitteln, welche die Basis für die Honorarberechnung sind. Durch die HTML5-API `window.history.pushState` und einen `popStateHandler` wird das Zurücknavigieren über die Browser-Historie deaktiviert.

Dateneingabe und Visualisierung

Die Benutzeroberfläche besteht hauptsächlich aus einer interaktiven Tabelle, die in vier Spalten untergliedert ist: Bezeichnung, Fläche (m^2), Preis/m^2 und Baukosten netto. In der Spalte der Bezeichnungen sind acht spezifische Bauwerkskategorien, wie etwa ”Nettonutzfläche Büroräume”, ”Nettonutzflächen Nutzungsräume” oder ”Unterschossflächen” festgelegt, damit immer die gleichen Bezeichnungen verwendet werden.

Für jede einzelne Kategorie stehen Eingabefelder für die Fläche und den Quadratmeterpreis zur Verfügung. Um die Datenqualität zu sichern, werden bei der Eingabe spezifische Event-Handler (`onFocus`, `onNumberInput`, `onBlur`) ausgelöst. Durch die Methode `onNumberInput` werde ungültige Zeichen bereits während des Tippens herausgefiltert. Es werden auch die Dezimalstellen limitiert und die Werte in ein JavaScript-kompatibles Zahlenformat konvertiert. Beim Verlassen des Feldes sorgt die Methode `formatInputValue` für eine konsistente, deutsche Zahlenformatierung. Dafür wird auch die `Intl.NumberFormat`-API [23] benutzt. Die resultierenden Baukosten der einzelnen Kategorien, sowie die Gesamtsumme in der letzten Zeile werden in der Spalte Baukosten netto in deaktivierten Feldern (`readonly-field` angezeigt, um manuelle Manipulation zu verhindern. In der letzten Zeile sind die Spalten für die Fläche und den Quadratmeterpreis entsprechend leer.

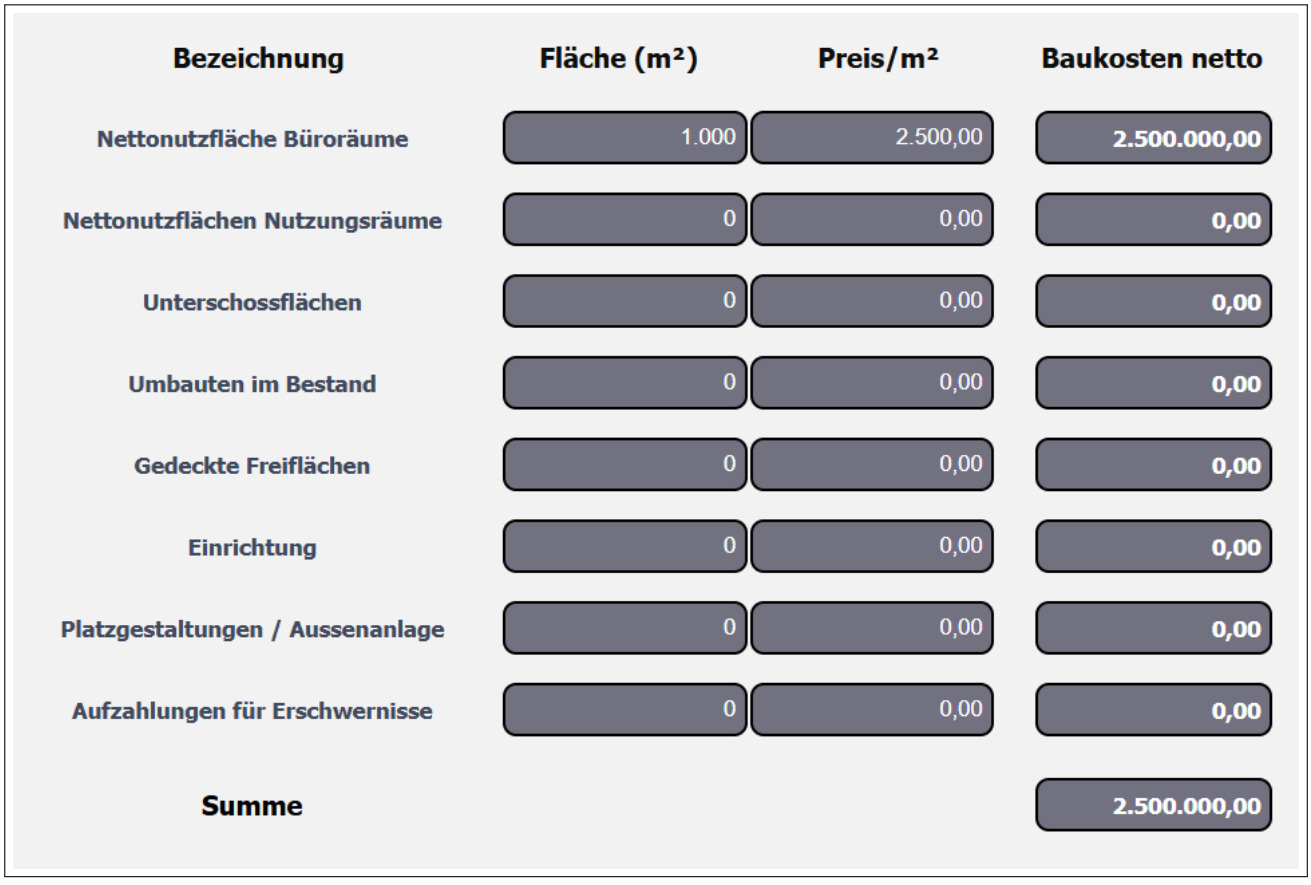


Abbildung 22: Screenshot Eingabetabelle Baukostenermittlung

Echtzeit-Berechnung und temporäre Speicherung

Sobald der Benutzer eine Eingabe in den Flächen- oder Preisfeldern tätigt, triggert das Frontend die Methode `recalculate`. Diese führt die Multiplikation der Einzelwerte durch und summiert die Ergebnisse aller acht Kategorien verzögerungsfrei zur Gesamtsumme (`totalCost`) auf, damit der Benutzer die Auswirkungen der Änderungen sofort erkennen kann.

Wenn diese Phase über den Button "Übermitteln" abgeschlossen wird, werden die erfassten Daten bewusst noch nicht an das Node.js-Backend gesendet. Stattdessen wird ein `ConstructionCost`-Objekt angelegt und über die Methode `setConstructionCost` temporär im lokalen `CalculationStoreService`, welcher auf dem `sessionStorage` [24] basiert, abgelegt. Dies reduziert unnötige Netzwerkzugriffe auf die Datenbank. Anschließend leitet der Angular-Router den Nutzer sofort in die nächste Phase, der Honorarberechnung (`/fee-invoice`) weiter. Die ermittelte Baukostensumme wird über den Navigations-State (`window.history.state`) übergeben.

Projektabbruch und Datenbereinigung

Es besteht jederzeit die Möglichkeit, den Vorgang über "Projekt verwerfen" abzubrechen. Damit versehentliches Klicken des Buttons nicht zu ungewolltem Datenverlust führt, öffnet sich zuvor ein modaler Dialog (`ConfirmationDialogComponent`). Wird der Abbruch bestätigt, wird – sofern das Projekt bereits eine valide Datenbank-ID besitzt – der Endpunkt `DELETE /projects/projectId` aufgerufen. Hiermit wird das gesamte Projekt inklusive aller verbundenen Datensätze aus der Datenbank gelöscht. Abschließend wird der lokale Speicher über `store.clear()` bereinigt und der Anwender zum Dashboard zurückgeleitet.

Abbildung 23: Screenshot Navigationsbuttons in aktiver Berechnung

4.5.2 Honorarberechnung

Nachdem die Baukostenermittlung abgeschlossen wurde, wird der Anwender sofort in die nächste Phase, der Honorarberechnung geleitet. In diesem Schritt erfolgt die Berechnung des Honorars, die auf den zuvor ermittelten Nettoherstellungskosten sowie spezifischen, anpassbaren Parametern basiert.

Dateneingabe und Validierung

Die Benutzeroberfläche gliedert sich in einen Eingabe- und einen Ergebnisbereich. Die Nettoherstellungskosten werden asynchron aus dem lokalen `CalculationStoreService` übernommen und dienen als fixe Ausgangsbasis. Der Benutzer kann den Schwierigkeitsgrad (Klasse 1 bis 10) sowie die beauftragte Leistung in Prozent angeben. Nach dem Feld für die Anzeige des ermittelten Honorars lässt sich ein Nachlass (0–100 Prozent) eingeben. Zu jedem Eingabefeld sind selbst erstellte +/– Buttons vorhanden, um kleine Änderungen schneller durchzuführen.

Wie schon bei der Baukostenermittlung werden fehlerhafte Berechnungen durch spezifische Event-Handler (`onInput` und `onBlur`) in der `FeeInvoiceComponent` ausgeschlossen. So wird beispielsweise verhindert, dass prozentuale Werte den Bereich von 0 bis 100 unter- beziehungsweise überschreiten. Die Resultate werden erfolgt und benutzerfreundlich mit zwei Dezimalstellen über die native `Intl.NumberFormat`-API dargestellt.

Abbildung 24: Screenshot Benutzeroberfläche Honorarberechnung

Architektur der Berechnungslogik

Damit das Prinzip der *Separation of Concerns* [25] implementiert wird, ist die gesamte mathematische Logik in eine eigenständige Utility-Datei (`fee-calculation.util.ts`) ausgelagert. Damit wird die Angular-Komponente schlank gehalten und sie fokussiert sich auf Datenbindung und DOM-Interaktionen.

Die Werte werden bei jeder Parameteränderung in Echtzeit neu berechnet. Der Algorithmus ermittelt hierbei zunächst über eine polynomische Funktion einen Planungsfaktor auf Basis des gewählten Schwierigkeitsgrades. Anschließend wird der Basishonorarsatz unter Berücksichtigung der Basisdaten (`k0`, `hop`, `f1`, `f2`), welche aus den Backend mittels `GET /base-data/latest` geholt werden, kalkuliert [26]. Ein essenzielles Detail der Berechnungslogik ist die integrierte Kostengrenze: Über die Konstante `COST_CAP` wird sichergestellt, dass Nettoherstellungskosten

von über 70.000.000 Euro den prozentualen Basishonorarsatz nicht weiter verfälschen, da die Formel ab hier streng auf diesen Maximalwert limitiert ist.

Listing 6: Honorarberechnungslogik (fee-calculation.util.ts)

```
export const COST_CAP = 70_000_000;

export function calculateFeeValues(
  input: FeeCalculationInput
): FeeCalculationResult {
  const netCost = Math.max(0, toFiniteNumber(input.netCost));
  const difficultyClass = clamp(
    toFiniteNumber(input.difficultyClass, 1), 1, 10
  );
  const performance = clamp(toFiniteNumber(input.performance), 0, 1);
  const discountPercent = clamp(
    toFiniteNumber(input.discountPercent), 0, 100
  );

  const x = 10 * difficultyClass;
  const planungsfaktor =
    -0.00004 * x ** 2 + 0.0151 * x + 0.5574;

  const referenceCost =
    netCost > 0 ? Math.min(netCost, COST_CAP) : COST_CAP;

  const baseHonorarsatz =
    input.hop
    * (input.f1 + input.f2 * Math.pow(input.k0 / referenceCost, 1/3))
    * planungsfaktor;

  const totalHonorarsatz = Number.isFinite(baseHonorarsatz)
    ? baseHonorarsatz : 0;
  const serviceHonorarsatz = totalHonorarsatz * performance;

  const honorarBeforeDiscount =
    Number((netCost * serviceHonorarsatz).toFixed(2));
  const honorar =
    Number((honorarBeforeDiscount * (1 - discountPercent/100)).toFixed(2));

  return {
    planungsfaktor, totalHonorarsatz, serviceHonorarsatz,
    honorarBeforeDiscount, honorar, referenceCost,
  };
}
```

Temporäre Speicherung und Navigation

Wie bei der Baukostenermittlung wird in dieser Phase bewusst auf eine sofortige Persistierung in der PostgreSQL-Datenbank verzichtet, um redundante Netzwerkzugriffe zu vermeiden. Bestätigt der Anwender die Eingaben, wird ein `FeeInvoice`-Objekt erstellt und temporär im lokalen `CalculationStoreService` hinterlegt, welcher den `sessionStorage` des Browsers nutzt.

Nach erfolgreicher Speicherung leitet der Angular-Router den Benutzer zur finalen Honorarverteilung weiter. Durch dieses State-Management-Konzept ist eine Rückkehr zu vorherigen Schritten jederzeit möglich, ohne dass bereits erfasste Daten verloren gehen.

4.5.3 Honorarverteilung

Nach der erfolgreichen Kalkulation des Honorars wird der Anwender in die abschließende Phase der Honorarverteilung geleitet. Ziel hier ist es, das ermittelte Planungshonorar in interne Kostenstellen aufzuteilen, um daraus ein präzises, rollenbasiertes (funktionbasiertes) Stundenbudget für die Abwicklung des Projekts zu ermitteln.

Interne Verteilung, Stundenermittlung und eventuelle Honorareingabe

Die Benutzeroberfläche untergliedert sich in zwei logische Bereiche. Der erste Abschnitt, die interne Verteilung, dient dazu, das Gesamthonorar in einen Büro- und Lohnkostenanteil aufzuteilen. Die initialen Werte dafür sind 40 Prozent für den Büroanteil und 60 Prozent für den Lohnkostenanteil. Diese können allerdings angepasst werden. Auch hier wurden wieder eigene +/- Buttons verwendet

Anschließend wird der resultierende Euro-Wert des Lohnkostenanteil in der nächsten Berechnung (Stunden- und Kostenverteilung) als Grundlage verwendet. Hierbei wird der Betrag auf vier vordefinierte Mitarbeiterfunktionen (Projektleitung, Mitarbeiter/-innen, Administration, Sonstige) mit jeweils 25 Prozent und deren Stundensatz verteilt. Neben den Eingabefeldern mit selbst erstellten Buttons befinden sich die Ergebnisse, welche automatisch berechnet werden. Der Benutzer kann bei Bedarf weitere Funktionen mittels eines Buttons hinzufügen. Hierbei öffnet sich ein modaler Dialog, in dem er Bezeichnung, Prozentanteil und Stundensatz angeben kann.

Abbildung 25: Screenshot Interne Verteilung, Stunden- und Kostenverteilung

Abbildung 26: Screenshot Eingabefenster neue Funktion hinzuzufügen

Falls der verkürzte Berechnungsvorgang mittels ”Honorar bereits vorhanden” ausgewählt wurde, erscheint vor der Internen Verteilung und der Stunden- und Kostenverteilung ein separater Bereich, wo der Benutzer das Honorar eingeben kann. Dies ist nötig, da der Benutzer in Spezialfällen das Honorar direkt vom Kunden bekommt und nicht selbst ermitteln muss.

Berechnungslogik und Entwurfs-Status (Draft-System)

Die Berechnung der Stunden erfolgt in Echtzeit innerhalb der Angular-Komponente. Sobald der Prozentanteil oder der Stundensatz einer Funktion bearbeitet wird, berechnet die Methode `updateAll()` das Stundenbudget nach der Vorgabe: *(Lohnkostenanteil * Rollenanteil) / Stundensatz*.

Für die Eingabe wird ein Entwurfs-Status-System (Draft-System) für die interaktiven Tabellenzellen implementiert. Es wird verhindert, dass die strikte deutsche Zahlenformatierung (Zifferngruppierung und Kommasetzung) die Benutzereingabe stört. Hierbei werden Rohdaten während der aktiven Eingabe temporär in isolierte Map-Datenstrukturen (`tableRateDrafts` und `distributionPercentDrafts`) zwischengespeichert. Erst beim Verlassen des Eingabefeldes werden über das `onBlur`-Event spezifische Hilfsmethoden wie `sanitizeDecimalInput` ausgeführt. Diese bereinigen die Zeichenkette, konvertieren sie in ein numerisches Format und wenden den finalen Wert an, woraufhin die abhängigen Werte fehlerfrei berechnet werden können.

Spezifische Geschäftslogik für Kleinstprojekte

Bei der Honorarverteilung gibt es einen Spezialfall, wenn die Nettoherstellungskosten weniger als 50.000 Euro betragen. Kommt dies zustande gilt das Projekt als Kleinstprojekt. Die Komponente prüft das über die Getter-Methode `istZeitaufwand`. Ist dies der Fall greift eine spezifische Geschäftslogik ein, welche das HTML-Template dynamisch modifiziert. Anstelle von den Stundenbudgets, welche in diesem Größenbereich in der Praxis kaum aussagekröftig sind, wird ”Zeitaufwand” als Platzhaltertext im Ergebnis der Stundenverteilung geschrieben.

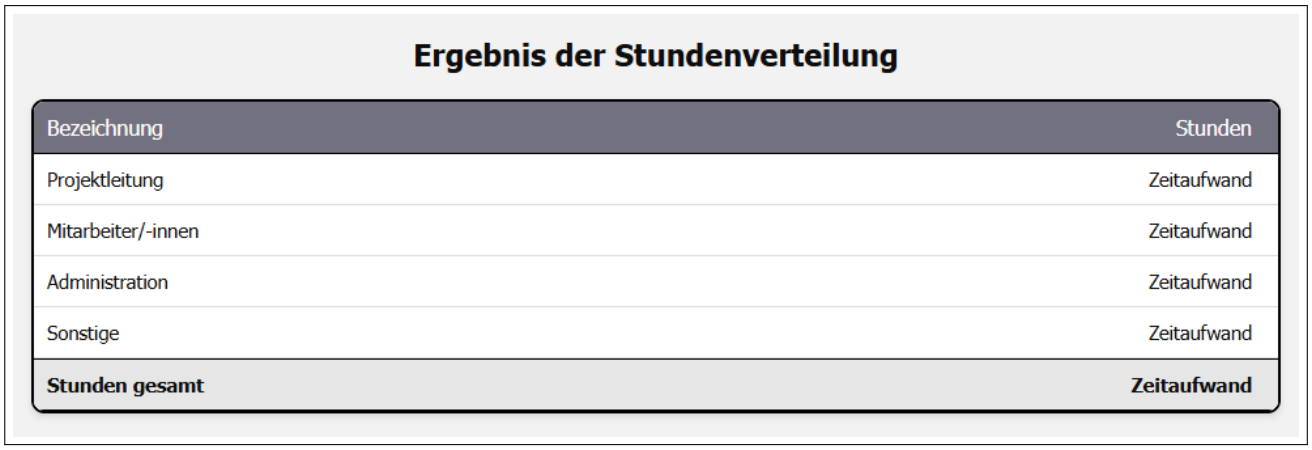


Abbildung 27: Screenshot Ergebnisfeld bei Zeitaufwand

Validierung und Navigation

Um die Konsistenz der Daten zu wahren, gib es in der Komponente strenge Validierungsregeln. Um den Berechnungsvorgang abzuschließen und um zur finalen Zusammenfassung zu gelangen, muss die Kondition `isReadyForSummary` erfüllt sein. Wenn die Summe aller Prozentanteile exakt 100 Prozent beträgt (`isPercentValid`) und für jeden Datensatz ein Stundensatz von über 0 Euro angegeben ist (`isCostValid`) ist die Bedingung erfüllt. Werden diese Kriterien nicht berücksichtigt, rendert das System entsprechende Fehlermeldungen und blockiert den Button ”Zur Zusammenfassung”.

Ist die Validierung erfolgreich und der Anwender bestätigt den Vorgang, werden die strukturierten Funktionsdaten an den `CalculationStoreService` übermittelt und im `sessionStorage` gespeichert. Falls der Benutzer den verkürzten Berechnungsablauf (”Honorar bereits vorhanden”) gewählt hat, wird in der Methode `confirmAndNext()` eine Ausnahmelogik ausgelöst. Das System generiert Dummy-Daten für die übersprungenen Phasen der Baukosten- und Honorarberechnung und injiziert diese in den Store. Somit wird garantiert, dass die nachfolgende Zusammenfassungsseite stets ein konsistentes `CompleteProjectData`-Objekt verarbeiten kann, bevor das Angular-Routing den finalen Schritt einleitet.

4.5.4 Zusammenfassung

Nachdem die Baukostenermittlung, die Honorarberechnung und die Honorarverteilung erfolgreich abgeschlossen wurde, gelangt der Nutzer zur Zusammenfassung. Hier werden alle Eingaben und kalkulierten Ergebnisse noch einmal übersichtlich dargestellt, bevor die Daten persistent im Backend abgespeichert oder als Dokument exportiert wird.

Zusammenführung und Visualisierung der Daten

Beim Initialisieren abonniert die Angular-Komponente die Datenströme (Observables) des `CalculationStoreService`, um die vollständige Berechnungsübersicht zu generieren. Die Benutzeroberfläche gliedert sich in die drei zuvor durchlaufenen Berechnungsphasen, welche in verschiedenen Tabellen angezeigt werden. Davor befinden sich noch im oberen Bereich zwei editierbare Eingabefelder für den Projektnamen und den Kunden. Diese sind mittels Two-Way-Data-Binding (`[(ngModel)]`) verknüpft, sodass Metadaten vor dem Abschluss nochmals angepasst werden können.

Abbildung 28: Screenshot Beginn der Zusammenfassung

Die tabellarischen Aufstellungen der Baukosten, Honorarparameter und der Kostenverteilung werden durch die `Intl.NumberFormat`-API dynamisch in ein spezifisches deutsches Format mit zwei Dezimalstellen konvertiert.

Anpassung bei Kleinstprojekten

Für den Fall von Kleinstprojekten passt sich die Anzeige der Zusammenfassung entsprechend an Über die Eigenschaft `isZeitaufwand` prüft die Komponente, ob die ermittelten Nettoherstellungskosten unter den Schwellenwert von 50.000 Euro liegen. Falls dieser Fall eintritt, passt sich das HTML-Template automatisch über Angular-Directives wie `*ngIf` an. Die Berechnungen selbst werden hier alle normal angezeigt und bei dem zentralen Ergebnis, also der Stundenverteilung, wird der Platzhaltertext ”Zeitaufwand” angezeigt, damit die Darstellung der Geschäftslogik entspricht. Dies dient dazu, dass der Benutzer nachvollziehen kann, wie es zum Zeitaufwand kam und falls einzelne Informationen heruasgefiltert werden sollen.

Architektur zur Persistierung

Am Ende der Zusammenfassung stehen dem Benutzer drei Aktionen per Button-Klick zur Verfügung:

- Zurück zur Startseite
- PDF Export
- Projekt verwerfen

Falls sich der Benutzer für den Abbruch ("Projekt verwerfen") entscheidet, kontrolliert die Applikation zunächst den Status des Projekts. Handelt es sich um eine Neuanlage, wird lediglich nur der lokale Zwischenspeicher (`CalculationStoreService`) geleert, da zu diesem Zeitpunkt noch keine Datenbankeinträge existieren. Wurde das Projekt hingegen von der Startseite aus geöffnet, wird über den `XCalcService` ein HTTP-DELETE-Request an die REST-API (`DELETE /projects/{projectId}`) gesendet, um den Datensatz mitsamt aller Relationen kaskadierend aus der PostgreSQL-Datenbank zu entfernen.

Entscheidet sich der Benutzer allerdings für den regulären Prozess und klick auf den Button "Zurück zur Startseite", bündelt die Applikation alle temporären Daten zu einem umfassenden `CompleteProjectData`-Objekt. Dieses zusammengeführte Objekt wird über den Endpunkt `POST /projects/{projectId}/complete` an das Backend übermittelt, welches die Speicherung und Verknüpfung der Daten in der Datenbank über eine transaktionssichere Operation verwaltet.

Listing 7: Speicherung der Projektdaten mittels Prisma-Transaktion (projects.ts)

```
// 1. Projekt erstellen oder aktualisieren (Metadaten)
const result = await prisma.$transaction(async (tx) => {
  // ... (Projekt anlegen/updaten ausgeblendet) ...

  // 2. Bestehende Detail-Daten l schen , um Duplikate zu vermeiden
  await tx.constructionCostItem.deleteMany({
    where: { constructionCost: { projectId: projectId } }
  });
  await tx.feeDistributionRole.deleteMany({
    where: { feeDistribution: { projectId: projectId } }
  });

  // 3. Neue Detail-Daten anlegen (Baukosten, Honorar, Verteilung)
  const newFeeInvoice = await tx.feeInvoice.create({
    data: {
      projectId: projectId,
      difficultyClass: parsedDifficultyClass,
      performance: performanceValue,
      discountPercent: discountValue
    }
  });

  const newFeeDistribution = await tx.feeDistribution.create({
    data: {
      projectId: projectId,
      totalFee: gesamthonorar,
      feeShareSum: lohnkostenanteil,
      // ... Verteilung der einzelnen Rollen
      roles: {
        create: safeDistribution.map((role: any) => ({
          roleName: role.role,
          sharePercent: Math.min(999.99, Number(role.percent) || 0),
          hours: Number(role.hours) || 0,
          hourlyRate: Number(role.rate) || 0,
        }))
      }
    }
  });

  return { projectId, newFeeInvoice, newFeeDistribution };
});
```

4.5.5 PDF-Export

Die Anwendung ermöglicht es, in der Zusammenfassung die finalen Projektdaten als CI-konformes PDF-Dokument zu exportieren, um sie, falls gewollt, änderungssicher zu archivieren oder an den Kunden weiterzugeben. Das Dokument wird vollkommen clientseitig im Webbrowser erstellt, und zwar mit Hilfe der Open-Source-Bibliothek `jsPDF`.

Separation of Concerns

Wie bei der Honorarberechnung wurde auch beim PDF-Export das Prinzip der Trennung von Zuständigkeiten (*Separation of Concerns*) angewandt. Die komplexe Logik der Dokumentengenerierung wird in eine separate Utility-Datei ausgelagert (`pdf-export.util.ts`), anstatt die `SummaryComponent` mit aufwendigen, exzessiven Layout- und Zeichenanweisungen zu überladen. Die Hilfskomponente erhält die komplette SummaryComponent, damit es nicht nur auf die Daten sondern auch auf die Funktionen der Komponente zugreifen kann.

Listing 8: Hauptsteuerung der PDF-Generierung (pdf-export.util.ts)

```
export async function exportToPDF(summary: SummaryComponent) {
  try {
    const pdf = new jsPDF('p', 'mm', 'a4');
    let y = 10;

    // === Header ===
    y = await addPDFHeader(pdf, y, summary);
    y += 10;

    // === 1) Baukostenermittlung ===
    if (!summary.isDirectFee) {
      y = addConstructionCostSection(pdf, y, summary);
    }

    // === 2) Honorarberechnung ===
    if (!summary.isDirectFee && summary.feeInvoice) {
      y = addFeeSection(pdf, y, summary);
    }

    // === 3) Kostenverteilung + Stundenverteilung ===
    if (summary.distribution && summary.distribution.length > 0) {
      y = await addDistributionSection(pdf, y, summary);
    }

    pdf.save(`Zusammenfassung - ${summary.projectName}.pdf`);
  } catch (error) {
    console.error('PDF Export Fehler:', error);
    alert('Fehler beim Erstellen der PDF-Datei.');
  }
}
```

Aufbau und Visualisierung

Das Corporate Design der X Architekten wird im PDF-Aufbau strikt eingehalten. Zu Beginn des Generierungsprozess wird ein asynchroner Ladevorgang über die Utility-Funktion `loadImage` ausgeführt, welche das grafische Firmenlogo in eine Base64-Data-URL konvertiert. Dieses Logo wird anschließend gemeinsam mit den Metadaten des Projekts (Projekt- und Kundenname) sowie der Applikationsbezeichnung (XCalc) im Header positioniert.

Listing 9: Asynchrone Bildkonvertierung für den PDF-Export (pdf-export.util.ts)

```
export function loadImage(src: string): Promise<string> {
  return new Promise((resolve, reject) => {
    const img = new Image();
    img.crossOrigin = 'anonymous';
    img.onload = () => {
      const canvas = document.createElement('canvas');
      const ctx = canvas.getContext('2d');
      canvas.width = img.width;
      canvas.height = img.height;
      ctx!.drawImage(img, 0, 0);
      resolve(canvas.toDataURL('image/png'));
    };
    img.onerror = reject;
    img.src = src;
  });
}
```

Daraufhin werden die drei Hauptabschnitte sequentiell gerendert:

- Baukostenermittlung
- Honorarberechnung
- Kostenverteilung

Da die eingesetzte Bibliothek `jsPDF` auf einer Low-Level-Canvas-API basiert und bewusst auf fehleranfällige Plugins zur Tabellengenerierung verzichtet wurde, erfolgt die Platzierung von Texten, Linien und Tabellen durch präzise Koordinatenberechnungen (X- und Y-Offsets) sowie definierten Zeilenhöhen (`rowHeight`).

Es wurden eigene generische Zeichenmethoden für die Tabellenstrukturen, wie `addTwoColTableToPDF` und `addThreeColTableToPDF` implementiert. Hier werden auf Basis der verfügbaren Seitenbreite (`pageWidht` die Spaltenbreiten dynamisch berechnet. Dabei werden feste Verhältnisse – beispielsweise 55 % zu 45 % in der zweispaltigen Ansicht verwendet. Um das moderne User Interface der Webapplikation zu replizieren, nutzen diese Hilfsmethoden Funktionen wie `pdf.setFillColor()`, um die entsprechenden Farbunterschiede der Tabellen einzusetzen und `pdf.roundedRect()` zur Zeichnung eines abgerundeten äußeren Rahmens um die gesamten Tabellen. Essenzielle Ergebnisse, wie das finale Honorar oder die Summe der Baukosten, werden über den Parameter `highlightRowIdx` identifiziert und durch eine farblich abgehobene Hintergrundfüllung und Fettdruck gezielt visuell hervorgehoben.

Listing 10: Generierung CI-konformer Tabellenstrukturen (pdf-export.util.ts)

```
// Hintergrundfarbe des Tabellenkopfes zeichnen (Dunkelgrau)
pdf.setFillColor(114, 113, 127);
pdf.rect(startX + 0.35, headerTopY + 0.35, tableWidth - 0.7, rowHeight - 0.35, 'F');

// Hervorgehobene Summenzeile zeichnen (Hellgrau)
if (hasHighlight) {
  pdf.setFillColor(224, 224, 224);
  pdf.rect(startX + 0.2, highlightTopY, tableWidth - 0.4, highlightRowHeight - 0.2, 'F');
}

/* ... (Text-Rendering und Trennlinien ausgeblendet) ... */

// Abschliessender Rahmen mit abgerundeten Ecken (Radius 2.5)
pdf.setLineWidth(0.8);
pdf.setDrawColor(0, 0, 0);
pdf.roundedRect(startX, startY, tableWidth, tableHeight, 2.5, 2.5, 'S');
```

Darstellung dynamischer Daten und Sonderfälle

Falls mehrere Funktionen in der Honorarverteilung hinzugefügt wurden, kann es zu Überlappungen der Inhalte führen welche vermieden werden müssen. Die Export-Logik reagiert entsprechend darauf und beachtet die räumlichen Begrenzungen des Dokuments. Um dies zu vermeiden, berechnet der Algorithmus vor dem Zeichnen der abschließenden Stundenverteilung den verbleibenden Platz auf der aktuellen Seite. Unterschreitet dieser einen definierten Schwellenwert, initiiert die Methode automatisch einen Seitenumbruch (`pdf.addPage()`) und rendert den Kopfbereich auf der neuen Seite erneut.

Weiters wird auch die geschäftliche Logik für Kleinstprojekte berücksichtigt. Anhand des übergebenen Objekts der gesamten `SummaryComponent` wird in der Utility-Datei selbst ermittelt, ob es sich um solch ein Kleinstprojekt handelt oder nicht. Hierfür wird direkt im Code kontrolliert, ob eine Direkteingabe des Honorars vorliegt (`isDirectFee`) oder die Baukostensumme (`ccTotal`) bei 50.000 Euro oder mehr liegt [1]. Während bei regulären Bauvorhaben die berechneten Stundenbudgets in den Tabellen angezeigt werden, überschreibt die Exportfunktion diese Werte bei Kleinstprojekten [1]. In diesem Fall werden die numerischen Stundenangaben bei der Tabellengenerierung durch den textuellen Platzhalter ”Zeitaufwand” ersetzt [1]. Dieser dynamische Eingriff stellt sicher, dass die erzeugten Dokumente inhaltlich fehlerfrei bleiben und den internen Honorarrichtlinien des Auftraggebers exakt entsprechen.

Listing 11: Dynamischer Seitenumbruch und Sonderfallprüfung (pdf-export.util.ts)

```
async function addDistributionSection(
  pdf: jsPDF, startY: number, summary: SummaryComponent
): Promise<number> {
  let y = startY;
  const pageWidth = pdf.internal.pageSize.getWidth();

  /* ... Rendern der Kostenverteilung (ausgeblendet) ... */

  // ---- Stundenverteilung ----
  // Dynamischer Seitenumbruch, falls der Platz nicht ausreicht
  if (y + 40 > pdf.internal.pageSize.getHeight() - 15) {
    pdf.addPage();
    let newY = 10;
    newY = await addPDFHeader(pdf, newY, summary);
    y = newY + 10;
  }

  /* ... Titel setzen (ausgeblendet) ... */

  const stundenRows: string[][] = [];
  summary.distribution.forEach(r => {
    // Pruefung auf Kleinstprojekte (Nettoherstellungskosten < 50.000 Euro)
    const text = (summary.isDirectFee || summary.ccTotal >= 50000)
      ? `${formatNumber(r.hours)} Std.`
      : 'Zeitaufwand';
    stundenRows.push([r.role, text]);
  });

  stundenRows.push([
    'Stunden gesamt',
    (summary.isDirectFee || summary.ccTotal >= 50000)
      ? `${formatNumber(summary.totalHours)} Std.`
      : 'Zeitaufwand'
  ]);

  // Aufruf der generischen Tabellen-Zeichenmethode
  y = addTwoColTableToPDF(
    pdf, y, ['Bezeichnung', 'Stunden'], stundenRows, stundenRows.length - 1
  );

  return y + 6;
}
```

4.6 Desktop-Anwendung mittels Electron

Um die entwickelte Webapplikation zusätzlich als eigenständige Desktop-Anwendung bereitzustellen, wurde das Framework Electron integriert. Damit wird die Ausführung der Angular-Anwendung sowie des Node.js-Backends innerhalb einer vollständig gekapselten, plattformübergreifenden Laufzeitumgebung ermöglicht, wodurch die Software lokal auf den Firmenrechnern des Auftraggebers installiert werden kann.

Architektur und Prozessverwaltung

Der Hauptprozess `main.js` übernimmt die zentrale Steuerung der gesamten Applikation. Da die Architektur aus getrennten Frontend- und Backend-Komponenten besteht, muss Electron beide Teile initialisieren. Zunächst wird über das native Node.js-Modul `child_process` das Backend über die Funktion `spawn` als eigenständiger Hintergrundprozess gestartet. Dabei wird der Port auf 4000 festgelegt und die entsprechenden Umgebungsvariablen des Systems werden an den untergeordneten Prozess vererbt.

Anschließend initialisiert Electron ein neues `BrowserWindow`. Dieses evaluiert, ob es sich um eine Entwicklungs- oder Produktivumgebung handelt. Falls es sich um die Produktivumgebung handelt, wird die Angular-Benutzeroberfläche (`index.html`) geladen und präsentiert.

Listing 12: Initialisierung von Backend und Frontend in Electron (main.js)

```
function startBackend() {
  console.log('Starting backend...');
  const backendPath = isDev
    ? path.join(__dirname, 'backend', 'src', 'index.ts')
    : path.join(__dirname, 'backend', 'dist', 'src', 'index.js');

  const command = isDev ? 'npx' : 'node';
  const args = isDev
    ? ['ts-node', backendPath]
    : [backendPath];

  backendProcess = spawn(command, args, {
    shell: true,
    cwd: path.join(__dirname, 'backend'),
    env: { ...process.env, PORT: 4000 }
  });
}

function createWindow() {
  win = new BrowserWindow({
    width: 1200,
    height: 800,
    webPreferences: {
      nodeIntegration: false,
      contextIsolation: true,
      preload: path.join(__dirname, 'preload.js')
    },
    icon: path.join(__dirname, 'frontend', 'src', 'assets', 'icon_app.png')
  });

  if (isDev) {
    win.loadURL('http://localhost:4200');
    win.webContents.openDevTools();
  } else {
    win.loadFile(path.join(__dirname, 'frontend', 'dist', 'frontend', 'browser',
        'index.html'));
  }
}
```

Build-Prozess und Paketierung

Für die finale Bereitstellung und Auslieferung der Software kommt das Tool `electron-builder` zum Einsatz, welches in der zentralen Konfigurationsdatei (`package.json`) definiert ist und den Erstellungsprozess automatisiert. Die Build-Skripte bündeln sowohl die Frontend-Ressourcen als auch das Backend samt der Prisma-Umgebung und generieren daraus ein ausführbares Installationsprogramm. Spezifische Event-Listener (`window-all-closed` und `will-quit`) im Electron-Hauptprozess stellen sicher, dass beim Beenden der Anwendung auch der im Hintergrund laufende Backend-Prozess ordnungsgemäß beendet und der belegte Port wieder freigegeben wird.

5 Ergebnis

In diesem Kapitel werden die finalen Ergebnisse des Projekts *XCalc* erläutert und bewertet. Es wird berichtet, inwieweit die zu Beginn definierten Ziele und Anforderungen des Auftraggebers (X Architekten ZT GmbH) erfolgreich umgesetzt wurden. Die Strukturierung dieses Kapitels orientiert sich an der technischen Systemarchitektur und gliedert sich in die Teilbereiche Frontend, Backend sowie einer Zusammenfassung des Berechnungsablaufs aus Sicht des Endbenutzers.

5.1 Frontend

Das Frontend der Applikation *XCalc* wurde erfolgreich als dynamische Single-Page Application (SPA) [2] mit dem Framework Angular realisiert. Diese Single-Page-Application wurde anschließend mit dem Electron Framework [5] gebündelt, um auch eine Desktop-Anwendung zur Verfügung zu stellen. Wie vom Auftraggeber gefordert, präsentiert sich die finale Benutzeroberfläche in einem schlichten und streng CI-konformen Design, das sich nahtlos in die bestehenden Unternehmensstrukturen der X Architekten einfügen lässt.

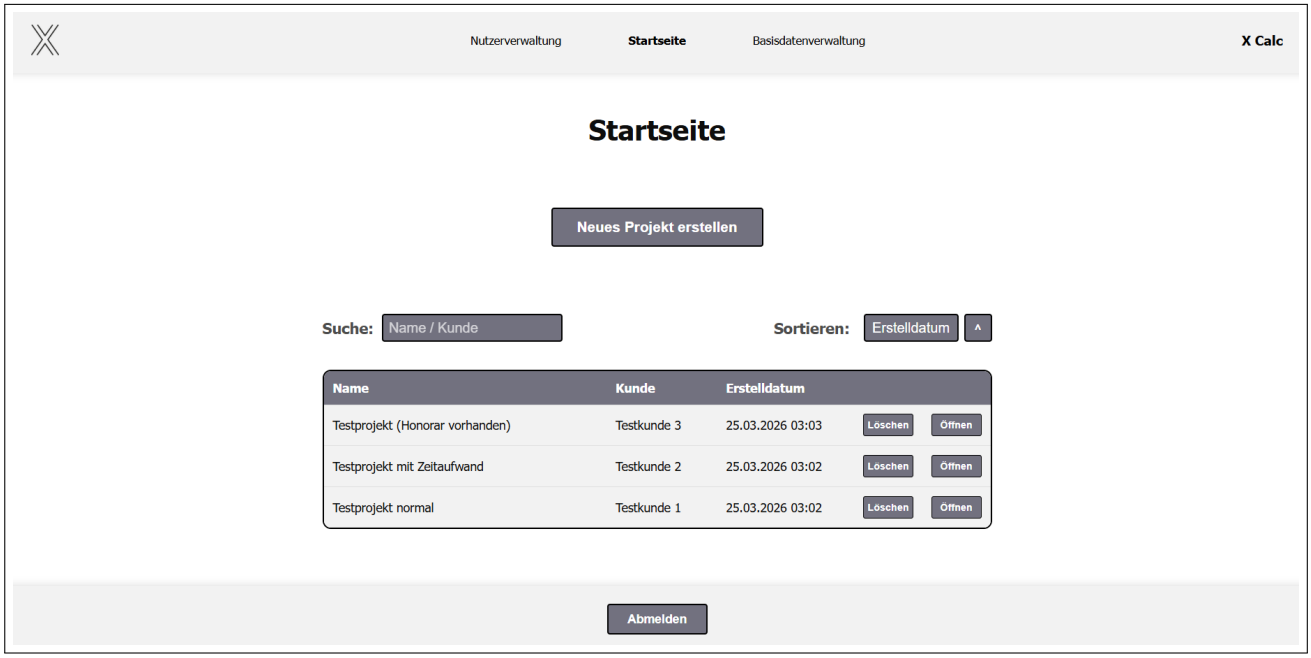


Abbildung 29: Screenshot Vollständige Startseite

Design und Benutzererfahrung (UX)

Ein wichtiger Aspekt für den Projekterfolg war es, die bisherigen, fehleranfälligen Excel-Tabellen zu ersetzen. Dies wurde erreicht, indem die Berechnungsphasen (Baukostenermittlung, Honorarberechnung und Honorarverteilung) logisch getrennt und als strukturierter Workflow gestaltet wurden. Um die Benutzererfahrung (User Experience) zu verbessern, werden während des aktiven Berechnungsvorgangs störende Elemente – wie die globale Navigationsleiste – absichtlich ausgeblendet. So wird die Aufmerksamkeit des Nutzers komplett auf die Eingabe der Daten gerichtet und mögliche Anwendungsfehler eliminiert.

Interaktivität und Vermeidung von Fehlern

Ein weiteres wichtiges Ergebnis der Frontend-Implementierung ist die sofortige visuelle Rückmeldung an den Nutzer. Angulars reaktive Architektur ermöglicht es, dass sämtliche Berechnungen in Echtzeit erfolgen. Wird ein Parameter geändert (wie die Nutzfläche oder den Schwierigkeitsgrad), so werden alle abhängigen Werte und Gesamtsummen ohne Verzögerung aktualisiert.

Die Validierungsmechanismen zur Prävention helfen stark bei der Fehlervermeidung. Die Applikation verhindert fehlerhafte Eingaben, wandelt Rohdaten automatisch in das richtige, länderspezifische Währungsformat um und passt die Benutzeroberfläche bei Sonderfällen (wie dem automatischen Ausblenden von Stundenbudgets bei Kleinstprojekten zugunsten des Textes "Zeitaufwand") dynamisch an.

Zustandsverwaltung (State Management) und integrierter Workflow Ein Merkmal des Frontends ist der flüssige, unterbrechungsfreie Berechnungsworkflow. Mit dem `CalculationStoreService` werden alle Zwischenergebnisse der einzelnen Kalkulationsschritte im Browser lokal gespeichert (`sessionStorage`). Auf diese Weise wird die Serverlast erheblich reduziert, weil nicht nach jeder Eingabe ein Zugriff auf die Datenbank erforderlich ist. Der Benutzer kann über die bereitgestellten Navigationsschaltflächen beliebig zwischen den Phasen wechseln, ohne dass seine Daten verloren gehen. Alle aggregierten Datenpakete werden erst nach dem Verlassen der finalen Zusammenfassungsseite gebündelt an das Backend übermittelt.

Listing 13: Persistierung von Berechnungsdaten (calculation-store.service.ts)

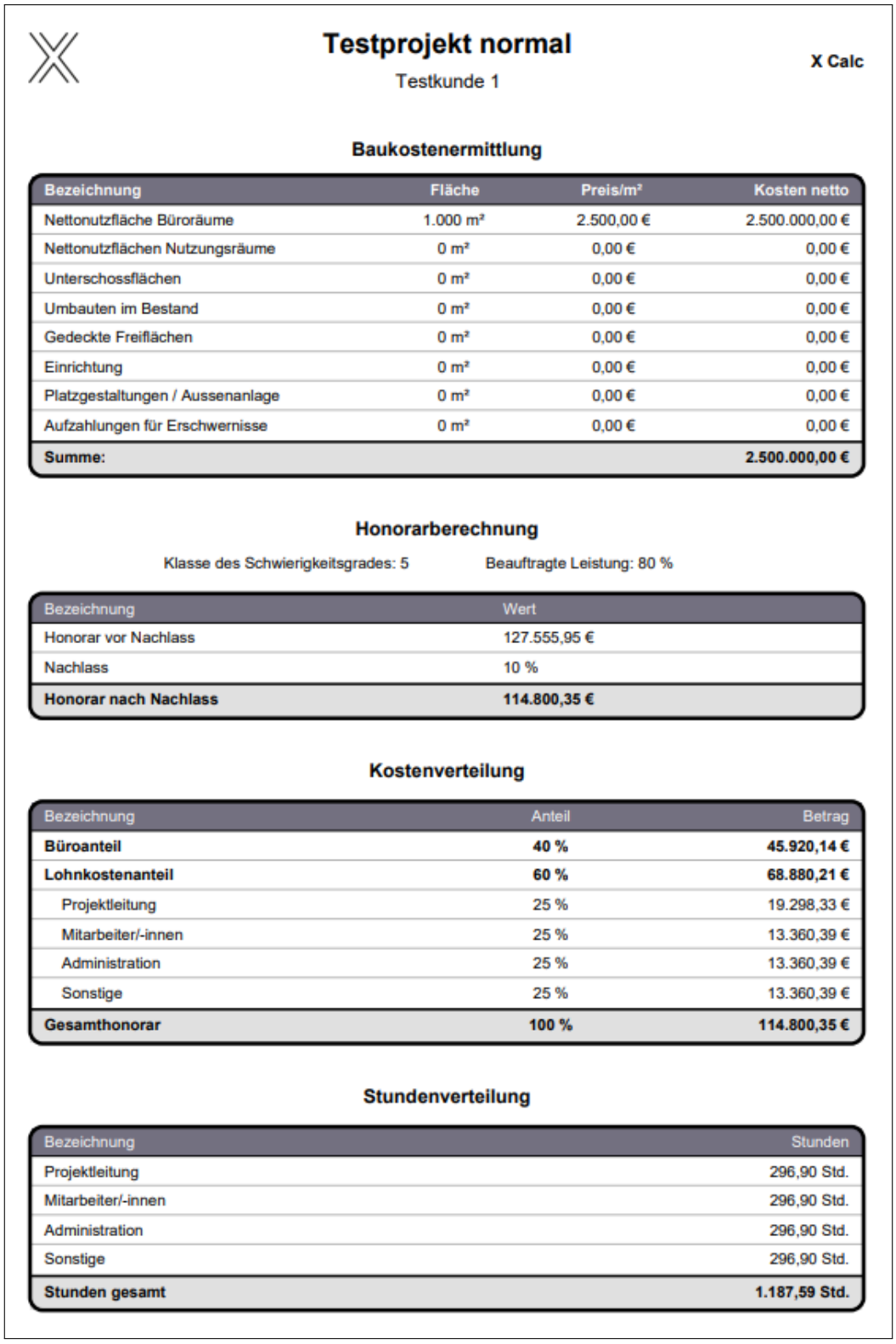
```
private persistToSessionStorage(): void {
  const payload = {
    constructionCost: this._constructionCost.value,
    feeInvoice: this._feeInvoice.value,
    distribution: this._distribution.value,
    internalPercents: this._internalPercents.value,
  };
  sessionStorage.setItem(this.storageKey, JSON.stringify(payload));
}

private hydrateFromSessionStorage(): void {
  const raw = sessionStorage.getItem(this.storageKey);
  if (!raw) return;

  try {
    const parsed = JSON.parse(raw);
    // Wiederherstellung der reaktiven Datenstroeme (Observables)
    this._constructionCost.next(parsed?.constructionCost ?? null);
    this._feeInvoice.next(parsed?.feeInvoice ?? null);
    this._distribution.next(
      Array.isArray(parsed?.distribution) ? parsed.distribution : []
    );
  } catch {
    sessionStorage.removeItem(this.storageKey);
  }
}
```

Rollenbasierte Ansichten und Verwaltung

Das Frontend beinhaltet neben den primären Berechnungsfunktionen auch eine vollwertige, rollenbasierte Verwaltungsoberfläche. Mit dem `authGuard` in der Angular-Routing wird gewährleistet, dass unautorisierte Zugriffe immer verhindert werden. Dynamische Direktiven (wie `*ngIf=ïsAdmin()"`) sorgen dafür, dass spezielle Ansichten, wie die ”Nutzerverwaltung” und die ”Basisdatenverwaltung”, nur jenen Nutzern angezeigt werden, die über die entsprechenden administrativen Berechtigungen verfügen. Berechnungsparameter wie der Grundwert $k0$ oder der Planungsfaktor können von Administratoren direkt über das User Interface angepasst werden, ohne dass sie in den Code eingreifen müssen.

Automatisierte Dokumentengenerierung

Ein weiteres wichtiges Ergebnis der Frontend-Entwicklung ist der PDF-Export. Benutzer können zum Abschluss des Workflows ein professionelles PDF-Dokument erstellen, das der Corporate Identity (CI) entspricht. Die Erstellung des Exports ist dank der `jsPDF`-Bibliothek, die alle Export-Logik clientseitig verarbeitet, nicht nur sehr performant, sondern sie spiegelt auch exakt das Layout der Zusammenfassung wider – abgerundete Container und farblich hervorgehobene Summenzeilen inklusive.



Testprojekt normal

Testkunde 1

X Calc

Baukostenermittlung

Bezeichnung	Fläche	Preis/m²	Kosten netto
Nettonutzfläche Büroräume	1.000 m²	2.500,00 €	2.500.000,00 €
Nettonutzflächen Nutzungsräume	0 m²	0,00 €	0,00 €
Unterschossflächen	0 m²	0,00 €	0,00 €
Umbauten im Bestand	0 m²	0,00 €	0,00 €
Gedeckte Freiflächen	0 m²	0,00 €	0,00 €
Einrichtung	0 m²	0,00 €	0,00 €
Platzgestaltungen / Aussenanlage	0 m²	0,00 €	0,00 €
Aufzahlungen für Erschwernisse	0 m²	0,00 €	0,00 €
Summe:			**2.500.000,00 €**

Honorarberechnung

Klasse des Schwierigkeitsgrades: 5 Beauftragte Leistung: 80 %

Bezeichnung	Wert
Honorar vor Nachlass	127.555,95 €
Nachlass	10 %
Honorar nach Nachlass	**114.800,35 €**

Kostenverteilung

Bezeichnung	Anteil	Betrag
Büroanteil	**40 %**	**45.920,14 €**
Lohnkostenanteil	**60 %**	**68.880,21 €**
Projektleitung	25 %	19.298,33 €
Mitarbeiter/-innen	25 %	13.360,39 €
Administration	25 %	13.360,39 €
Sonstige	25 %	13.360,39 €
Gesamthonorar	**100 %**	**114.800,35 €**

Stundenverteilung

Bezeichnung	Stunden
Projektleitung	296,90 Std.
Mitarbeiter/-innen	296,90 Std.
Administration	296,90 Std.
Sonstige	296,90 Std.
Stunden gesamt	**1.187,59 Std.**

Abbildung 30: Screenshot Exportiertes PDF-Dokument

Zusammenfassend lässt sich festhalten, dass das Frontend nicht nur die technischen Spezifikationen erfüllt, sondern die täglichen Arbeitsabläufe der Mitarbeiter durch eine klare Menüführung, schnelle Reaktionszeiten und die Integration des PDF-Exports signifikant vereinfacht.

5.2 Backend

Die REST-API im Backend von XCalc ist komplett funktionsfähig und stellt alle Berechnungs-, Verwaltungs- und Authentifizierungsfunktionen der Anwendung zur Verfügung. Es besteht aus etwa 3.000 Zeilen TypeScript-Code, die sich über 30 Quelldateien verteilen.

5.2.1 Architektur

Die in Abschnitt 4.3 beschriebene dreischichtige Architektur (Routen, Services, Prisma Client) wurde konsequent umgesetzt und eingehalten. Dank dieser Trennung ist es möglich, jede Schicht unabhängig zu testen und anzupassen. Die Routen haben keine Kenntnis von der Datenbankstruktur, während die Services keine HTTP-Statuscodes verwenden. So bleibt der Code übersichtlich und lässt sich leicht erweitern.

5.2.2 REST-API

Die fertige API umfasst 22 Pfade mit insgesamt 37 HTTP-Methoden. Die Endpunkte verteilen sich auf neun Routen-Module:

- `auth` – Anmeldung
- `user` – Benutzerverwaltung (Registrierung, Passwort, Benutzername)
- `project` – Projekte anlegen, auflisten, bearbeiten, löschen
- `constructionCost` – Baukostenermittlung je Projekt
- `constructionCostItem` – einzelne Baukostenpositionen
- `feeInvoice` – Honorarberechnung je Projekt
- `feeDistribution` – Honorarverteilung je Projekt
- `feeDistributionRole` – einzelne Rollen in der Verteilung
- `baseData` – systemweite Berechnungsparameter (versioniert)

In jedem dieser Bereiche gibt es eine entsprechende Service-Datei. Auf der Routeebene werden die Eingaben überprüft und Fehler mit passenden HTTP-Statuscodes beantwortet, während die Services die Datenbankoperationen über den Prisma Client ausführen.

JSDoc-Kommentare im OpenAPI-Format sind allen Endpunkten beigefügt. Die Swagger-Dokumentation, die daraus generiert wird, ist unter `/docs` zu finden und ermöglicht es, jeden Endpunkt direkt im Browser auszuprobieren. Die Spezifikation wird bei jedem Serverstart im Projektverzeichnis als `swagger.json` abgelegt.

5.2.3 Datenbank (DB)

In einem Docker-Container läuft eine PostgreSQL-15-Instanz, auf der acht Prisma-Modelle in dem Datenbankschema ausgeführt werden. Die Schemaentwicklung ist in 17 Prisma-Migrationen festgehalten, die von Juli bis Oktober 2025 reichen. Die Migrationen illustrieren den schrittweisen Aufbau der Datenbank: beginnend mit der `init-Migration`, gefolgt von Anpassungen am Benutzermodell, der Einführung des Honorarordnungsparameters `hop` sowie der Ergänzung um Summary-Snapshots und endend mit der Konsolidierung im finalen Schema.

Ein Seed-Skript (`prisma/seed.ts`) ist vorhanden, um die Datenbank initial zu befüllen. Es generiert eine `BaseDataVersion` mit realistischen Werten ($k0 = 255.000$, $f1 = 0{,}65$, $f2 = 0{,}55$, $hop = 0{,}058$), legt einen Administrator und einen regulären Benutzer mit gehashten Passwörtern an und generiert außerdem für beide jeweils ein Beispielprojekt mit vollständiger Baukostenermittlung und Honorarberechnung. So kann die Anwendung nach der Einrichtung sofort mit Testdaten ausprobiert werden.

5.2.4 Sicherheit

Die in Abschnitt 4.3.1 und 4.3.3 beschriebenen Sicherheitsmaßnahmen (Helmet, CORS, express-validator) wurden vollständig umgesetzt. In Kombination mit der JWT-basierten Authentifizierung und der rollenbasierten Autorisierung über `requireAdmin` ergibt sich ein durchgängiges Sicherheitskonzept, das den Anforderungen einer Intranet-Anwendung entspricht. Das Token wird als HttpOnly-Cookie übermittelt, sodass es nicht über JavaScript ausgelesen werden kann und das Risiko von Cross-Site-Scripting-Angriffen reduziert wird.

5.2.5 Deployment

Für den Betrieb stehen drei Shell-Skripte im Verzeichnis `scripts/` bereit:

- `migrate.sh` – führt die Datenbankmigrationen aus
- `seed.sh` – befüllt die Datenbank mit Beispieldaten
- `start.sh` – startet den kompilierten Server

Die Konfiguration des Backends erfolgt über Umgebungsvariablen in einer `.env-Datei`. Mindestens drei Variablen sind erforderlich: `DATABASE_URL` für die Verbindung zur PostgreSQL-Datenbank, `JWT_SECRET` für die Signierung der JSON Web Tokens und `PORT` für den Server-Port (standardmäßig 4000). Die PostgreSQL-Datenbank wird über eine `docker-compose.yml` mit einem einzigen Befehl (`docker-compose up`) gestartet. Ein benanntes Volume stellt die Datenpersistenz sicher. Beim Starten des Servers wird die Swagger-Dokumentation unter `/docs` automatisch im Standardbrowser geöffnet, sodass die API-Endpunkte sofort getestet werden können.

Zusammenfassend erfüllt das Backend alle funktionalen Anforderungen und bietet durch die konsequente Schichtentrennung eine solide Grundlage für zukünftige Erweiterungen. Eine mögliche Verbesserung wäre die Einführung automatisierter Tests (Unit- und Integrationstests), die im aktuellen Projektumfang nicht umgesetzt wurden, aber die langfristige Wartbarkeit weiter erhöhen würden.

5.3 Berechnungsablauf

Im Folgenden wird der typische Ablauf der Anwendung aus Sicht des Anwenders zusammenfassend beschrieben. Die einzelnen Schritte bilden den vollständigen Berechnungsworkflow ab – von der initialen Projekterstellung bis hin zum fertigen PDF-Exportdokument.

Projekterstellung

Der Vorgang beginnt auf dem zentralen Dashboard (Startseite) der Applikation. Über die Schaltfläche ”Neues Projekt erstellen” öffnet sich ein mehrstufiges Dialogfenster. Im ersten Schritt definiert der Benutzer den Projektnamen und den zugehörigen Kunden.

Im zweiten Schritt kann er sich zwischen zwei Arten des Berechnungsworkflows entscheiden:

- Honorar bereits vorhanden
- Honorar nicht vorhanden

Entscheidet sich der Benutzer für die Option "Honorar nicht vorhanden", startet der reguläre Prozess mit der Baukostenermittlung. Wählt er hingegen "Honorar bereits vorhanden", werden die ersten beiden Kalkulationsphasen übersprungen und das System leitet direkt zur Honorarverteilung weiter.

Baukostenermittlung

Sofern der reguläre Workflow gewählt wurde, erfasst der Benutzer im ersten Berechnungsschritt die Nettoherstellungskosten des Bauvorhabens. Die Benutzeroberfläche stellt hierfür eine interaktive Tabelle mit acht vorgegebenen Bauwerkskategorien bereit. Der Anwender trägt für die relevanten Kategorien die jeweilige Fläche in Quadratmetern sowie den Quadratmeterpreis ein. Die Anwendung berechnet daraufhin in Echtzeit die Nettokosten der einzelnen Positionen sowie die Gesamtsumme der Baukosten. Der benutzer muss dabei nicht für jede Bauwerkskategorie Daten eintragen, sondern kann sie beliebig auslassen und sobald er fertig ist, kann über den Button "Übermitteln" mit der Honorarberechnung fortsetzen.

Honorarberechnung

Im zweiten Schritt werden die zuvor ermittelten Nettoherstellungskosten automatisch als Basiswert übernommen.

Der Benutzer definiert nun die projektspezifischen Parameter für die Honorarermittlung:

- Klasse des Schwierigkeitsgrades (1 bis 10)
- Beauftragten Leistung in Prozent (1 bis 100)
- Nachlass (optional, 0-100)

Aus diesen Variablen kalkuliert das System ohne Verzögerung den Planungsfaktor, den Basishonorarsatz sowie das resultierende Honorar vor und nach dem Nachlass. Über die Bestätigung der Eingaben wird der Nutzer zur finalen Honorarverteilung weitergeleitet.

Honorarverteilung

Falls die Option "Honorar nicht vorhanden" gewählt wurde wird bei der Honorarverteilung das berechnete Gesamthonorar aus der Honorarberechnung übernommen. Wenn die andere Option "Honorar bereits vorhanden" ausgewählt wurde, überspringt der Benutzer die ersten beiden Berechnungsschritte und ihm steht ein separates Eingabefeld am Anfang der Benutzeroberfläche zur Verfügung, in dem er das Honorar manuell eingeben kann.

Die Honorarverteilung selbst beginnt indem der Benutzer festlegt, wie sich das Honorar prozentual in einen Büro- und einen Lohnkostenanteil gliedert. Der ermittelte Euro-Betrag des Lohnkostenanteils wird danach auf die beteiligten Mitarbeiterfunktionen verteilt. Es existieren vier vordefinierte Funktionen. Weitere Funktionen können flexibel hinzugefügt werden. Für jede Funktion berechnet das System aus dem gewünschten Prozentanteil und dem individuell festgelegten Stundensatz automatisch das zur Verfügung stehende Stundenbudget. Ein Abschluss der Honorarverteilung ist nur dann möglich, wenn die Summe aller zugewiesenen Prozentanteile exakt 100 % ergibt und alle Stundensätze mindestens einen Wert über 0 besitzen.

Zusammenfassung und Abschluss

In der Zusammenfassung werden die Ergebnisse aller durchlaufenen Berechnungsschritte übersichtlich dargestellt. Der Anwender hat hier drei Möglichkeiten, den Vorgang zu beenden:

Über den Button **"Zurück zur Startseite"** wird die Berechnung erfolgreich abgeschlossen und alle Daten des Workflows werden dauerhaft für spätere Bearbeitungen oder Einsehen der Daten gespeichert.

Die Aktion **"PDF Export"** erzeugt ein formatiertes Dokument im Corporate Design der X Architekten ZT GmbH, welches die Zusammenfassung übersichtlich exportiert.

Über **"Projekt verwerfen"** wird der gesamte Kalkulationsvorgang abgebrochen, die erfassten Daten werden rückstandslos gelöscht und der Benutzer kehrt zum Dashboard zurück. Diese Option befindet sich auch in jedem Berechnungsschritt.

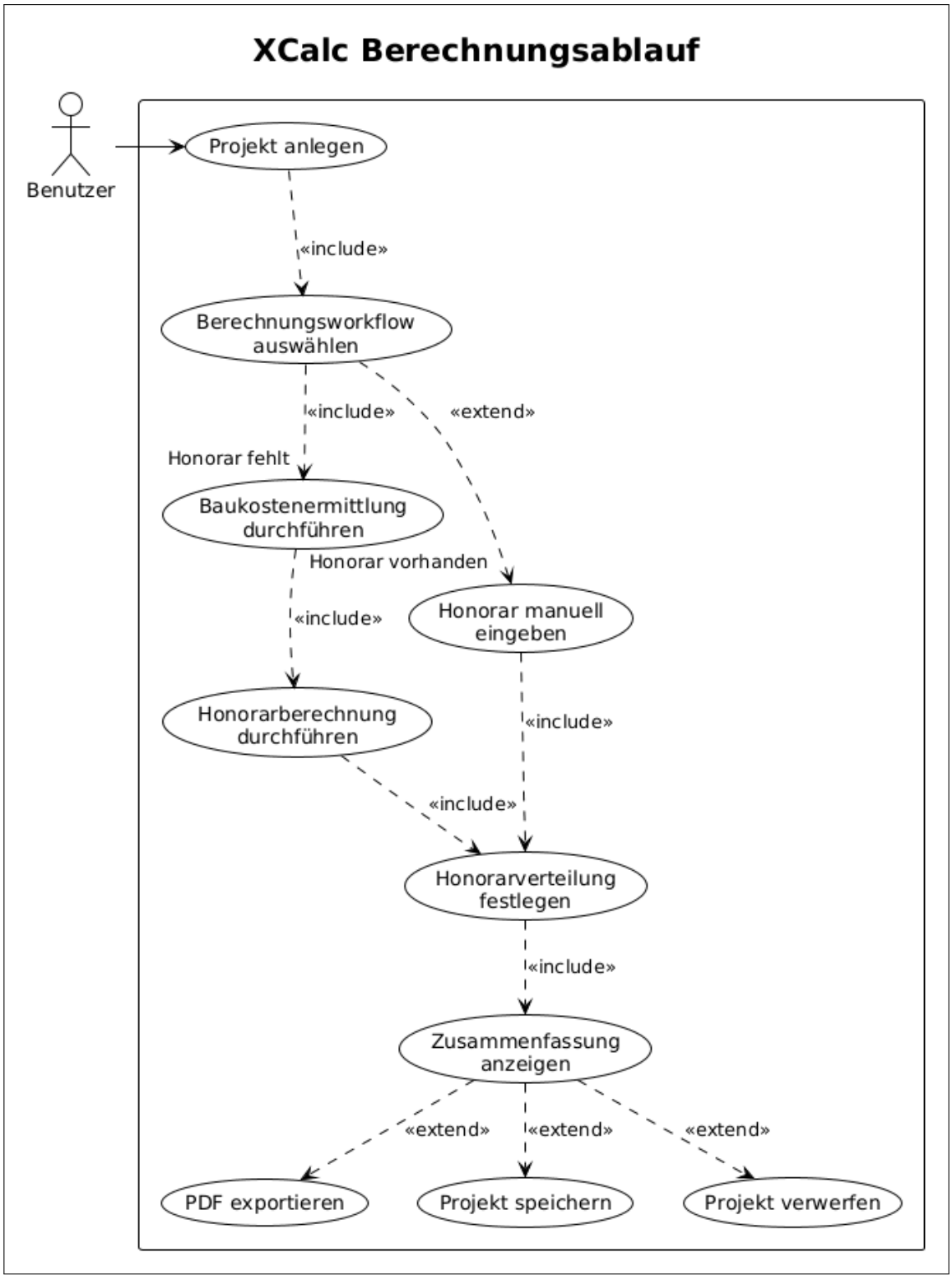


Abbildung 31: Use-Case-Diagramm Berechnungsablauf

Quelle: eigene Darstellung, gerendert mit PlantUML [1]

6 Resümee

Während der vierwöchigen Entwicklungszeit im Sommer 2025 haben wir den Grundstein für die Diplomarbeit legen können. Rückblickend hätte in dieser Phase noch mehr erreicht werden können, da der direkte Austausch mit dem Auftraggeber vor Ort deutlich einfacher war als in den darauffolgenden Wochen und Monaten.

Nichtsdestotrotz war es eine wertvolle Erfahrung, in einem Unternehmen zu arbeiten, das nicht primär Softwareentwicklung betreibt und keine eigene IT-Abteilung hat. Dadurch waren wir, mit Ausnahme der Unterstützung durch unseren Betreuungslehrer, weitgehend auf uns allein gestellt. Dieser forderte jede Woche einen Statusbericht, um sicherzustellen, dass kontinuierlicher Fortschritt erzielt und der Entwicklungsprozess zeitig abgeschlossen werden konnte. Auch im weiteren Verlauf, insbesondere während des Schreibprozesses, gab er klare Vorgaben, die einzuhalten waren. Gleichzeitig zeigte er Verständnis, wenn eine Deadline aus guten Gründen nicht eingehalten werden konnte.

Das ursprüngliche Ziel, das Produkt bis zum Beginn des Schuljahres fertigzustellen, konnte aufgrund fortlaufender Änderungswünsche nicht erreicht werden. Dennoch wurde das Wesentliche realisiert: Die Anwendung entspricht den Anforderungen des Auftraggebers, und die Diplomarbeit konnte ordentlich verfasst werden.

7 Aufgabenverteilung

Da das Team aus zwei Mitgliedern besteht und man die Implementierung in Frontend und Backend unterteilen kann, wurde auch die Aufgabenverteilung nach diesen Prinzip gemacht. Simon Brunner hat sich um die Datenbank gekümmert und sämtliche Backend-Logik inklusive der REST-API implementiert und kümmert sich um das Deployment. Daniel Matzinger hat sich hingegen voll auf das Frontend konzentriert. Beide Mitglieder haben sich allerdings trotzdem gegenseitig beratet und bei Problemen ausgeholfen. Bei der Verschriftlichung wurde das gleiche Prinzip verfolgt und die Kapitel entsprechend zu dem was programmiert wurde, aufgeteilt.

7.1 Simon Brunner

In der Entwicklung war die Aufgabe das Aufsetzen der Datenbank und die Programmierung des Backends. Zu Beginn wurde die PostgreSQL-Datenbank auf einem Docker-Container aufgesetzt sowie ein initiales Datenmodell entworfen und umgesetzt. Parallel zur weiteren Backendentwicklung wurde das Datenmodell im Laufe des Projekts aufgrund neuer Erkenntnisse und sich ändernder Anforderungen kontinuierlich angepasst. Das Backend wurde mit Node.js, Express und TypeScript umgesetzt. Nach der Initialisierung des Projekts und der Erstellung der ersten Middleware wurde die Geschäftslogik in einer Service-Schicht implementiert. Anschließend wurden das Routing, die API-Endpunkte sowie die JWT-basierte Authentifizierung realisiert. Abschließend wurden alle API-Endpunkte mit Swagger dokumentiert.

Folgende Kapitel wurden ausgearbeitet:

- 1 - Einleitung
- 2 - Grundlagen und Methoden
- 3.1 - Projektorganisation
- 3.3 - Projektzeitplan
- 4.1 bis 4.3
- 4.4.1 - Login-Prozess
- 5.2 - Backend

7.2 Daniel Matzinger

Die Aufgabe in der Entwicklung bestand darin das Frontend zu programmieren. Hierbei wurde zu allererst über Figma ein Design, mit Rücksprache des Auftraggebers erstellt. Danach wurde Schritt für Schritt die Angular Komponenten programmiert. Dabei wurde immer zuerst die TypeScript-Datei erstellt um daraufhin das HTML entsprechend zu programmieren. Dieser Ablauf wiederholte sich bis das gewünschte Ergebnis erreicht wurde. Für das Design mittels einfachen CSS wurde der Grundstein mit KI generiert und entsprechende Details und kleine Änderungen wurden selbst implementiert. Komplexe Algorithmen wurden anfangs in die jeweilige Komponente geschrieben und anschließend in eine Utility-Datei übertragen um das Prinzip `Separation of Concerns` anzuwenden. Abgesehen von der Entwicklung des Frontend wurden auch die meisten Dokumente erstellt und Statusberichte geschrieben.

Aufgrunddessen wurden folgende Kapitel ausgearbeitet:

- 4.4.2 bis 4.4.6
- 4.5 - Berechnungen und PDF-Export
- 4.6 - Desktop-Anwendung mittels Electron
- 5.1 - Frontend
- 5.3 - Ablauf
- 6 - Resümee
- 7 - Aufgabenverteilung

Literaturverzeichnis

[1] PlantUML Team, „PlantUML Language Reference Guide," 2026, letzter Zugriff am 20.03.2026. Online verfügbar: https://plantuml.com/guide

[2] Google LLC, „Angular – Introduction to the Angular Docs," 2024, letzter Zugriff am 01.02.2025. Online verfügbar: https://angular.dev/overview

[3] Microsoft Corporation, „TypeScript Documentation," 2024, letzter Zugriff am 01.02.2025. Online verfügbar: https://www.typescriptlang.org/docs/

[4] J. Hall *et al.*, „jsPDF – A library to generate PDFs in JavaScript," 2024, letzter Zugriff am 01.02.2025. Online verfügbar: https://github.com/parallax/jsPDF

[5] OpenJS Foundation, „Electron Documentation," 2024, letzter Zugriff am 20.02.2026. Online verfügbar: https://www.electronjs.org/docs/latest

[6] ——, „Node.js – About Node.js," 2024, letzter Zugriff am 01.02.2025. Online verfügbar: https://nodejs.org/en/about

[7] ——, „Express – Fast, unopinionated, minimalist web framework for Node.js," 2024, letzter Zugriff am 01.02.2025. Online verfügbar: https://expressjs.com

[8] The PostgreSQL Global Development Group, „PostgreSQL 15 Documentation," 2024, letzter Zugriff am 01.02.2025. Online verfügbar: https://www.postgresql.org/docs/15/

[9] Prisma Data, Inc., „Prisma Documentation – Overview," 2024, letzter Zugriff am 01.02.2025. Online verfügbar: https://www.prisma.io/docs

[10] Docker Inc., „Docker Documentation – Overview," 2024, letzter Zugriff am 01.02.2025. Online verfügbar: https://docs.docker.com/get-started/overview/

[11] Microsoft Corporation, „Documentation for Visual Studio Code," 2024, letzter Zugriff am 01.02.2025. Online verfügbar: https://code.visualstudio.com/docs

[12] GitHub, Inc., „GitHub Docs," 2024, letzter Zugriff am 01.02.2025. Online verfügbar: https://docs.github.com

[13] S. Chacon und B. Straub, „Pro Git," 2014, 2. Auflage. Apress, letzter Zugriff am 01.02.2025. Online verfügbar: https://git-scm.com/book/en/v2

[14] SmartBear Software, „Swagger Documentation," 2024, letzter Zugriff am 01.02.2025. Online verfügbar: https://swagger.io/docs/

[15] Postman, Inc., „Postman Learning Center," 2024, letzter Zugriff am 01.02.2025. Online verfügbar: https://learning.postman.com/docs/

[16] Docker Inc., „Docker Desktop Overview," 2024, letzter Zugriff am 01.02.2025. Online verfügbar: https://docs.docker.com/desktop/

[17] TRELLO, „Trello," letzter Zugriff am 18.01.2025. Online verfügbar: https://trello.com

[18] express-validator Contributors, „express-validator Documentation," 2024, letzter Zugriff am 01.02.2025. Online verfügbar: https://express-validator.github.io/docs/

[19] Mozilla Developer Network, „Set-Cookie: SameSite – HTTP | MDN," 2024, letzter Zugriff am 20.03.2026. Online verfügbar: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie#samesite

[20] OWASP Foundation, „Session Management Cheat Sheet," 2024, letzter Zugriff am 20.03.2026. Online verfügbar: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

[21] M. Jones, J. Bradley, und N. Sakimura, „RFC 7519: JSON Web Token (JWT)," 2015, letzter Zugriff am 20.03.2026. Online verfügbar: https://datatracker.ietf.org/doc/html/rfc7519

[22] npm, Inc., „bcrypt – A library to help you hash passwords (npm package)," 2024, letzter Zugriff am 20.03.2026. Online verfügbar: https://www.npmjs.com/package/bcrypt

[23] Mozilla Developer Network, „Intl.NumberFormat – JavaScript | MDN," 2024, letzter Zugriff am 20.03.2026. Online verfügbar: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Intl/NumberFormat

[24] ——, „Window: sessionStorage Property – Web APIs | MDN," 2024, letzter Zugriff am 20.03.2026. Online verfügbar: https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage

[25] E. W. Dijkstra, „On the Role of Scientific Thought," 1982, In: Selected Writings on Computing: A Personal Perspective. Springer. letzter Zugriff am 20.03.2026. Online verfügbar: https://www.cs.utexas.edu/~EWD/transcriptions/EWD04xx/EWD447.html

[26] X Architekten ZT GmbH, „Interne Berechnungsgrundlage für die Honorarermittlung," 2025, Unveröffentlichtes internes Dokument, bereitgestellt im Rahmen der Kooperation zur Diplomarbeit XCalc.

Abbildungsverzeichnis

Quellcodeverzeichnis

Anhang

A Logo

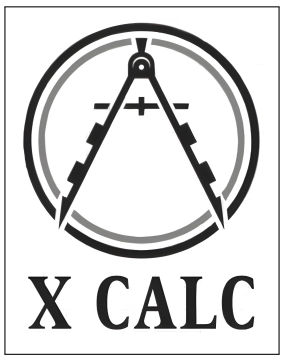


Abbildung 32: Projektlogo

Quelle: selbst erstellte Grafik

B Projektplakat

Abbildung 33: Projektplakat

Quelle: selbst erstellte Grafik

C Kooperationsangebot

htlperg
innovativ | kreativ | praxisnah

Höhere Technische Bundeslehranstalt Perg

Machlandstraße 48, A-4320 Perg
http://www.htl-perg.ac.at
office@htl-perg.ac.at
0 72 62 / 5 39 26

Kooperationsangebot an die HTL – Perg

☒ Höhere Abteilung für Informatik ☐ Fachschule für Informationstechnik

Unternehmen:	X ARCHITEKTEN
Straße:	Wiener Str. 21
PLZ, Ort, Staat:	4020, Linz, Österreich
Ansprechperson:	Bettina Brunner-Krenn
Position:	Geschäftsführung
Telefon:	06641059080
Fax:	
E-Mail:	b.brunner-krenn@xarchitekten.at
Web-Adresse:	https://www.xarchitekten.at/

Gewünschte Kooperation (Mehrfachnennung möglich):

☒ Ferialpraktikum
☐ Schulprojekt
☒ Diplomarbeit
☐ Berufspraktikum (Fachschule)
☐ Abschlussprojekt (Fachschule)

Betroffene SchülerInnen (falls bekannt): Simon Brunner, Daniel Matzinger

Titel/Thema der Kooperationsidee:

Honorarermittlungstool

Kurze Beschreibung der Kooperationsidee:

Die Kooperationsidee sieht vor, gemeinsam ein praxisorientiertes Honorarermittlungstool für Architekturplanungsleistungen zu entwickeln. Das Tool nutzt Basisdaten aus den Bauwerkskosten pro Quadratmeter und ordnet diese einer variablen Schwierigkeitsklasse zu, um eine realitätsnahe Kalkulation zu ermöglichen. Dabei wird die Honorarsumme in einen fixen Büroanteil und einen personalabhängigen Stundenaufwand aufgeteilt – was eine präzise Prognose der Bearbeitungsdauer erlaubt. In enger Abstimmung mit der Kammer der Architekten und Ingenieurkonsulenten soll das System an die aktuellen Bedürfnisse der Branche angepasst werden, ohne die oft unvermeidlichen Herausforderungen zu beschönigen.

X ARCHITEKTEN ZT GMBH
Staatlich befugte und beeidigte Ziviltechniker
Wiener Straße 21 A-4020 Linz
T +43 (0)732 [illegible]607
linz@xarchitekten.at

Frühest möglicher Beginn:	23.06.2025
Spätest mögliches Ende:	04.2026

25.March 2025
Datum

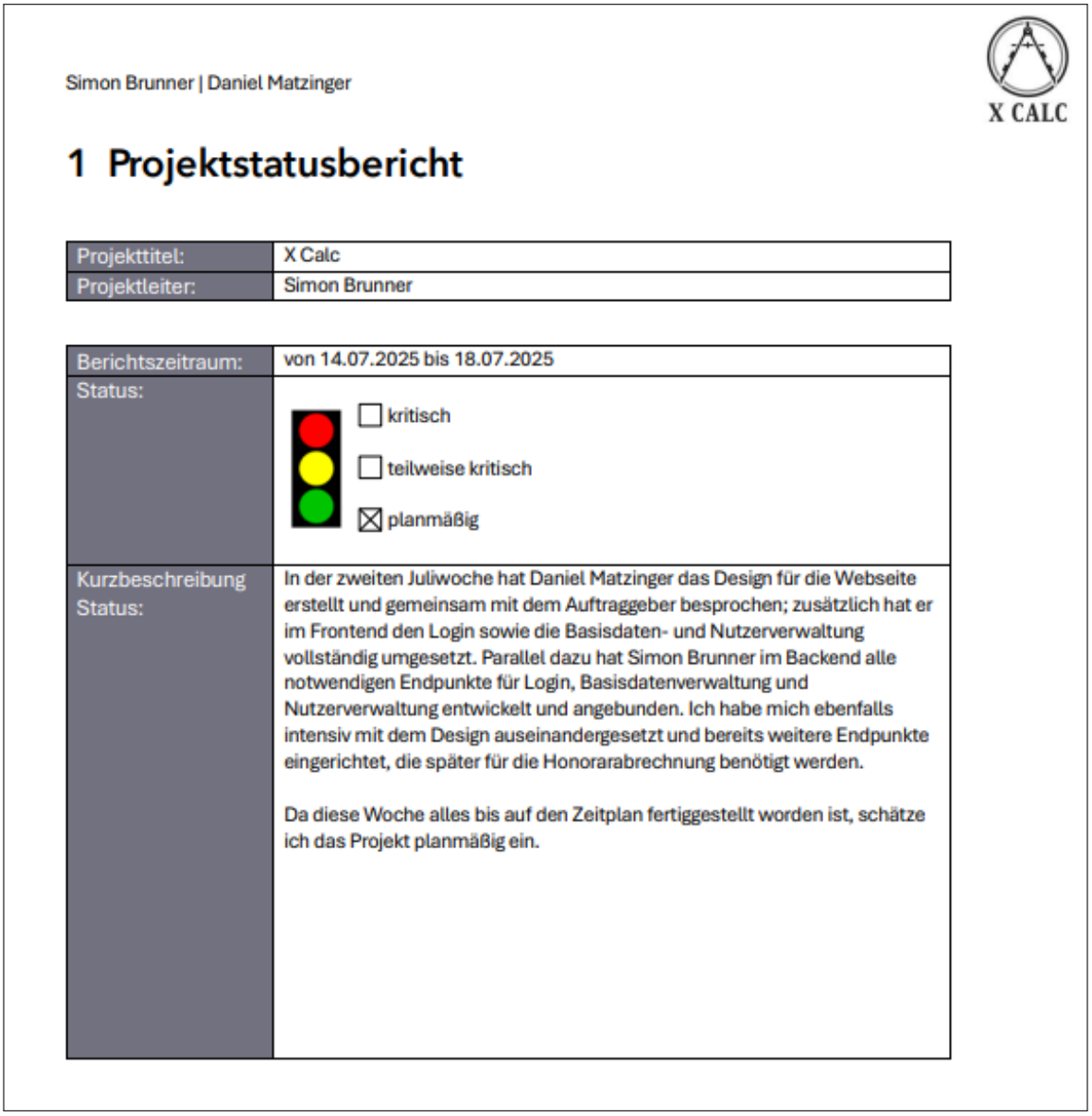
Unterschrift

Bitte senden Sie dieses Formular an die HTL-Perg, Machlandstraße 48, A-4320 Perg, oder via E-Mail an office@htl-perg.ac.at

Abbildung 34: Kooperationsangebot

Quelle: selbst erstelltes Dokument

D Statusbericht



Simon Brunner | Daniel Matzinger

X CALC

1 Projektstatusbericht

Projekttitel:	X Calc
Projektleiter:	Simon Brunner

Berichtszeitraum:	von 14.07.2025 bis 18.07.2025
Status:	☐ kritisch ☐ teilweise kritisch ☒ planmäßig
Kurzbeschreibung Status:	In der zweiten Juliwoche hat Daniel Matzinger das Design für die Webseite erstellt und gemeinsam mit dem Auftraggeber besprochen; zusätzlich hat er im Frontend den Login sowie die Basisdaten- und Nutzerverwaltung vollständig umgesetzt. Parallel dazu hat Simon Brunner im Backend alle notwendigen Endpunkte für Login, Basisdatenverwaltung und Nutzerverwaltung entwickelt und angebunden. Ich habe mich ebenfalls intensiv mit dem Design auseinandergesetzt und bereits weitere Endpunkte eingerichtet, die später für die Honorarabrechnung benötigt werden. Da diese Woche alles bis auf den Zeitplan fertiggestellt worden ist, schätze ich das Projekt planmäßig ein.

Abbildung 35: Ausschnitt des Statusbericht aus Woche 2

E Künstliche Intelligenz

Im Laufe der Diplomarbeit wurde bei der Programmierung der Anwendung XCalc und während der Verschriftlichung diverse, öffentlich zugängliche KI verwendet:

- Google Gemini - erreichbar unter https://gemini.google.com/app?hl=de
- Google NotebookLLM - erreichbar unter https://notebooklm.google.com
- Claude - erreichbar unter https://claude.ai
- ChatGPT - erreichbnar unter https://chatgpt.com
- QuillBot - erreichbar unter https://quillbot.com

In der Programmierung kam es hauptsächlich beim Designen der Pages zum Einsatz. Hier wurde immer zuvor die Grundlage selbst erstellt, um daraufhin einzelne Besonderheiten mit der KI zu lösen. Auch bei der Fehlerbehebung wurde oft nach einer genauen Erklärung des Fehlers gefragt oder es wurde nach einer Einleitung in Electron gefragt, da dies eine komplett neue Technologie für das betroffene Projektmitglied Daniel Matzinger ist.

In der Verschriftlichung wurde der KI oft die gesamte Verschriftlichung zur Verfügung gestellt, um Prompts wie:

- Kontrolliere auf sämtliche Rechtschreibfehler
- Wird das Wort ”man” oft benutzt?
- Kontrolliere ob überall in der Gegenwartsform (Präsens, Perfekt, Futur) geschrieben wurde

auszuführen und Fehler leichter zu erkennen und schneller zu beheben. Da keine Zeit mit der Syntax von Latex verschwendet werden wollte wurde auch hier sehr oft zur Hilfe von Formatierungen und Fehlern die KI benutzt. Als der Rohtext der Diplomarbeit fertiggeschrieben wurde, haben die Mitglieder nach Formulierungsverbesserungen gefragt und kontrolliert ob die Erklärungen exakt dem Code der Anwendung entsprechen.