Titel:

Web-applikation für organisatorische Planung im Alltag auf Basis von C++

Lernfeld 12:
Kundenspezifische Anwendungsentwicklung durchführen

Bea Janott
Mikel Thiele
Jake Pettke
Alma Kosuta
Leo Hepting
Philip Adam

Ausbildungsberuf:

Fachinformatiker*in für Anwendungsentwicklung

Inhaltsverzeichnis

1. Projektbeschreibung

1.1 Projektumfeld

Das Projekt wurde im Rahmen der Berufsschule als praxisorientierte Übung zur Vorbereitung auf die Abschlussprüfung Teil 2 im IT-Bereich durchgeführt. Ziel war es, ein realistisches Projekt zu planen, umzusetzen und gemäß den formalen Anforderungen der IHK zu dokumentieren. Die Projektarbeit erfolgte als Gruppenarbeit mit klarer Aufgabenverteilung. Bea Janott und Mikel Thiele waren für die Projektdokumentation verantwortlich. Die Backend-Entwicklung in C++ wurde von Jake Pettke und Philip Adam umgesetzt, während das Frontend mit HTML, CSS und JavaScript von Leo Hepting und Alma Kosuta realisiert wurde. Zur Zusammenarbeit und Versionsverwaltung wurde ein gemeinsames GitHub-Repository genutzt, in dem der gesamte Quellcode zentral verwaltet wurde.

1.2 Ausgangssituation

Die Ausgangssituation für dieses Projekt ergab sich aus der Beobachtung, dass im schulischen Alltag Aufgaben, Termine und Abgabefristen häufig nicht zentral erfasst werden. Schülerinnen und Schüler sowie Projektbeteiligte nutzen unterschiedliche Hilfsmittel wie handschriftliche Notizen, Chatverläufe oder einfache Tabellen, um ihre Aufgaben zu organisieren. Durch diese dezentrale Vorgehensweise fehlt oftmals ein einheitlicher Überblick über anstehende Aufgaben und deren Priorität. Insbesondere bei parallellaufenden Unterrichtsfächern und Projekten entsteht das Risiko, Fristen zu übersehen oder Aufgaben doppelt zu erfassen. Zusätzlich zeigte sich, dass viele existierende Aufgabenverwaltungs-Tools entweder zu umfangreich und komplex für den alltäglichen Einsatz sind oder nicht den gewünschten Fokus auf eine einfache, lokal betriebene Lösung legen. Daraus entstand der Bedarf nach einer schlanken Anwendung, die sich auf die wesentlichen Funktionen der Aufgabenverwaltung konzentriert und ohne externe Abhängigkeiten auskommt.

1.3 Ist-Zustand

Im Ist-Zustand werden Aufgaben und Fälligkeiten von den Beteiligten auf mehreren, voneinander unabhängigen Plattformen verwaltet. Hierzu zählen unter anderem Notizfunktionen auf Endgeräten, Chatverläufe in Messenger-Diensten sowie einfache Tabellenlösungen. Eine zentrale Stelle zur einheitlichen Verwaltung aller Aufgaben existiert nicht. Diese fragmentierte Organisation führt dazu, dass der Überblick über anstehende Aufgaben und Termine erschwert wird. Prioritäten sind nicht eindeutig erkennbar, der Bearbeitungsstatus einzelner Aufgaben ist nicht transparent und es besteht ein erhöhtes Risiko, Fristen zu übersehen. Darüber hinaus entsteht zusätzlicher Aufwand, da Informationen mehrfach gepflegt oder nachträglich zusammengetragen werden müssen. Technisch betrachtet liegt im Ist-Zustand keine einheitliche Anwendung

zur Aufgabenverwaltung vor. Es existiert weder eine strukturierte Datenhaltung noch eine Möglichkeit zur automatisierten Sortierung, Filterung oder Erinnerung an fällige Aufgaben. Die Organisation erfolgt überwiegend manuell und ohne technische Unterstützung durch eine speziell dafür entwickelte Anwendung.

1.4 Projektziel

Ziel des Projekts ist die Entwicklung einer lokal betriebenen Webanwendung zur zentralen Verwaltung von Aufgaben und Fälligkeiten. Die Anwendung soll es ermöglichen, Aufgaben übersichtlich anzulegen, zu bearbeiten, zu löschen sowie nach relevanten Kriterien wie Status, Priorität und Fälligkeitsdatum zu strukturieren. Durch den Einsatz eines Backends auf Basis von C++ wird eine stabile Geschäftslogik sowie eine persistente, lokale Datenhaltung realisiert. Die Speicherung der Aufgaben erfolgt in einer lokalen Datenbank in Form strukturierter JSON-Dateien. Damit wird sichergestellt, dass alle Daten unabhängig von externen Diensten dauerhaft verfügbar sind. Das Frontend wird mit HTML, CSS und JavaScript umgesetzt und stellt eine übersichtliche, klar strukturierte Benutzeroberfläche für Desktop- und Laptop-Browser bereit. Eine feste Fenstergröße sowie der Ausschluss mobiler Endgeräte gewährleisten eine konsistente Darstellung und einfache Bedienbarkeit.

1.5 Meilensteine

Zur strukturierten Umsetzung des Projekts wurde die Arbeit in mehrere Meilensteine unterteilt. Diese orientieren sich an den einzelnen Projektphasen und ermöglichen eine klare zeitliche sowie inhaltliche Abgrenzung der Arbeitsschritte. Der erste Meilenstein umfasst die **Analysephase**, in der die fachlichen und technischen Anforderungen an die Anwendung ermittelt und dokumentiert wurden. Auf Basis der Ausgangssituation wurden die notwendigen Funktionen sowie die technischen Rahmenbedingungen definiert. Im zweiten Meilenstein erfolgte die **Entwurfsphase**. Hierbei wurde ein Soll-Konzept entwickelt, die System- und Softwarearchitektur festgelegt sowie das grundlegende Datenmodell für die Aufgabenverwaltung entworfen. Zudem wurden das Kommunikationskonzept zwischen Frontend und Backend sowie erste Überlegungen zur Benutzeroberfläche erarbeitet. Der dritte Meilenstein bildet die **Implementierungsphase** (Anhang: Abbildung 2). In dieser Phase wurden das Backend in C++ inklusive Socket-Server, Aufgabenlogik und persistenter Datenhaltung umgesetzt. Parallel dazu wurde das Frontend mit React, Vite und Typescript entwickelt und die Kommunikation zwischen den Komponenten realisiert. Der vierte Meilenstein beinhaltet die **Abnahme- und Einführungsphase** (Anhang: Abbildung 3, 4). Hier wurden Funktions- und Integrationstests durchgeführt, die Persistenz der Daten überprüft sowie die Stabilität der Socket-Verbindung getestet. Der abschließende Meilenstein umfasst die **Dokumentationsphase**, in der ein Benutzerhandbuch (Anhang: Runbook), eine technische Dokumentation sowie der Projektbericht erstellt wurden.

1.6 Projektzeitraum

Der Projektzeitraum erstreckte sich über 23.11.2025 - 04.02.2026 mit insgesamt 68 Stunden. Die zeitliche Planung orientierte sich an den Projektphasen Analyse, Entwurf, Implementierung, Abnahme und Dokumentation. Die detaillierte zeitliche Aufteilung der einzelnen Phasen ist tabellarisch in Kapitel 1.9 dargestellt.

1.7 Projektabgrenzung

Im Rahmen des Projekts wurde der Funktionsumfang bewusst auf die Kernfunktionen einer Aufgabenverwaltungsanwendung begrenzt. Ziel war es, eine übersichtliche und stabile Lösung zu entwickeln, ohne den zeitlichen Rahmen des Projekts zu überschreiten. Nicht Bestandteil des Projekts ist eine komplexe Nutzer- und Rechteverwaltung. Die Anwendung ist für eine einfache, lokale Nutzung konzipiert und richtet sich nicht an eine größere Anzahl von gleichzeitig arbeitenden Nutzern. Ebenso wurden keine externen Dienste oder Cloud-basierte Systeme in die Lösung integriert. Erinnerungen und Auswertungen erfolgen ausschließlich innerhalb der Anwendung. Darüber hinaus ist die Anwendung ausschließlich für Desktop- und Laptop-Browser ausgelegt. Die Nutzung auf mobilen Endgeräten wie Smartphones oder Tablets ist nicht vorgesehen und wird technisch unterbunden. Eine Anpassung an unterschiedliche Bildschirmgrößen (Responsive Design) ist nicht Bestandteil des Projekts.

1.8 Qualitätssicherungsmaßnahmen

Zur Sicherstellung der Qualität der entwickelten Anwendung wurden während des gesamten Projekts verschiedene Qualitätssicherungsmaßnahmen eingesetzt. Ziel war es, sowohl die fachliche Korrektheit als auch die technische Stabilität der Lösung zu gewährleisten. Während der Implementierungsphase erfolgte eine kontinuierliche Überprüfung der entwickelten Funktionen. Neue oder geänderte Programmteile wurden direkt getestet, um Fehler frühzeitig zu erkennen und zu beheben. Der Fokus lag dabei insbesondere auf der Aufgabenlogik sowie der Kommunikation zwischen Frontend und Backend. Zur Nachvollziehbarkeit von Änderungen und zur Vermeidung von Versionskonflikten wurde ein Versionsverwaltungssystem auf Basis von GitHub eingesetzt. Dadurch konnten Entwicklungsstände dokumentiert und Änderungen bei Bedarf rückverfolgt werden. In der Abnahmephase wurden gezielte Funktions- und Integrationstests durchgeführt. Hierbei wurde überprüft, ob alle definierten Anforderungen erfüllt sind, die Daten korrekt gespeichert und geladen werden sowie die Socket-Verbindung stabil funktioniert. Ergänzend dazu wurde eine verständliche Dokumentation erstellt, die sowohl die Nutzung der Anwendung als auch deren technischen Aufbau beschreibt. Diese unterstützt die Qualität der Lösung im Hinblick auf Wartbarkeit und Nachvollziehbarkeit.

1.9 Budgetplanung

Für das Projekt lagen keine realen Kosten- oder Gehaltsdaten vor. Daher basiert die Budgetplanung auf realistischen, marktüblichen Stundensätzen aus dem IT-Bereich, wie sie typischerweise in Softwareentwicklungs- und Integrationsprojekten verwendet werden. Diese Annahmen ermöglichen eine wirtschaftliche Bewertung, ohne personenbezogene Daten zu nutzen. Da es sich um ein schulisches Projekt handelt und aus Datenschutzgründen (DSGVO) sowie mangels betrieblicher Abrechnungsdaten keine echten Personalkosten vorlagen, war die Verwendung solcher Annahmen erforderlich. Die Kalkulation berücksichtigte verschiedene Rollen der Anwendungsentwicklung – etwa Analyse, Implementierung, Test und Dokumentation – und bewertete sie mit einem einheitlichen Stundensatz, um Vergleichbarkeit und Transparenz zu gewährleisten. Der größte Kostenanteil entfiel erwartungsgemäß auf die Entwicklungs- und Dokumentationsphase, was der typischen Kostenstruktur softwaretechnischer Projekte entspricht. Darüber hinaus wurden ausschließlich Personalkosten einbezogen. Sachkosten wie Hardware, Endgeräte, Lizenzen oder Infrastruktur wurden nicht berücksichtigt, da diese bereits vorhanden waren und keine zusätzlichen Aufwände verursachten. Durch die klare Rollenverteilung, die Nutzung bestehender Ressourcen und den Verzicht auf externe Dienstleister konnte das Projekt wirtschaftlich und realitätsnah geplant werden. Die kalkulierten Annahmen bieten somit eine fundierte Grundlage für die betriebswirtschaftliche Betrachtung, ohne gegen Datenschutzvorgaben oder interne Richtlinien zu verstoßen.

Projektphase	Beteiligte Personen	Aufwand (Stunden)	Stundensatz (Annahme)	Kosten (Annahme)
Analyse	Alle	2 h	35 €	70 €
Entwurf	Alle	4 h	35 €	140 €
Implementierung Backend	Jake Pettke, Philip Adam	18 h	40 €	720 €
Implementierung Frontend	Leo Hepting, Alma Kosuta	18 h	35 €	630 €
Abnahme & Einführung	Alle	6 h	35 €	210 €

Dokumentation	Bea Janott, Mikel Thiele	20 h	35 €	700 €
Gesamt		**68 h**		**2.470 €**

Legende: Alle = Alle am Projekt beteiligten Entwickler

2. Analysephase

2.1 Anforderungsanalyse

Ausgangssituation:

Ist:

- fehlender zentraler Überblick über offene Aufgaben
- potenzielle Doppelarbeit
- Aufgaben und Fristen werden übersehen
- Tools sind teilweise überladen und für kleine Teams zu komplex
- keine einheitliche Struktur für Prioritäten, Status oder Deadlines

Nutzeranforderungen:

- zentrale Verwaltung von Aufgaben
- Eingabe von: Titel, Beschreibung, Status, Priorität, Deadline
- einfache Sortier- und Filterfunktionen
- Erinnerungsfunktion vor Fälligkeiten
- lokale Speicherung ohne Cloud
- schnelle Bedienbarkeit und intuitive Oberfläche

Technische Ausgangslage:

- lokale Anwendung, kein Webserver
- Browserbasierte Bedienung (Desktop)
- Speicherung lokal

Ziel der Anforderungsanalyse war es, die fachlichen und technischen Anforderungen an die zu entwickelnde Anwendung systematisch zu ermitteln und abzugrenzen. Grundlage hierfür bildeten die in der Projektbeschreibung dargestellte Ausgangssituation sowie die definierten Projektziele. Die Anforderungen wurden innerhalb der Projektgruppe gemeinsam erarbeitet. Dabei wurden typische Nutzungsszenarien betrachtet, wie das Anlegen, Bearbeiten und Verwalten von Aufgaben sowie der Umgang mit Fälligkeiten und Prioritäten. Die Abstimmung erfolgte überwiegend mündlich und wurde anschließend in strukturierter Form festgehalten. Im Rahmen der Analyse wurden sowohl funktionale als auch nicht-funktionale Anforderungen identifiziert. Funktionale Anforderungen beschreiben die konkreten Funktionen der Anwendung, während nicht-funktionale Anforderungen technische Rahmenbedingungen, Einschränkungen und Qualitätsmerkmale festlegen. Die Ergebnisse der Anforderungsanalyse dienten als Grundlage für die anschließende Entwurfsphase. Durch die klare Definition der Anforderungen konnte sichergestellt werden, dass der geplante Funktionsumfang realistisch umsetzbar ist und innerhalb des vorgesehenen Zeitrahmens bleibt.

2.2 Fachliche Anforderungen

Die fachlichen Anforderungen definieren die Funktionen, die die Anwendung aus Anwendersicht bereitstellen muss. Ziel ist es, eine übersichtliche und einfach bedienbare Aufgabenverwaltung bereitzustellen, die den Nutzer bei der Organisation von Aufgaben und Fälligkeiten unterstützt. Zentrale fachliche Anforderung ist die Möglichkeit, Aufgaben anzulegen, zu bearbeiten und zu löschen. Jede Aufgabe muss mindestens einen **Titel**, ein **Fälligkeitsdatum**, eine **Priorität** sowie einen **Status** besitzen. Der Status soll den Bearbeitungsstand einer Aufgabe abbilden, beispielsweise „offen“ oder „erledigt“. Darüber hinaus muss die Anwendung eine strukturierte Darstellung der Aufgaben ermöglichen. Aufgaben sollen nach verschiedenen Kriterien wie Fälligkeitsdatum, Priorität oder Status sortiert und gefiltert werden können, um einen schnellen Überblick zu gewährleisten. Zusätzlich wird eine persistente Speicherung der Aufgaben gefordert, sodass angelegte Aufgaben auch nach einem Neustart der Anwendung weiterhin verfügbar sind. Die Bedienung der Anwendung soll dabei intuitiv erfolgen und keine umfangreiche Einarbeitung erfordern.

2.3 Technische Anforderungen

Die technischen Anforderungen legen die Rahmenbedingungen für die Umsetzung der Anwendung fest und definieren den eingesetzten Technologie-Stack sowie grundlegende Systemvorgaben. Ziel war es, eine schlanke und lokal betriebene Lösung ohne externe Abhängigkeiten zu realisieren. Das Backend der Anwendung wird in der Programmiersprache C++ umgesetzt. Es übernimmt die Geschäftslogik der Aufgabenverwaltung sowie die persistente Speicherung der Daten. Die Datenhaltung erfolgt lokal in Form strukturierter JSON-Dateien, die als einfache Datenbank dienen. Die Kommunikation zwischen Frontend und Backend erfolgt über eine Socket-basierte Schnittstelle. Der Datenaustausch wird in einem definierten JSON-Format realisiert, um eine klare und nachvollziehbare Kommunikation zwischen den Komponenten zu gewährleisten. Das Frontend wird mit HTML, CSS und JavaScript entwickelt und ist ausschließlich für die Nutzung in Desktop- und Laptop-Browsern vorgesehen. Die Anwendung ist nicht responsiv, die Fenstergröße ist fest vorgegeben und darf nicht verändert werden, da eine Anpassung an unterschiedliche Bildschirmgrößen nicht Bestandteil des Projekts ist. Weitere technische Anforderungen betreffen die Wartbarkeit und Nachvollziehbarkeit des Codes. Der Quellcode wird versioniert in einem GitHub-Repository verwaltet. Fehlerbehandlung und grundlegende Tests der Kernfunktionen sind Bestandteil der Umsetzung.

2.4 Wirtschaftliche und zeitliche Betrachtung

Im Rahmen der Analysephase wurde das Projekt sowohl wirtschaftlich als auch zeitlich betrachtet, um sicherzustellen, dass die Umsetzung innerhalb des vorgegebenen

Rahmens realisierbar ist. Grundlage hierfür bildete die geplante Gesamtprojektzeit von 68 Stunden sowie die definierte Aufgabenverteilung innerhalb der Projektgruppe. Aus wirtschaftlicher Sicht wurde das Projekt bewusst schlank gehalten. Durch den Einsatz freier und weit verbreiteter Technologien wie C++, HTML, CSS und JavaScript entstehen keine zusätzlichen Lizenzkosten. Die Nutzung eines lokalen Systems ohne externe Dienste reduziert potenzielle laufende Kosten und Abhängigkeiten. Die wesentlichen Aufwände entstehen durch den personellen Einsatz während der Analyse-, Entwurfs-, Implementierungs- und Dokumentationsphase. Eine detaillierte Kostenbetrachtung auf Basis angenommener Stundensätze wurde im Kapitel 1.9 Budgetplanung dargestellt. Diese verdeutlicht, dass der größte wirtschaftliche Aufwand in der Implementierungsphase liegt. Die zeitliche Planung wurde so gewählt, dass der Schwerpunkt auf der Implementierung der Kernfunktionen liegt, ohne die notwendigen Analyse-, Entwurfs- und Testphasen zu vernachlässigen. Durch die klare Strukturierung der Projektphasen konnte ein realistischer und umsetzbarer Zeitplan erstellt werden, der eine termingerechte Fertigstellung des Projekts ermöglicht.

3. Entwurfsphase

3.1 Soll-Konzept

Entwurfsphase:

Architekturentwurf
Die Anwendung wird in zwei logische Bereiche unterteilt:
Backend (C++)
- verwaltet Aufgaben und speichert sie lokal in JSON
- stellt eine Geschäftslogik bereit
- bietet eine Socket-Schnittstelle für CRUD-Operationen (Create/Read/Update/Delete)
- kümmert sich um Erinnerungslogiken und Datenvalidierungen

Frontend (Browser)
- visuelle Darstellung der Aufgaben
- Eingabeformulare für neue Tasks
- Filter- und Sortiermöglichkeiten
- feste Fenstergröße (Layout nicht responsiv)

Das Soll-Konzept beschreibt den angestrebten Zielzustand der zu entwickelnde Anwendung und dient als Grundlage für die anschließende Implementierung. Ziel ist eine lokal betriebene Webanwendung zur zentralen Verwaltung von Aufgaben und Fälligkeiten mit klarer Trennung zwischen Benutzeroberfläche und Geschäftslogik. Die Anwendung ist als Client-Server-Lösung konzipiert. Das Frontend stellt die Benutzeroberfläche bereit und ermöglicht die Interaktion mit dem Nutzer, während das Backend die Verarbeitung der Anfragen, die Aufgabenlogik sowie die persistente

Speicherung der Daten übernimmt. Die Aufgabenverwaltung soll es dem Nutzer ermöglichen, Aufgaben mit definierten Attributen wie Titel, Fälligkeitsdatum, Priorität und Status zu erfassen und zu verwalten. Die Darstellung der Aufgaben erfolgt strukturiert und übersichtlich, sodass ein schneller Überblick über anstehende und erledigte Aufgaben gewährleistet ist. Die Datenhaltung erfolgt lokal in Form strukturierter JSON-Dateien. Diese dienen als einfache Datenbank und stellen sicher, dass alle Aufgaben auch nach einem Neustart der Anwendung weiterhin verfügbar sind. Externe Datenbanken oder Cloud-Dienste sind nicht Bestandteil des Soll-Konzepts. Das Soll-Konzept berücksichtigt bewusst die im Projekt definierten Einschränkungen. Die Anwendung ist ausschließlich für Desktop- und Laptop-Browser vorgesehen, nutzt eine feste Fenstergröße und verzichtet auf ein responsives Design. Dadurch kann die Benutzeroberfläche klar und einfach gestaltet werden und der Fokus liegt auf der funktionalen Umsetzung der Kernanforderungen.

3.2 System- und Softwarearchitektur

- Client–Server
- Frontend ↔ Backend
- JSON
- Socket

Eine detaillierte technische Beschreibung der Architektur befindet sich in Kapitel 7.2

3.3 Datenmodell der Aufgaben

Das Datenmodell bildet die Grundlage für die strukturierte Verwaltung der Aufgaben innerhalb der Anwendung. Ziel ist es, alle für die Aufgabenorganisation relevanten Informationen in einer klar definierten Struktur abzubilden. Eine Aufgabe wird als eigenständiges Objekt modelliert und enthält mehrere Attribute. Zu den zentralen Eigenschaften gehören ein Titel zur eindeutigen Benennung der Aufgabe, eine Beschreibung zur näheren Erläuterung sowie ein Fälligkeitsdatum, das den zeitlichen Bezug herstellt. Zusätzlich verfügt jede Aufgabe über eine Priorität, mit der die Wichtigkeit der Aufgabe festgelegt wird, sowie über einen Status, der den aktuellen Bearbeitungsstand widerspiegelt. Typische Statuswerte sind beispielsweise „offen“ und „erledigt“. Zur eindeutigen Identifikation einer Aufgabe wird eine interne ID vergeben. Diese ermöglicht es, Aufgaben eindeutig zu referenzieren, zu bearbeiten oder zu löschen, auch wenn mehrere Aufgaben ähnliche Inhalte besitzen. Die Aufgabenobjekte werden in strukturierter Form in einer JSON-Datei gespeichert. Das gewählte Datenmodell ist bewusst einfach gehalten und ermöglicht eine leichte Erweiterung, beispielsweise durch

zusätzliche Attribute, ohne die bestehende Struktur grundlegend zu verändern (Anhang: Abbildung 1).

3.4 Schnittstellen- und Kommunikationskonzept (JSON/ Socket)

Für die Kommunikation zwischen Frontend und Backend wurde ein klar definiertes Schnittstellen- und Kommunikationskonzept festgelegt. Ziel ist ein zuverlässiger und strukturierter Datenaustausch zwischen den beiden Komponenten der Anwendung. Die Kommunikation erfolgt über eine Socket-basierte Verbindung im lokalen Umfeld. Das Frontend fungiert dabei als Client, der Anfragen an das Backend sendet, während das Backend als Server die Anfragen verarbeitet und entsprechende Antworten zurückliefert. Der Datenaustausch erfolgt in einem einheitlichen JSON-Format. Jede Anfrage enthält eine eindeutige Aktionskennung, die beschreibt, welche Operation ausgeführt werden soll, beispielsweise das Anlegen oder Löschen einer Aufgabe. Zusätzlich werden die zur Verarbeitung benötigten Daten als strukturierte JSON-Objekte übertragen. Das Backend wertet die empfangenen Nachrichten aus, führt die entsprechende Geschäftslogik aus und sendet das Ergebnis in Form einer JSON-Antwort an das Frontend zurück. Diese Antwort enthält Informationen über den Erfolg oder Misserfolg der Anfrage sowie gegebenenfalls die angeforderten Daten. Durch die Verwendung von JSON als Austauschformat ist die Schnittstelle klar strukturiert, leicht verständlich und unabhängig von der jeweiligen Implementierung der beteiligten Komponenten. Dies erleichtert sowohl die Wartung als auch mögliche Erweiterungen der Anwendung. (Anhang: Abbildung 7)

3.5 UI- und Bedienkonzept

Das UI- und Bedienkonzept der Anwendung verfolgt das Ziel, eine übersichtliche und einfach verständliche Benutzeroberfläche bereitzustellen. Der Fokus liegt auf einer klaren Struktur und einer intuitiven Bedienung, sodass der Nutzer ohne lange Einarbeitung mit der Anwendung arbeiten kann. Die Benutzeroberfläche ist in logisch getrennte Bereiche gegliedert. Ein zentraler Bereich dient der Anzeige der vorhandenen Aufgaben, während Eingabemasken für das Anlegen und Bearbeiten von Aufgaben vorgesehen sind. Durch diese klare Aufteilung bleibt der Überblick auch bei einer größeren Anzahl von Aufgaben erhalten. Die Bedienung erfolgt über klassische Bedienelemente wie Eingabefelder, Dropdown-Menüs und Schaltflächen. Aktionen wie das Hinzufügen oder Liste löschen von Aufgaben sind eindeutig gekennzeichnet und direkt aus der Aufgabenübersicht heraus möglich. Durch die bewusste Reduktion auf wesentliche Bedienelemente und eine konsistente Gestaltung wird sichergestellt, dass die Anwendung auch bei längerer Nutzung übersichtlich und gut bedienbar bleibt.

4. Implementierungsphase

4.1 Einrichtung der Entwicklungsumgebung

Zu Beginn der Implementierungsphase wurde die Entwicklungsumgebung eingerichtet, um eine reibungslose Umsetzung der Anwendung zu ermöglichen. Ziel war es, eine einheitliche und stabile Arbeitsumgebung für alle Projektbeteiligten zu schaffen. Für die Entwicklung des Backends wurde eine geeignete Entwicklungsumgebung für C++ eingesetzt, die das Schreiben, Kompilieren und Debuggen des Quellcodes unterstützt. Für das Frontend kamen gängige Werkzeuge zur Bearbeitung von HTML-, CSS- und JavaScript-Dateien zum Einsatz. Die Ausführung und das Testen des Frontends erfolgten in aktuellen Desktop-Browsern (Google Chrome, Microsoft Edge). Zur Zusammenarbeit innerhalb der Projektgruppe wurde ein gemeinsames GitHub-Repository eingerichtet. Dieses diente der zentralen Ablage des Quellcodes sowie der Nachverfolgung von Änderungen. Durch die Nutzung der Versionsverwaltung konnten parallele Arbeiten an verschiedenen Komponenten koordiniert und Konflikte reduziert werden. Zusätzlich wurden grundlegende Projektstrukturen festgelegt, beispielsweise die Trennung von Backend- und Frontend-Verzeichnissen. Dadurch wurde eine klare Ordnung des Quellcodes erreicht und die spätere Wartung der Anwendung erleichtert. Nach Abschluss der Einrichtung konnte die eigentliche Implementierung der Anwendung beginnen.

4.2 Implementierung des Backends (C++)

4.21 Socket-Server

Der Socket-Server bildet die Grundlage für die Kommunikation zwischen Frontend und Backend. Er wurde in C++ implementiert und läuft lokal auf dem System des Nutzers. Der Server lauscht auf einem definierten Port und nimmt eingehende Verbindungen des Frontends entgegen. Nach dem Verbindungsaufbau werden die vom Frontend gesendeten Anfragen empfangen, verarbeitet und entsprechende Antworten zurückgesendet. Die Verarbeitung der Anfragen erfolgt synchron. Jede eingehende Nachricht wird vollständig ausgewertet, bevor eine Antwort generiert wird. Dadurch wird eine einfache und stabile Kommunikation gewährleistet, die für den vorgesehenen Projektumfang ausreichend ist. Der Socket-Server ist so aufgebaut, dass er wiederverwendbar und erweiterbar bleibt. Fehlerhafte oder unvollständige Anfragen werden erkannt und mit einer entsprechenden Fehlermeldung beantwortet.

4.22 Aufgabenlogik

Die Aufgabenlogik stellt den zentralen funktionalen Kern des Backends dar. Sie ist dafür verantwortlich, die vom Frontend angeforderten Aktionen fachlich korrekt umzusetzen. Zu den implementierten Funktionen zählen das Anlegen und Beenden von Aufgaben. Zusätzlich wurde eine Logik zur Sortierung und Filterung der Aufgaben nach Kriterien wie

Fälligkeitsdatum, Priorität und Status umgesetzt. Bei der Verarbeitung von Aufgaben werden Plausibilitätsprüfungen durchgeführt. So wird beispielsweise sichergestellt, dass Pflichtfelder befüllt sind und Fälligkeitsdaten ein gültiges Format besitzen. Die Aufgabenlogik ist klar vom Socket-Server getrennt implementiert. Dadurch bleibt die Geschäftslogik unabhängig von der Art der Kommunikation und kann bei Bedarf leichter angepasst oder erweitert werden.

4.23 Persistente Speicherung (JSON)

Zur dauerhaften Speicherung der Aufgaben wurde eine persistente Datenhaltung auf Basis von JSON-Dateien implementiert. Diese dienen als einfache lokale Datenbank und werden vom Backend verwaltet. Beim Start der Anwendung werden vorhandene Aufgaben aus der JSON-Datei eingelesen und in interne Datenstrukturen überführt. Änderungen an Aufgaben, wie das Hinzufügen, Bearbeiten oder Löschen, werden nach der jeweiligen Aktion wieder in der Datei gespeichert. Durch dieses Vorgehen ist sichergestellt, dass der aktuelle Stand der Aufgaben auch nach einem Neustart der Anwendung erhalten bleibt. Die Wahl von JSON als Speicherformat ermöglicht eine gut lesbare und strukturierte Ablage der Daten. Zudem kann die Datenstruktur bei Bedarf erweitert werden, ohne bestehende Daten zu verlieren.

4.3 Implementierung des Frontends (React, ShadCn, Typescript)

4.31 Layout und feste Fenstergröße

Das Layout der Anwendung wurde mit React umgesetzt und ist auf eine feste Fenstergröße ausgelegt. Ziel war es, eine konsistente und übersichtliche Darstellung unabhängig von der Bildschirmauflösung zu gewährleisten. Die Struktur der Benutzeroberfläche basiert auf klar voneinander getrennten Bereichen. Diese umfassen unter anderem einen Bereich zur Anzeige der Aufgaben sowie separate Bereiche für Eingaben und Aktionen. Die feste Fenstergröße wurde bewusst gewählt und technisch abgesichert, um Layoutverschiebungen und Darstellungsprobleme zu vermeiden. Eine Anpassung an unterschiedliche Bildschirmgrößen oder ein responsives Design sind nicht Bestandteil der Umsetzung. Durch die klare Layoutstruktur und die feste Fenstergröße bleibt die Benutzeroberfläche jederzeit übersichtlich und leicht bedienbar.

4.32 Eingabemasken

Für das Anlegen und Bearbeiten von Aufgaben wurden strukturierte Eingabemasken implementiert. Diese ermöglichen dem Nutzer, alle relevanten Informationen zu einer Aufgabe zu erfassen. Die Eingabemasken enthalten Felder für Titel, Fälligkeitsdatum, Priorität und Status. Die Eingaben werden vor dem Absenden geprüft, um fehlerhafte oder unvollständige Daten zu vermeiden. Die Bedienung der Eingabemasken erfolgt

intuitiv über klassische Formularelemente. Die erfassten Daten werden nach der Eingabe an das Backend übermittelt und dort weiterverarbeitet.

4.33 Aufgabenübersicht

Die Aufgabenübersicht stellt den zentralen Bestandteil der Benutzeroberfläche dar. Sie zeigt alle vorhandenen Aufgaben in strukturierter Form an und ermöglicht dem Nutzer einen schnellen Überblick über den aktuellen Aufgabenbestand. Die Darstellung der Aufgaben erfolgt übersichtlich und nachvollziehbar. Wichtige Informationen wie Titel, Fälligkeitsdatum, Priorität und Status sind auf einen Blick erkennbar. Zusätzlich stehen Funktionen zur Sortierung und Filterung der Aufgaben zur Verfügung, um gezielt nach bestimmten Kriterien zu suchen. Durch die zentrale Aufgabenübersicht wird eine effiziente Verwaltung der Aufgaben unterstützt und die Nutzung der Anwendung erleichtert.

4.4 Fehlerbehandlung und Debugging

Während der Implementierung wurde auf eine stabile und strukturierte Fehlerbehandlung geachtet. Im Backend wurden ungültige Anfragen, fehlerhafte JSON-Daten und Probleme beim Zugriff auf die Persistenzdatei erkannt und mit passenden Rückmeldungen an das Frontend beantwortet. Das Frontend prüft Eingaben bereits vor dem Senden, um unvollständige oder fehlerhafte Daten abzufangen. Fehlermeldungen werden dem Nutzer verständlich und ohne technische Details angezeigt. Zur Fehleranalyse kamen Debugging-Werkzeuge sowie gezielte Funktionstests zum Einsatz. Die Kombination aus präventiver Validierung, klarer Fehlerkommunikation und kontinuierlichem Debugging sorgte für eine stabile und zuverlässige Anwendung.

5. Abnahme- und Einführungsphase

5.1 Funktionstests

Es wurden alle fachlichen Kernfunktionen getestet, darunter das Anlegen, Löschen sowie Anzeigen von Aufgaben. Ebenso wurden Sortier- und Filterfunktionen geprüft. Die Tests erfolgten anhand typischer Nutzungsszenarien und wurden mehrfach ausgeführt, um ein konsistentes Verhalten sicherzustellen. Gefundene Fehler wurden dokumentiert und behoben.

5.2 Integrationstests

Die Integrationstests überprüften das Zusammenspiel zwischen Frontend und Backend. Getestet wurde, ob Anfragen korrekt übertragen, verarbeitet und im Interface dargestellt werden. Die Tests bestätigten eine zuverlässige Kommunikation und ein fehlerfreies Zusammenspiel der Komponenten.

5.3 Tests der Datenpersistenz

Zur Prüfung der dauerhaften Datenspeicherung wurden Aufgaben angelegt und gelöscht. Anschließend wurde die Anwendung neu gestartet, um die Korrektheit des gespeicherten Zustands zu überprüfen. Die Tests bestätigten, dass die JSON-Speicherung zuverlässig funktioniert und keine Daten verloren gehen.

5.4 Stabilitätsprüfung der Socket-Verbindung

Zur Sicherstellung eines stabilen Betriebs wurde die Socket-Verbindung zwischen Frontend und Backend gezielt überprüft. Dabei wurde getestet, wie sich die Anwendung bei wiederholten Anfragen und längerer Laufzeit verhält. Es wurde überprüft, ob Verbindungen zuverlässig aufgebaut und korrekt geschlossen werden. Zudem wurde getestet, ob die Anwendung bei kurzzeitigen Verbindungsunterbrechungen stabil reagiert und keine undefinierten Zustände entstehen. Die Stabilitätsprüfung zeigte, dass die Socket-Kommunikation im vorgesehenen Nutzungsszenario zuverlässig funktioniert und für den Projektumfang ausreichend stabil ist.

6. Überführung

6.1 Projektvorstellung und Review

Nach Abschluss der Implementierungs- und Testphase wurde das Projekt im Rahmen der Projektgruppe vorgestellt. Ziel der Projektvorstellung war es, die entwickelte Anwendung sowie deren Funktionsumfang zu präsentieren und die Ergebnisse gemeinsam zu bewerten. Im Rahmen der Vorstellung wurde die Anwendung demonstriert und die wichtigsten Funktionen, wie die Aufgabenverwaltung, die Sortier- und Filtermöglichkeiten sowie die persistente Speicherung, erläutert. Anschließend erfolgte ein Review des Projekts. Dabei wurde überprüft, ob die ursprünglich definierten Projektziele erreicht wurden und ob die Anwendung den geplanten Anforderungen entspricht. Insgesamt konnte festgestellt werden, dass die Umsetzung dem geplanten Soll-Konzept entspricht und die Anwendung funktionsfähig ist.

6.2 Soll-Ist-Vergleich

Im Soll-Ist-Vergleich wurden die ursprünglich definierten Projektziele mit dem tatsächlich erreichten Ergebnis gegenübergestellt. Ziel war es, Abweichungen zu identifizieren und zu bewerten. Die wesentlichen funktionalen Anforderungen, wie das Anlegen und Löschen von Aufgaben sowie die strukturierte Darstellung nach Status, Priorität und Fälligkeit, konnten vollständig umgesetzt werden. Auch die lokale, persistente Speicherung der Aufgaben sowie die Kommunikation zwischen Frontend und Backend entsprechen dem geplanten Soll-Zustand. Abweichungen ergaben sich bei der

Implementierung des Frontends, wo von einer HTML, CSS, JS Struktur zu einer React, ShadCn, Typescript Struktur gewechselt wurde. Hierzu gibt es eine Entscheidungsmatrix.

Kriterium	HTML/CSS/JS	React + Vite
Wartbarkeit	✗ schwierig	✓ sehr gut
Komponentenaufbau	✗ nicht vorgesehen	✓ nativ verfügbar
State-Management	✗ manuell	✓ React-State
Entwicklungszeit	mittel	niedrig (wegen Wiederverwendung)
Einarbeitung	✓ sehr leicht	✗ benötigt Wissen
Performance	gut	sehr gut

- **Bessere Komponentenstruktur** → Aufgabenliste, Formulare, Filter etc. als wiederverwendbare Komponenten
- **State-Management** → Beim Aktualisieren von Aufgaben muss nicht die ganze Seite neu gerendert werden
- **Schnellere Entwicklung dank Vite** (Hot Reload, moderne Build-Chain)
- **Klare Trennung von Logik & Darstellung**

Andere Abweichungen traten lediglich in Bereichen, die bewusst aus dem Projektumfang ausgeklammert wurden, beispielsweise eine erweiterte Nutzerverwaltung oder ein responsives Design. Diese Punkte wurden bereits in der Projektabgrenzung definiert und stellen keine negative Abweichung dar. Insgesamt zeigt der Soll-Ist-Vergleich, dass die geplanten Ziele erreicht und die definierten Anforderungen erfüllt wurden.

6.3 Projektschwierigkeiten

Während der Umsetzung des Projekts traten vereinzelt Herausforderungen auf, die im Verlauf bearbeitet und gelöst wurden. Eine zentrale Schwierigkeit bestand in der Abstimmung zwischen Frontend und Backend, insbesondere bei der Definition und Umsetzung des JSON-Kommunikationsformats. Zusätzlich erforderte die Implementierung der Socket-Kommunikation eine sorgfältige Fehlerbehandlung, da Verbindungsabbrüche oder fehlerhafte Datenübertragungen zu unerwartetem Verhalten führen konnten. Durch kontinuierliche Abstimmungen innerhalb der Projektgruppe sowie gezielte Tests konnten diese Schwierigkeiten schrittweise behoben werden. Die gemachten Erfahrungen trugen dazu bei, das Verständnis für verteilte Systeme und die Bedeutung klar definierter Schnittstellen zu vertiefen.

7. Dokumentation

7.1 Benutzerhandbuch

Das Benutzerhandbuch befindet sich im Anhang (Anhang: Runbook).

7.2 Technische Dokumentation

Die Technische Dokumentation befindet sich im Anhang (Anhang: Abbildung 5,, Anhang: Abbildung 6.

7.3 Fazit

Im Rahmen des Projekts wurde eine lokal betriebene Webanwendung zur Verwaltung von Aufgaben und Fälligkeiten erfolgreich konzipiert und umgesetzt. Ausgehend von einer klar definierten Ausgangssituation konnte eine Lösung entwickelt werden, die die zentralen Anforderungen an eine übersichtliche Aufgabenverwaltung erfüllt. Durch die strukturierte Vorgehensweise entlang der Projektphasen Analyse, Entwurf, Implementierung und Abnahme war eine zielgerichtete und nachvollziehbare Umsetzung möglich. Insbesondere die klare Trennung zwischen Frontend und Backend sowie die bewusste Beschränkung auf einen schlanken Technologie-Stack trugen zur Übersichtlichkeit und Stabilität der Anwendung bei. Die im Projekt gewonnenen Erfahrungen, insbesondere im Bereich der Socket-Kommunikation, der JSON-basierten Datenhaltung sowie der Teamarbeit mit Versionsverwaltung, stellten einen wichtigen Lernzuwachs dar. Insgesamt kann das Projekt als erfolgreich bewertet werden, da die definierten Ziele erreicht und die geplanten Anforderungen umgesetzt wurden.

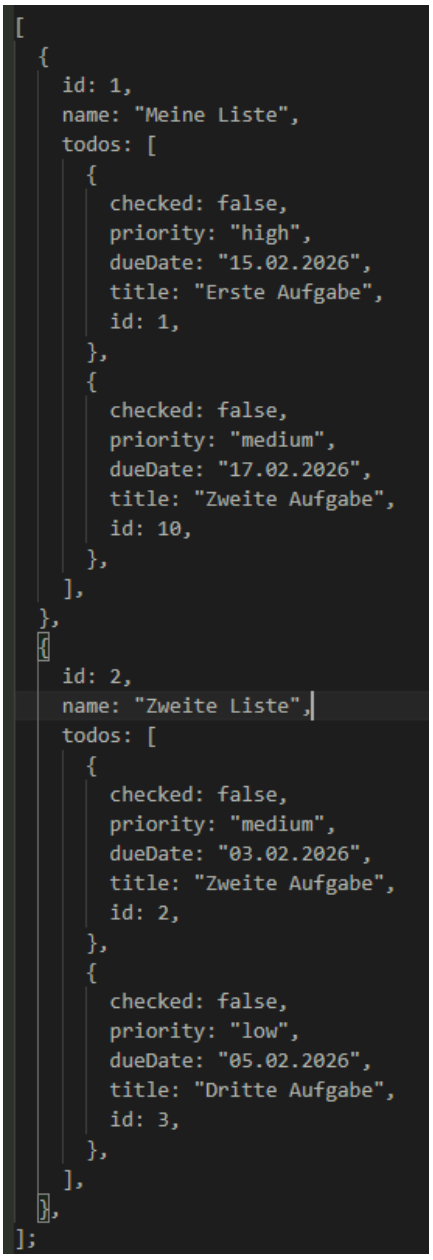
7.4 Zukunftsaussichten

Auch wenn die definierten Projektziele erreicht wurden, bietet die Anwendung weiterhin Potenzial für zukünftige Erweiterungen. Denkbar ist beispielsweise eine erweiterte Nutzerverwaltung, um mehrere Anwender mit getrennten Aufgabenbeständen zu unterstützen. Auch die Benutzeroberfläche ließe sich verbessern – etwa durch ein responsives Design für mobile Endgeräte. Zusätzliche Funktionen wie Erinnerungen, wiederkehrende Aufgaben oder statistische Auswertungen wären ebenfalls möglich. Mit wachsendem Funktionsumfang käme zudem der Einsatz einer echten Datenbank statt der JSON-basierten Speicherung infrage. Die modulare Architektur bildet dabei eine geeignete Grundlage für spätere Erweiterungen.

Anhang

Struktur der Datenspeicherung

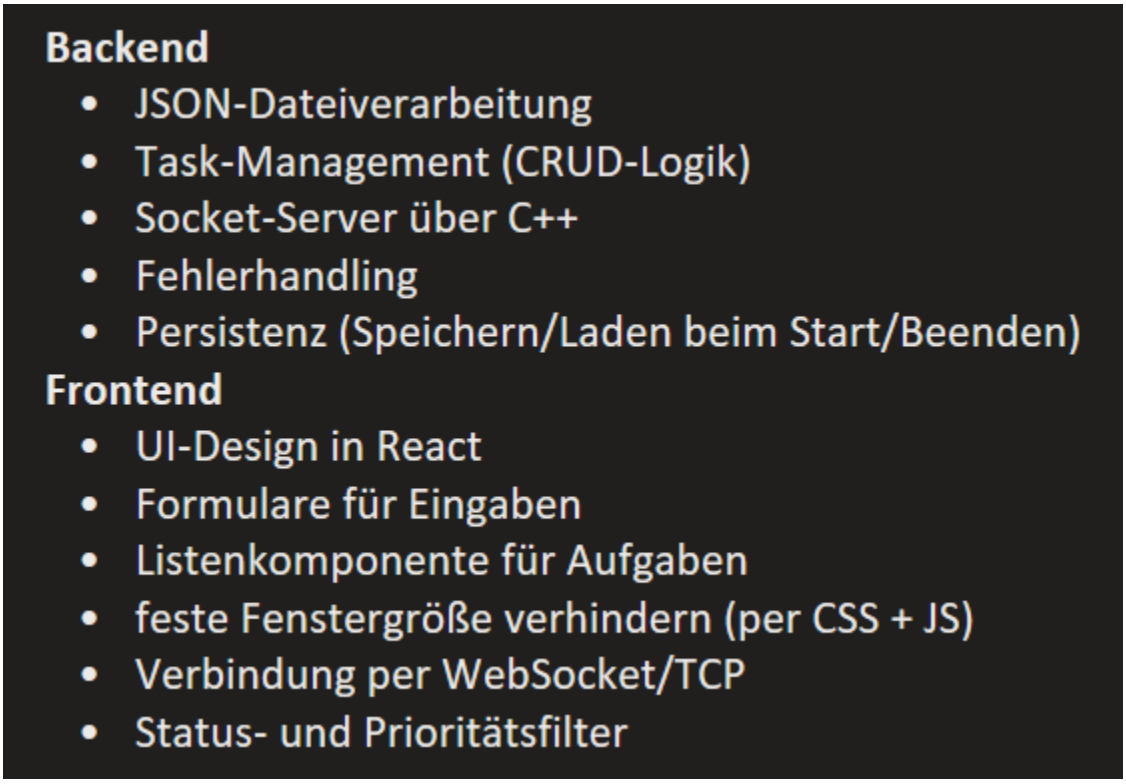
Abbildung 1



```
[
  {
    id: 1,
    name: "Meine Liste",
    todos: [
      {
        checked: false,
        priority: "high",
        dueDate: "15.02.2026",
        title: "Erste Aufgabe",
        id: 1,
      },
      {
        checked: false,
        priority: "medium",
        dueDate: "17.02.2026",
        title: "Zweite Aufgabe",
        id: 10,
      },
    ],
  },
  {
    id: 2,
    name: "Zweite Liste",
    todos: [
      {
        checked: false,
        priority: "medium",
        dueDate: "03.02.2026",
        title: "Zweite Aufgabe",
        id: 2,
      },
      {
        checked: false,
        priority: "low",
        dueDate: "05.02.2026",
        title: "Dritte Aufgabe",
        id: 3,
      },
    ],
  },
];
```

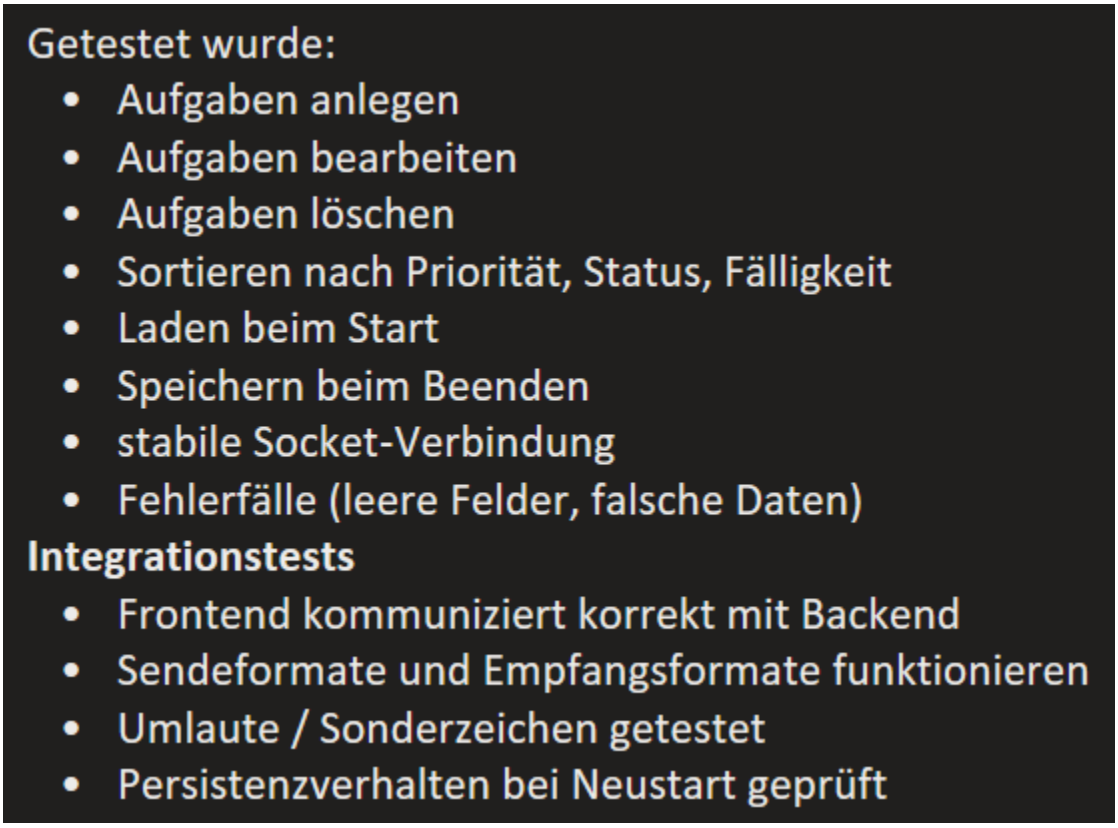
Implementierungsinhalte

Abbildung 2



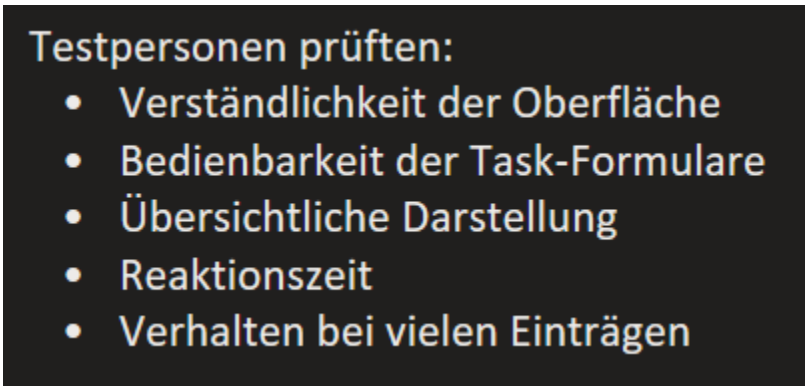
Tests

Abbildung 3



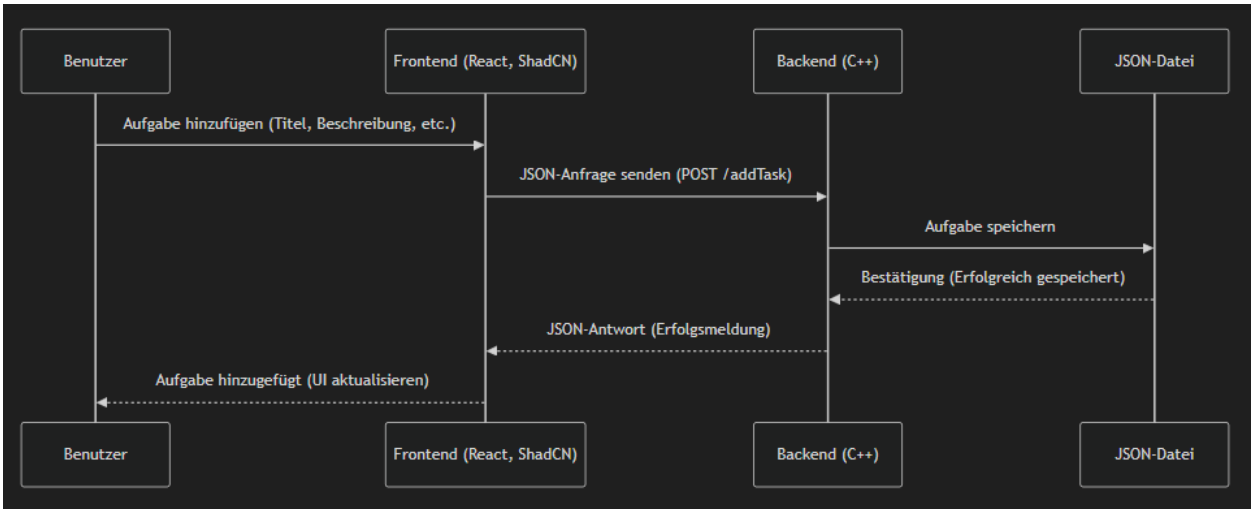
Nutzertests

Abbildung 4



Sequenzdiagramm

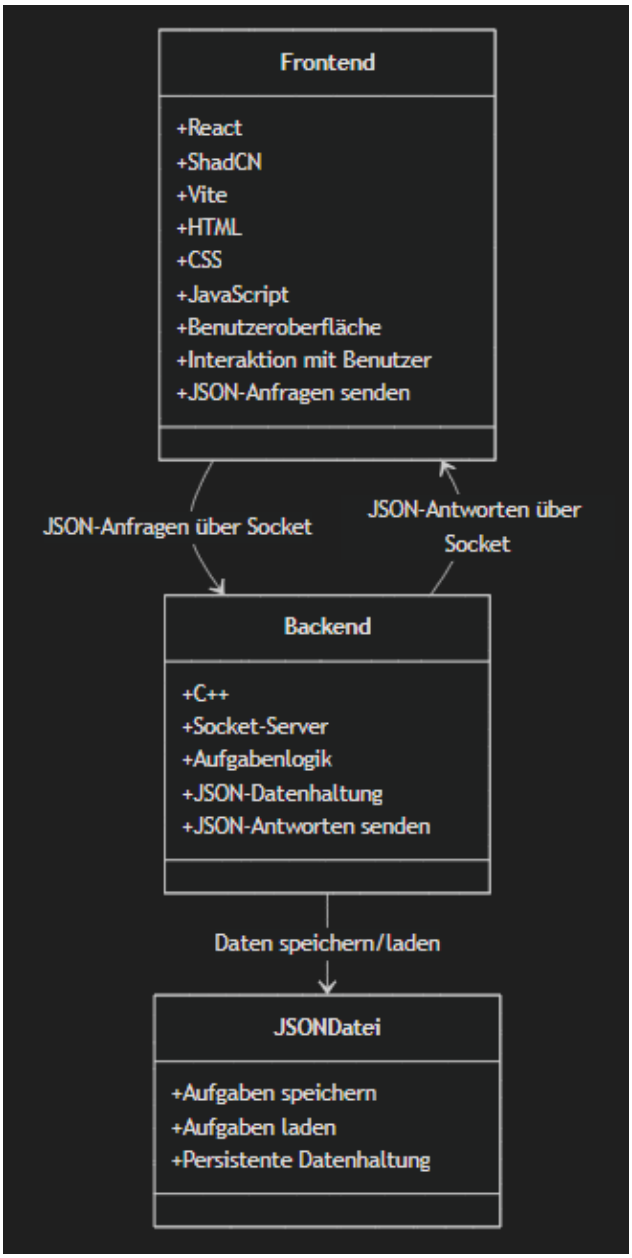
Abbildung 5



Die System- und Softwarearchitektur der Anwendung folgt einem klaren Client-Server-Modell. Ziel dieser Architektur ist eine saubere Trennung zwischen Präsentationsschicht und Geschäftslogik, um die Wartbarkeit und Erweiterbarkeit der Anwendung zu erhöhen. Das Frontend fungiert als Client und wird in HTML, CSS und JavaScript umgesetzt. Es stellt die grafische Benutzeroberfläche bereit, verarbeitet Benutzereingaben und sendet entsprechende Anfragen an das Backend. Die Darstellung ist auf eine feste Fenstergröße ausgelegt und für die Nutzung in Desktop- und Laptop-Browsern konzipiert. Das Backend wird als eigenständige Server-Komponente in C++ realisiert. Es übernimmt die Verarbeitung der vom Frontend empfangenen Anfragen, implementiert die zentrale Aufgabenlogik und steuert die persistente Datenhaltung. Das Backend läuft lokal auf dem System des Nutzers und ist nicht öffentlich erreichbar. Die Kommunikation zwischen Frontend und Backend erfolgt über eine Socket-basierte Schnittstelle. Der Datenaustausch wird in einem definierten JSON-Format durchgeführt. Dadurch ist eine strukturierte, technologieunabhängige und leicht erweiterbare Kommunikation zwischen den Komponenten möglich. Zur Speicherung der Aufgaben wird eine lokale JSON-Datei verwendet, die vom Backend gelesen und geschrieben wird. Diese einfache Datenhaltung ersetzt eine klassische Datenbank und ist für den vorgesehenen Projektumfang ausreichend. Der gesamte Quellcode der Anwendung wird in einem gemeinsamen GitHub-Repository verwaltet. Die Versionsverwaltung ermöglicht eine klare Trennung der Entwicklungsstände sowie eine nachvollziehbare Dokumentation von Änderungen.

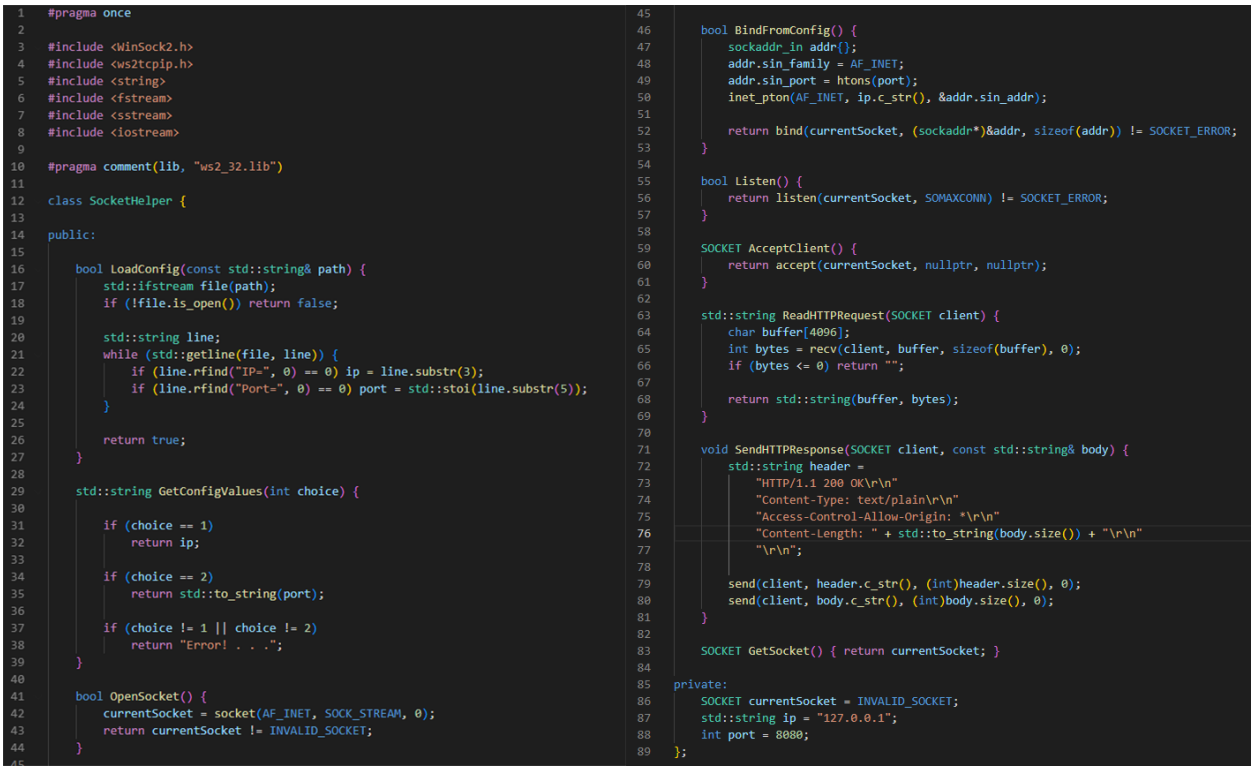
Architektur

Abbildung 6



Aufbau Socket

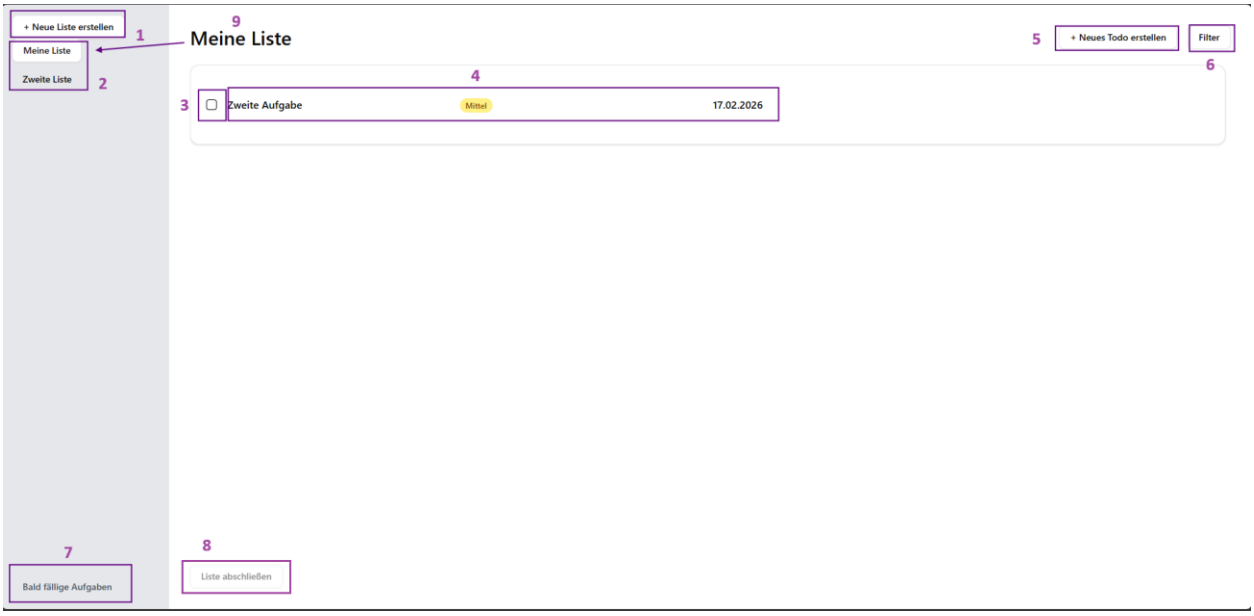
Abbildung 7



```cpp
#pragma once

#include <WinSock2.h>
#include <ws2tcpip.h>
#include <string>
#include <fstream>
#include <sstream>
#include <iostream>

#pragma comment(lib, "ws2_32.lib")

class SocketHelper {

public:

    bool LoadConfig(const std::string& path) {
        std::ifstream file(path);
        if (!file.is_open()) return false;

        std::string line;
        while (std::getline(file, line)) {
            if (line.rfind("IP=", 0) == 0) ip = line.substr(3);
            if (line.rfind("Port=", 0) == 0) port = std::stoi(line.substr(5));
        }

        return true;
    }

    std::string GetConfigValues(int choice) {

        if (choice == 1)
            return ip;

        if (choice == 2)
            return std::to_string(port);

        if (choice != 1 || choice != 2)
            return "Error! . . .";
    }

    bool OpenSocket() {
        currentSocket = socket(AF_INET, SOCK_STREAM, 0);
        return currentSocket != INVALID_SOCKET;
    }

    bool BindFromConfig() {
        sockaddr_in addr{};
        addr.sin_family = AF_INET;
        addr.sin_port = htons(port);
        inet_pton(AF_INET, ip.c_str(), &addr.sin_addr);

        return bind(currentSocket, (sockaddr*)&addr, sizeof(addr)) != SOCKET_ERROR;
    }

    bool Listen() {
        return listen(currentSocket, SOMAXCONN) != SOCKET_ERROR;
    }

    SOCKET AcceptClient() {
        return accept(currentSocket, nullptr, nullptr);
    }

    std::string ReadHTTPRequest(SOCKET client) {
        char buffer[4096];
        int bytes = recv(client, buffer, sizeof(buffer), 0);
        if (bytes <= 0) return "";

        return std::string(buffer, bytes);
    }

    void SendHTTPResponse(SOCKET client, const std::string& body) {
        std::string header =
            "HTTP/1.1 200 OK\r\n"
            "Content-Type: text/plain\r\n"
            "Access-Control-Allow-Origin: *\r\n"
            "Content-Length: " + std::to_string(body.size()) + "\r\n"
            "\r\n";

        send(client, header.c_str(), (int)header.size(), 0);
        send(client, body.c_str(), (int)body.size(), 0);
    }

    SOCKET GetSocket() { return currentSocket; }

private:
    SOCKET currentSocket = INVALID_SOCKET;
    std::string ip = "127.0.0.1";
    int port = 8080;
};
```

Runbook (Bedienungsanleitung)

Hauptseite



Runbook Abbildung 1

1: Neue Liste erstellen

2: Anzeige/ Auswahl der vorhandenen Listen

3: Checkbox eines To-dos in der ausgewählter Liste

4: Informationen vom To-dos in der ausgewählten Liste

5: Erstellen eines neuen To-dos in einer Liste (die geöffnete)

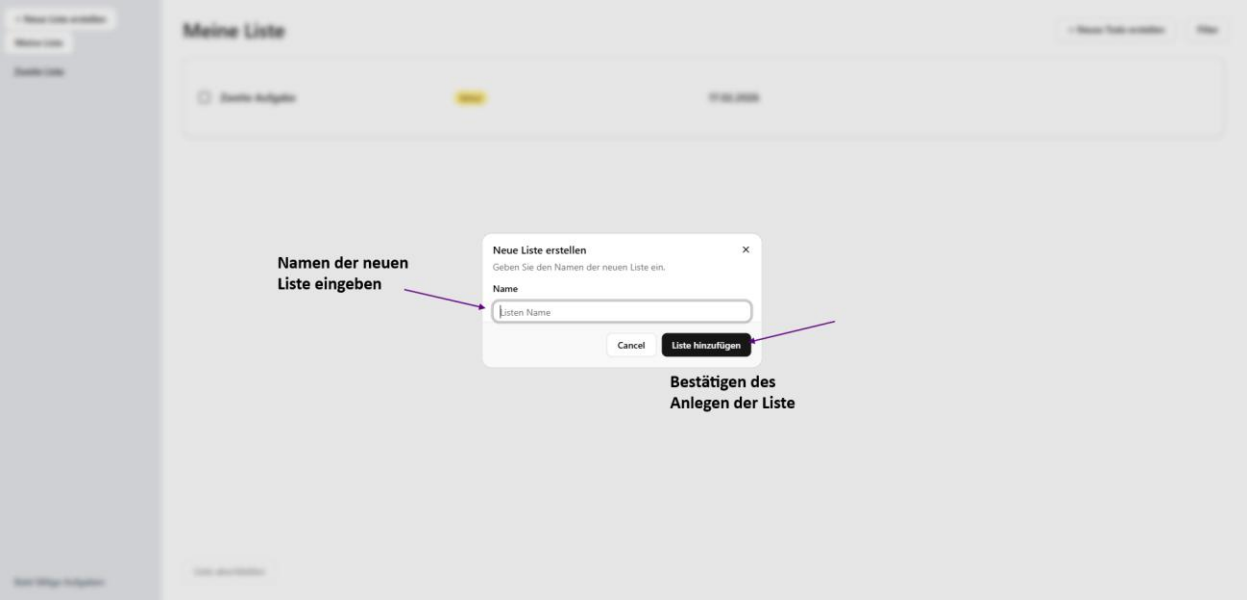
6: Öffnen des Filterfensters für das filtern innerhalb einer Liste

7: Filtern über alle Listen anhand des Fälligkeitsdatums; Anzeige kürzeste Fälligkeit oben

8: Löschen einer Liste (es muss mindestens ein To-do vorhanden sein und alle abgeschlossen innerhalb der Liste)

9: Name der aktuell ausgewählten Liste

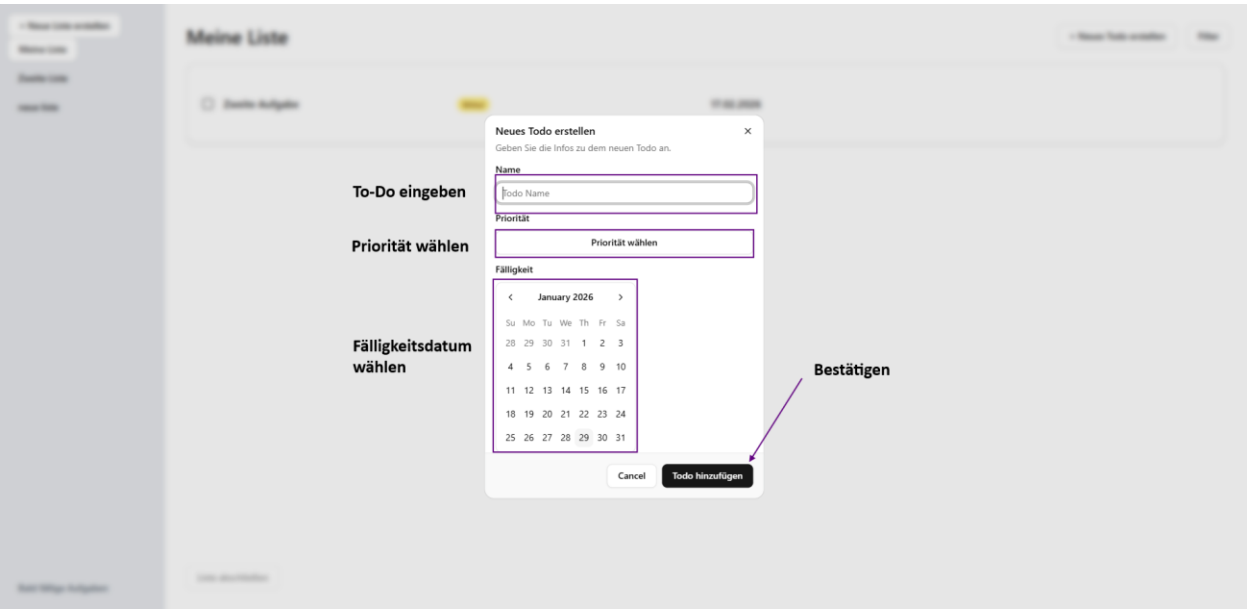
Eine Neue Liste erstellen:



Runbook Abbildung 2

Es wurde aus Abbildung 1 , der Button zum erstellen einer neuen liste gedrückt (Feld 1) Hier wird nun der Name der neuen Liste eingegeben und mithilfe des Buttons bestätigt. Eine neue Liste ist erstellt.

Ein neues To-do erstellen:

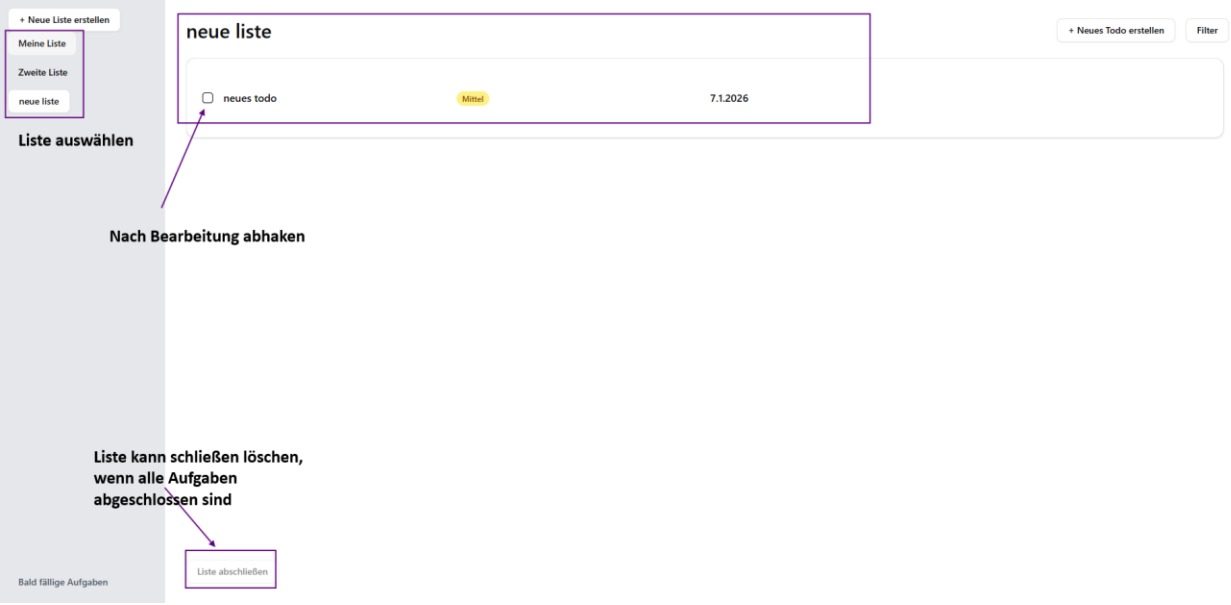


Runbook Abbildung 3

Es wurde nach dem Auswählen der Liste in der ein neues to-do erstellt werden soll (Abb. Runbook 1, Feld 2) der Button zum erstellen eines neuen To-dos gedrückt (Abb.

Runbook, Feld 5). Nun öffnet sich eine Maske in der man den Titel/Beschreibung, Priorität, Fälligkeitsdatum konfigurieren muss. Anschließend bestätigt man mit dem Button ‘Todo hinzufügen’

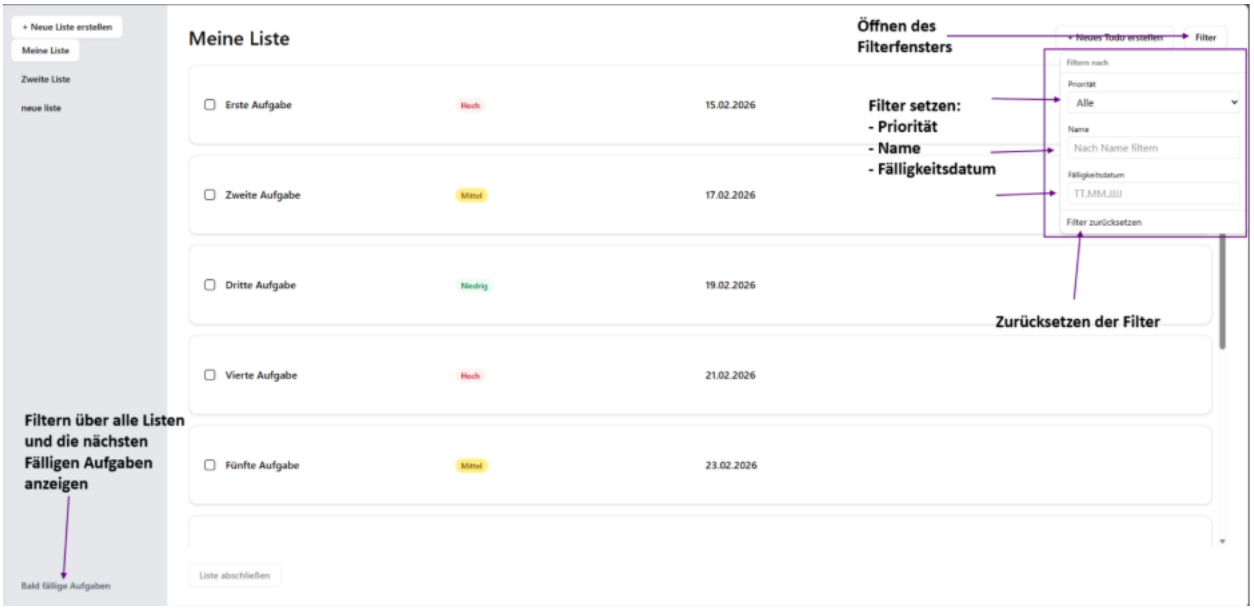
Quittieren von abgeschlossenen To-dos und Löschen einer Liste:



Runbook Abbildung 4

Eine Liste ist ausgewählt, dann werden die To-dos angezeigt und man kann mithilfe der Checkbox das Abschließen des To-dos quittieren. Eine Liste muss mindestens ein To-do haben und alle To-dos müssen abgeschlossen sein, damit man die Liste, mithilfe des Buttons in der Abbildung unten zu sehen, abschließen/Löschen kann.

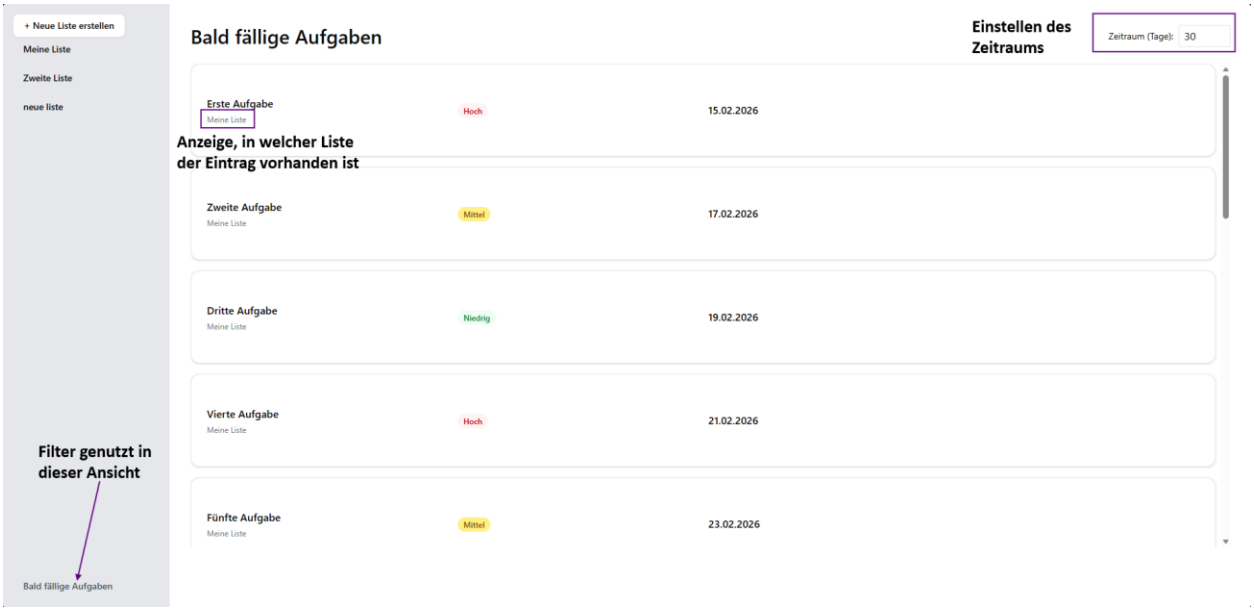
Filtern in Listen:



Runbook Abbildung 5

Es wurde nach dem Auswählen der Liste in der gefiltert werden soll der Button Filtern (Abb. Runbook 1, Feld 6) gedrückt, damit öffnet sich das Filtermenü, oben rechts zu sehen, für das Filtern innerhalb einer Liste. Anhand der gesetzen Filter ändert sich das ausgegebene Ergebnis der Liste in der Mitte der Runbook Abb. 5.

Filtern für nächsten Fälligkeiten über alle Listen:



Runbook Abbildung 6

Für das nutzen dieser Funktion wird lediglich der Button unten links genutzt (Abb. Runbook 1; Feld 7).