ClockWise

Project Oriented Time Management Application

Software-Entwicklungspraktikum (SEP)
Sommersemester 2025

Technischer Entwurf

Auftraggeber
Technische Universität Braunschweig
Peter L. Reichertz Institute for Medical Informatics
Prof. Dr. Thomas M. Deserno
Mühlenpfordtstraße 23
38106 Braunschweig

Betreuer: Corinna Thoben

Auftragnehmer:

Name	E-Mail-Adresse
Joud Mawad	j.mawad@tu-braunschweig.de
Laden Zeynep Erkenci	l.erkenci@tu-braunschweig.de
Sophie Gebauer	sophie.gebauer@tu-braunschweig.de
Luis Jair Gutierrez Pacheco	l.gutierrez-pacheco@tu-braunschweig.de
Chantal Ebben	c.ebben@tu-braunschweig.de
Merle Lüer	m.lueer@tu-braunschweig.de

Braunschweig, 9. Juli 2025

Bearbeiterübersicht

Kapitel	Autoren	Kommentare
1	Luis Jair Gutierrez Pacheco	
1.1	Luis Jair Gutierrez Pacheco	
1.2	Luis Jair Gutierrez Pacheco	
1.3	Luis Jair Gutierrez Pacheco	
2	Chantal Ebben	
2.1	Chantal Ebben	
2.2	Chantal Ebben	
2.3	Chantal Ebben	
2.3.1	Chantal Ebben	
2.3.2	Chantal Ebben	
2.4	Chantal Ebben	
2.5	Laden Zeynep Erkenci	
2.5.1	Laden Zeynep Erkenci	
2.5.2	Laden Zeynep Erkenci	
2.5.3	Laden Zeynep Erkenci	
2.6	Laden Zeynep Erkenci	
2.6.1	Laden Zeynep Erkenci	
2.6.2	Laden Zeynep Erkenci	
2.6.3	Laden Zeynep Erkenci	
2.7	Laden Zeynep Erkenci	
2.8	Laden Zeynep Erkenci	
2.9	Joud Mawad	
2.10	Joud Mawad	
2.11	Joud Mawad	
2.12	Joud Mawad	
2.13	Chantal Ebben	
3	Sophie Gebauer, Merle Lüer, Joud Mawad	
3.1	Sophie Gebauer, Merle Lüer, Joud Mawad	
3.2	Sophie Gebauer, Merle Lüer, Joud Mawad	
3.3	Sophie Gebauer, Merle Lüer, Joud Mawad	

3.3.1	Joud Mawad	
3.3.2	Joud Mawad, Merle Lüer	
3.3.3	Joud Mawad	
3.3.4	Merle Lüer	
3.3.5	Sophie Gebauer	
4	Chantal Ebben	
5	Laden Zeynep Erkenci, Luis Jair Gutierrez Pacheco, Sophie Gebauer	
5.1	Laden Zeynep Erkenci	
5.2	Laden Zeynep Erkenci	
5.3	Luis Jair Gutierrez Pacheco	
5.4	Luis Jair Gutierrez Pacheco	
5.5	Sophie Gebauer	
6	Merle Lüer, Sophie Gebauer	
6.1	Merle Lüer	
6.2	Sophie Gebauer	
7	Sophie Gebauer	
8	Alle Teammitglieder	
9	Chantal Ebben	
9.1	Chantal Ebben	
9.2	Chantal Ebben	
9.3	Chantal Ebben	
10	Alle Teammitglieder	

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

In einer zunehmend projektbasierten und digitalisierten Arbeitswelt wird strukturierte Zeiterfassung zu einem entscheidenden Faktor für effiziente Planung und transparente Zusammenarbeit. Die Anwendung ClockWise adressiert dieses Bedürfnis mit einer webbasierten Lösung zur projektorientierten Zeiterfassung und Analyse.

Dieses Dokument beschreibt die fachliche Struktur und das erwartete Verhalten des Systems ClockWise. Anhand von typischen Nutzungsszenarien und modellbasierten Diagrammen wird dargestellt, wie Nutzer:innen mit dem System interagieren und welche Abläufe dabei im Vordergrund stehen. Ziel ist es, einen systematischen Überblick über die Funktionalität der Anwendung zu geben, aus einer fachlichen Perspektive.

Das vorliegende Dokument ist wie folgt aufgebaut: Kapitel 1 stellt die fachliche Grundlage sowie typische Nutzungsszenarien vor. Kapitel 2 analysiert die Funktionen anhand konkreter Abläufe und Interaktionen. Kapitel 3 beschreibt das Datenmodell und Kapitel 4 enthält Konfigurationshinweise für den Einsatz der Anwendung.

1.1 Projektübersicht

ClockWise ist eine browserbasierte Webanwendung zur projektorientierten Zeiterfassung und Auswertung. Die Anwendung ermöglicht es Nutzer:innen, Zeitdaten strukturiert zu erfassen, sowohl live per Start/Stopp Funktion als auch manuell und diese in Kalender und Diagrammansichten zu analysieren. Zusätzlich erlaubt das System die Definition von Zeitlimits für Projekte und gibt bei deren Überschreitung automatisierte Hinweise.

Zu den Kernfunktionen zählen unter anderem die Verwaltung von Projekten und Aufgaben, die zeitliche Kategorisierung von Aktivitäten, eine Teamfunktion mit Rollenverteilung sowie Exportmöglichkeiten der erfassten Daten. Die Bedienung erfolgt über eine intuitive grafische Oberfläche im Webbrowser, wodurch keine lokale Installation erforderlich ist.

ClockWise richtet sich an Einzelpersonen und Teams, die ihre Zeitplanung systematisch und nachvollziehbar gestalten möchten. Die Anwendung ist vielseitig einsetzbar, z. B. im universitären Umfeld, im Arbeitskontext oder bei der persönlichen Projektorganisation.

1.2 Projektdetails

In diesem Abschnitt werden die zentralen Funktionen der Anwendung ClockWise aus fachlicher Sicht beschrieben. Dabei steht im Vordergrund, wie der:die Nutzer:in mit dem System interagiert, welche Abläufe vorgesehen sind und welche fachlichen Anforderungen diesen Prozessen zugrunde liegen. Zur besseren Verständlichkeit werden zentrale Funktionsabläufe — wo sinnvoll — zusätzlich durch Aktivitätsdiagramme visualisiert.

1.2.1 Registrierung und Login

Um ClockWise nutzen zu können, ist zunächst eine Registrierung erforderlich. Der:Die Nutzer:in legt ein persönliches Konto an, das durch eine Kombination aus E-Mail Adresse und Passwort gesichert ist. Nach erfolgreicher Registrierung erfolgt die Anmeldung über die Login-Seite.

Nach dem Login wird der:die Nutzer:in direkt zum Dashboard weitergeleitet, das den Einstiegspunkt für alle weiteren Funktionen bildet. Die Authentifizierung ermöglicht eine individuelle Nutzung der Anwendung und gewährleistet die sichere Speicherung persönlicher Zeitdaten.

Alternative Abläufe wie OAuth-Login oder Passwort-Zurücksetzung sind ebenfalls berücksichtigt (vgl. Abbildungen 1.1 und 1.2).

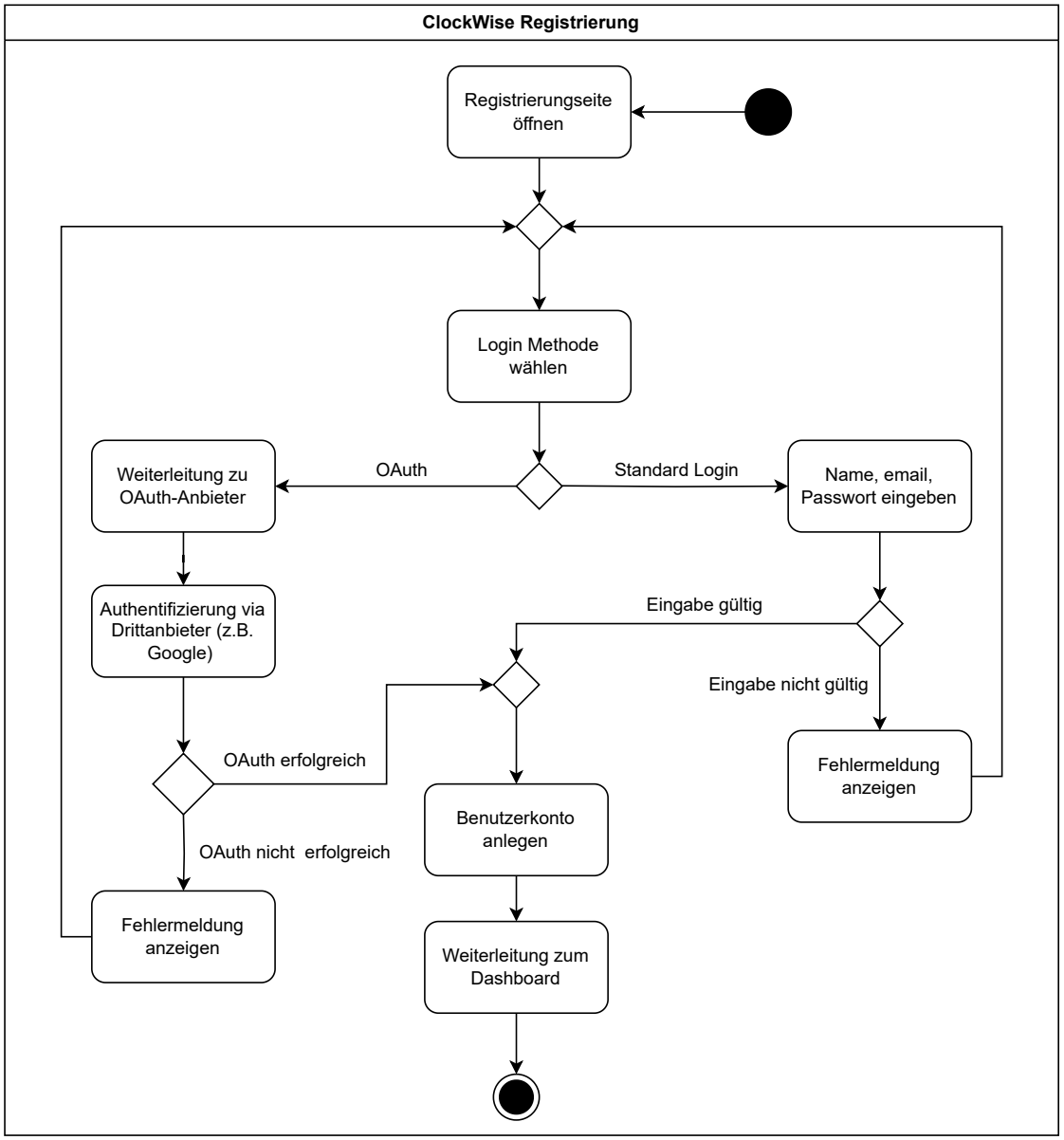

Abbildung 1.1: Aktivitätsdiagramm F10: Registrierung eines Nutzerkontos

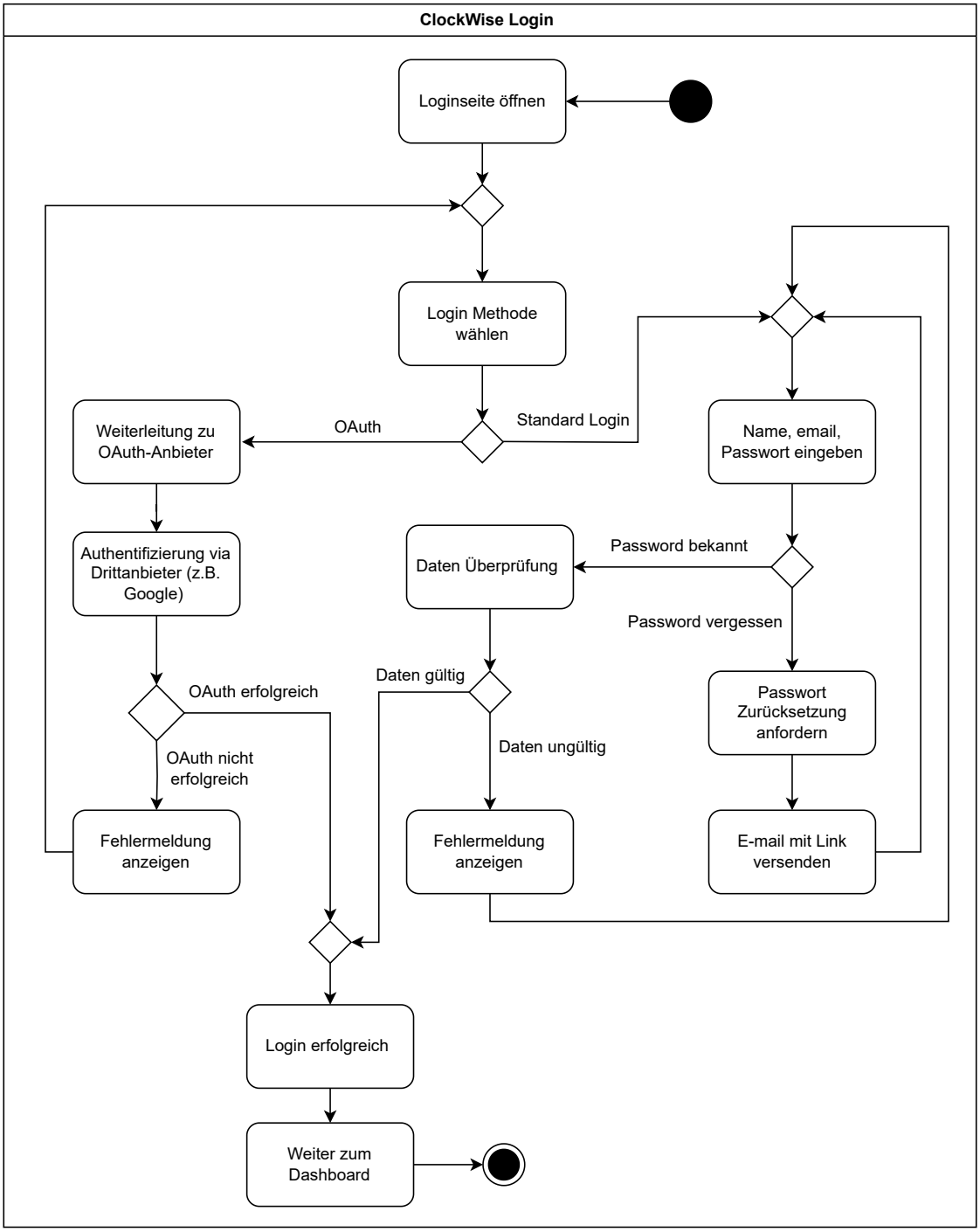

Abbildung 1.2: Aktivitätsdiagramm F20: Login eines Nutzers

1.2.2 Zeiterfassung starten und bearbeiten

Die zentrale Funktion von ClockWise ist die projektbezogene Erfassung von Zeit. Der:Die Nutzer:in kann entweder eine laufende Erfassung über eine Start/Stopp-Funktion aktivieren oder bestehende Zeiteinträge manuell hinzufügen und bearbeiten. Jeder Zeiteintrag wird einem spezifischen Projekt zugeordnet und kann bei Bedarf nachträglich angepasst werden.

Die Anwendung stellt sicher, dass laufende Zeiteinträge eindeutig sind, sodass nicht mehrere Aktivitäten gleichzeitig erfasst werden können. Durch die intuitive Benutzeroberfläche kann die Zeitdokumentation flexibel an den individuellen Arbeitsstil angepasst werden – etwa bei kontinuierlichem Arbeiten oder fragmentierten Aufgabenblöcken.

Der vollständige Ablauf der Zeiterfassung inklusive Start, Pause, Stoppen und Speichern wird in Abbildung 1.3 dargestellt.

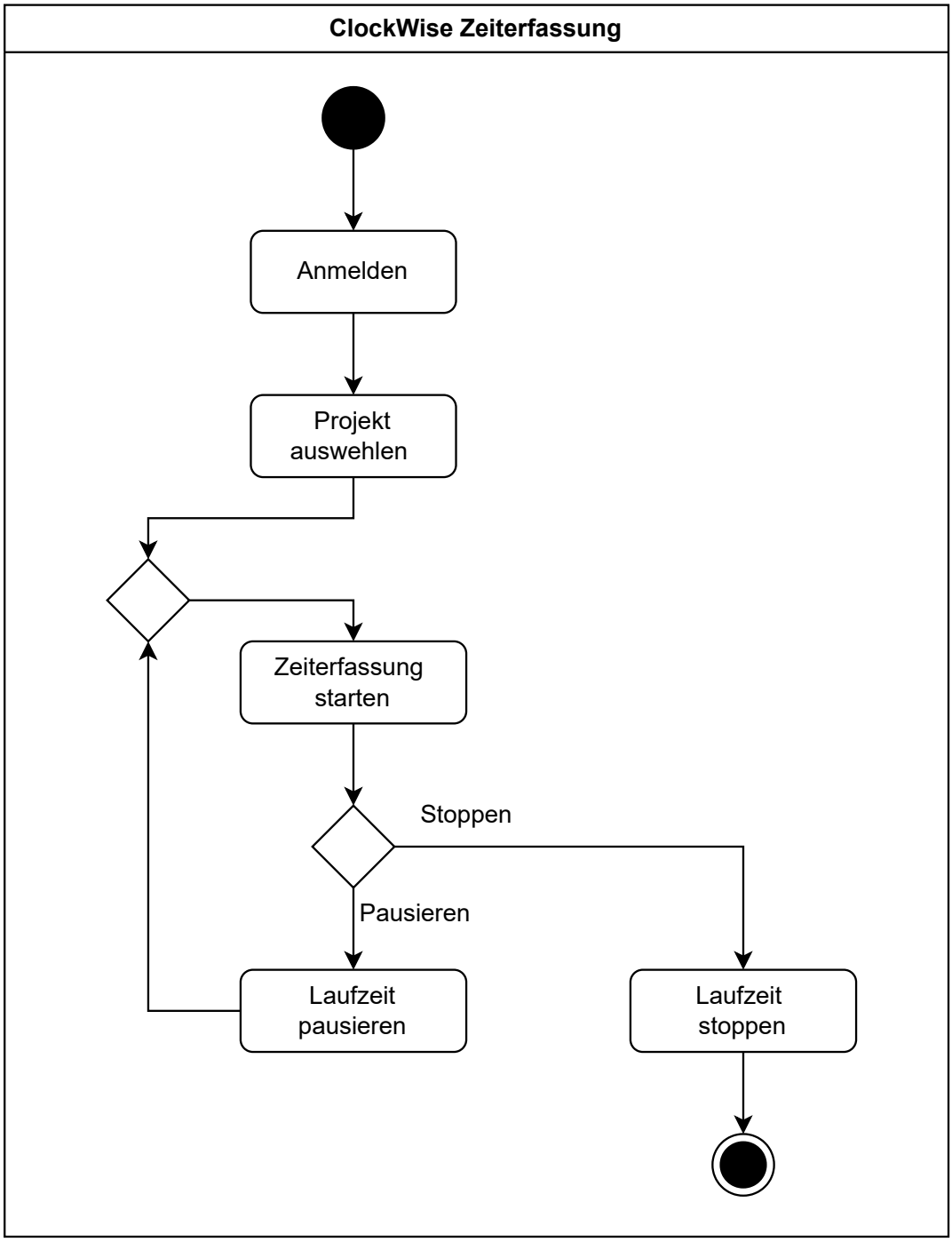

Abbildung 1.3: Aktivitätsdiagramm 3.3 des Pflichtenhefts F30 und F40: Zeiterfassung

1.2.3 Projekte und Aufgaben verwalten

ClockWise ermöglicht es Nutzer:innen, Projekte anzulegen, Aufgaben zu definieren und diese individuell oder innerhalb eines Teams zu strukturieren. Jedes Projekt dient als übergeordnete Einheit zur Sammlung von Zeiteinträgen, die thematisch oder zeitlich zusammengehören. Innerhalb eines Projekts können konkrete Aufgaben erstellt werden, um Arbeitsabschnitte feiner zu unterteilen.

Die Organisation in Projekte und Aufgaben erlaubt eine gezielte Planung und Nachverfolgung von Tätigkeiten. Zeiteinträge lassen sich eindeutig zuordnen, sodass die spätere Analyse nicht nur auf Projektebene, sondern auch auf Task Ebene erfolgen kann.

Beim Erstellen oder Bearbeiten von Projekten und Aufgaben erfolgt zunächst eine Validierung der eingegebenen Daten. Danach werden die Informationen in der Datenbank gespeichert. Aufgaben können optional einem bestimmten Teammitglied zugewiesen werden, wobei nur Projektverantwortliche entsprechende Berechtigungen besitzen. Neben der Erstellung sind auch das Aktualisieren oder Löschen von Aufgaben möglich. Dieser gesamte Ablauf ist in Abbildung 1.4 dargestellt.

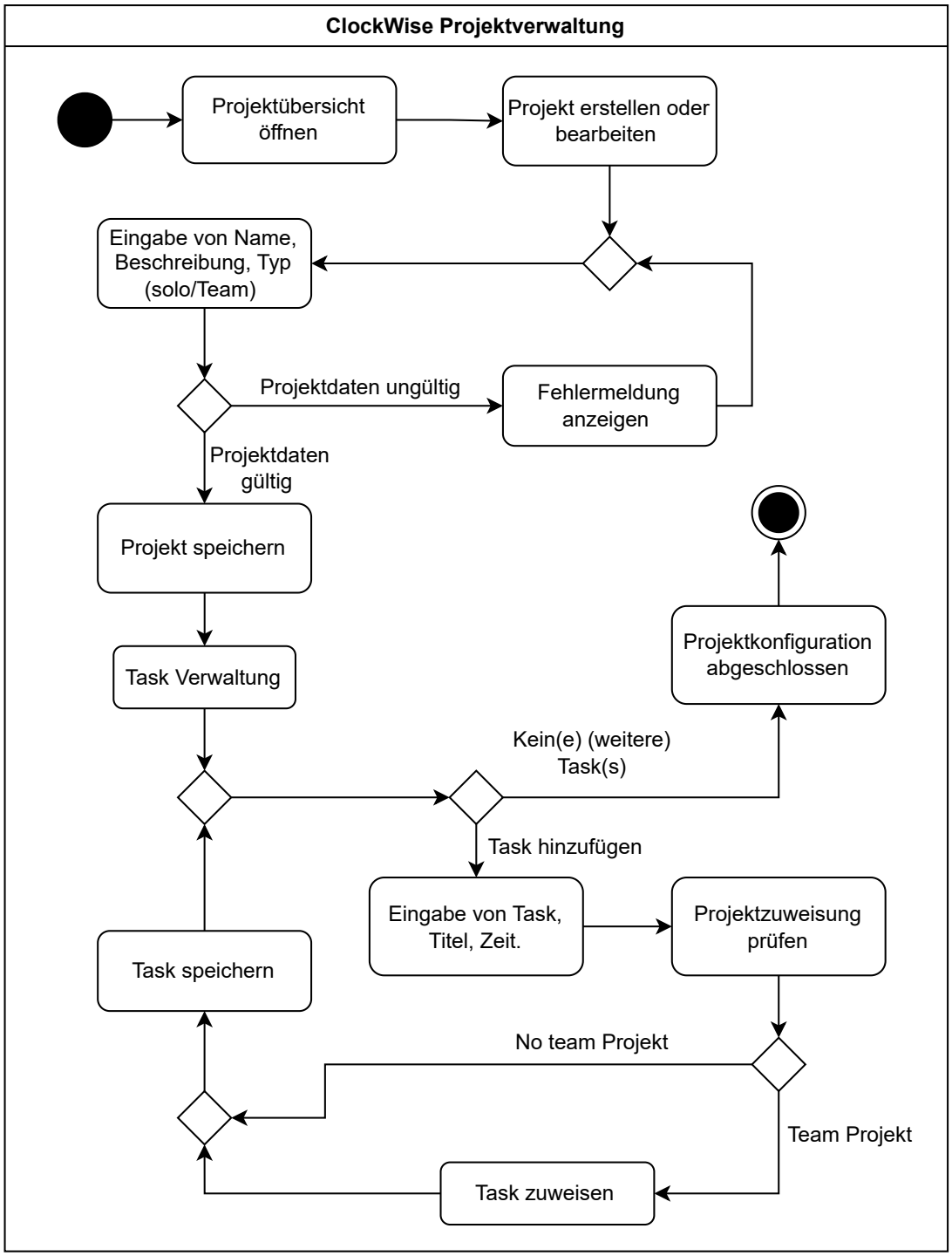

Abbildung 1.4: Aktivitätsdiagramm F50 und F60: Projektverwaltung

1.2.4 Analyse und Kalenderansicht

Zur Unterstützung der Selbstorganisation stellt ClockWise verschiedene Möglichkeiten zur Auswertung der erfassten Zeitdaten bereit. Nutzer:innen können ihre Zeitverteilung in Form von Diagrammen visualisieren oder über eine Kalenderansicht nachvollziehen, wann welche Aktivitäten stattgefunden haben.

Die Kalenderansicht zeigt alle Zeiteinträge tagesweise strukturiert und erleichtert so die zeitliche Orientierung. Zusätzlich werden projektbezogene Übersichten generiert, die eine Bewertung des Fortschritts im Vergleich zu vorher definierten Zielzeiten ermöglichen. Ein Fortschrittsbalken visualisiert dabei das Verhältnis von geleisteter zu geplanter Zeit.

Diese Funktionen dienen nicht nur der Reflexion, sondern auch der Motivation, indem sie individuelle Arbeitsmuster sichtbar machen und zur besseren Zeitplanung beitragen.

Da es sich bei den Analyse und Kalenderfunktionen nicht um klassische Prozessabläufe handelt, sondern um die Darstellung bereits gespeicherter Zeitdaten, wurde auf ein Aktivitätsdiagramm verzichtet. Die Nutzung dieser Funktionen erfolgt primär lesend und datengetrieben, sodass eine modellbasierte Visualisierung hier keinen zusätzlichen Erkenntnisgewinn bietet.

1.2.5 Teamfunktion und Rollen

ClockWise bietet eine integrierte Teamfunktion, die es mehreren Nutzer:innen ermöglicht, gemeinsam an Projekten zu arbeiten. Dabei können Teammitglieder einzelnen Projekten zugeordnet und unterschiedliche Rollen vergeben werden, z. B. zur Unterscheidung zwischen Administrator:innen und regulären Mitgliedern.

Administrator:innen haben die Möglichkeit, Projekte zu erstellen, Teammitglieder zu verwalten und Rollen zu vergeben. Reguläre Mitglieder können Zeiteinträge hinzufügen und bestehende Aufgaben bearbeiten, jedoch keine grundlegenden Strukturen verändern. Diese Rollenverteilung schafft klare Zuständigkeiten und vereinfacht die Koordination innerhalb des Teams.

Zusätzlich wird die Gesamtzeitverteilung im Team sichtbar gemacht, wodurch eine transparente Aufgabenverteilung gefördert und Überlastungen frühzeitig erkannt werden können.

Die Teamfunktion umfasst mehrere kontextabhängige Teilprozesse wie die Verwaltung von Mitgliedern, die Vergabe von Rollen und die Zuweisung von Aufgaben. Da diese Prozesse stark von der jeweiligen Nutzerrolle (z. B. Administrator:in oder reguläres Mitglied) und dem aktuellen Projektstatus abhängen, wurde auf ein einzelnes Aktivitätsdiagramm verzichtet. Eine schematische Darstellung hätte entweder zu einer unzureichenden Generalisierung oder zu einer übermäßig komplexen Visualisierung geführt. Stattdessen wurde der Ablauf verbal differenziert beschrieben.

1.3 Systemübersicht

Die folgende Tabelle bietet einen systematischen Überblick über die funktionalen Hauptbereiche der Anwendung ClockWise. Für jede Funktion wird die technische Umsetzung kurz beschrieben und die beteiligten Systemkomponenten benannt.

Funktion	Technische Umsetzung	Komponenten
Zeiterfassung (F30, F40)	Start/Stopp oder manuelle Eingabe, ein aktiver Eintrag pro Nutzer:in. Dauerberechnung automatisch. Untitled Task bei leerem Taskfeld. Pausieren/Fortsetzen möglich.	C10 (UI), C20 (Logik), C30 (TimeEntry Modell)
Projektverwaltung (F50)	CRUD Funktionalität mit Zeitlimits, Deadlines, Projekttypen, Teamzuweisung und optionalen Kreditpunkten. Validierung erfolgt serverseitig.	C10 (Projektformular), C20 (API Logik), C30 (Project Modell)
Taskverwaltung (F60)	Aufgaben können Projekten zugewiesen oder unabhängig davon erstellt werden. Sie beinhalten Titel, Beschreibung, Kategorie und Fälligkeitsdatum. Bei leerem Titel wird „Untitled Task“ gesetzt.	C10 (Taskformular), C20 (API F60), C30 (Task Modell)
Exportfunktion (F70)	Export der erfassten Zeitdaten als CSV Datei. Export erfolgt projektbezogen und beinhaltet Zeit, Dauer, Nutzer:in, Task und optional Teamrolle.	C10 (Downloadbutton), C20 (Routen F70), C30 (TimeEntry Modell)
Analyse (F80, F90)	Auswertung der erfassten Zeiten pro Tag und Projekt. Beinhaltet Wochenübersichten, Fortschrittsbalken, Kalenderansicht sowie den Vergleich zwischen Soll- und Ist-Zeit pro Projekt.	C10 (Analyse UI), C20 (Funktionen F80/F90)
Benachrichtigungen (Teil von F90)	Automatische Hinweise bei systemrelevanten Ereignissen. Speicherung in der Datenbank, Filterung nach Nutzer:in, Anzeige nach Erstellungszeit. Möglichkeit zum Löschen durch Nutzer:in.	C10 (Benachrichtigungsanzeige), C20 (Trigger), C30 (Notification Modell), C50
Teamfunktion (F100)	Erstellung von Teams, Rollenvergabe (Admin/Member), Mitgliederverwaltung (Hinzufügen/Entfernen), Rechteprüfung durch Authentifizierung. Benachrichtigung bei Teamerstellung.	C10 (Team-UI), C20 (Logik F100), C30 (Team-, UserTeam-Modell), C40
Authentifizierung (F10, F20)	Registrierung, Login, OAuth-Anmeldung, Passwort Zurücksetzen, Sessionverwaltung.	C10 (Formulare), C20 (Flask Auth), C30 (User Modell)

Tabelle 1.1: Systemübersicht der Kernfunktionen von ClockWise

2 Analyse der Produktfunktionen

In diesem Kapitel wird das Verhalten der im Pflichtenheft definierten Produktfunktionen untersucht und in Form von Sequenzdiagrammen veranschaulicht. Ziel ist es, ein besseres Verständnis über das Zusammenspiel der beteiligten Systemkomponenten zu gewinnen und aufzuzeigen, wie bestimmte Abläufe deren Verhalten und Funktionalität beeinflussen.

2.1 Analyse von Funktionalität <F10>: Registrierung

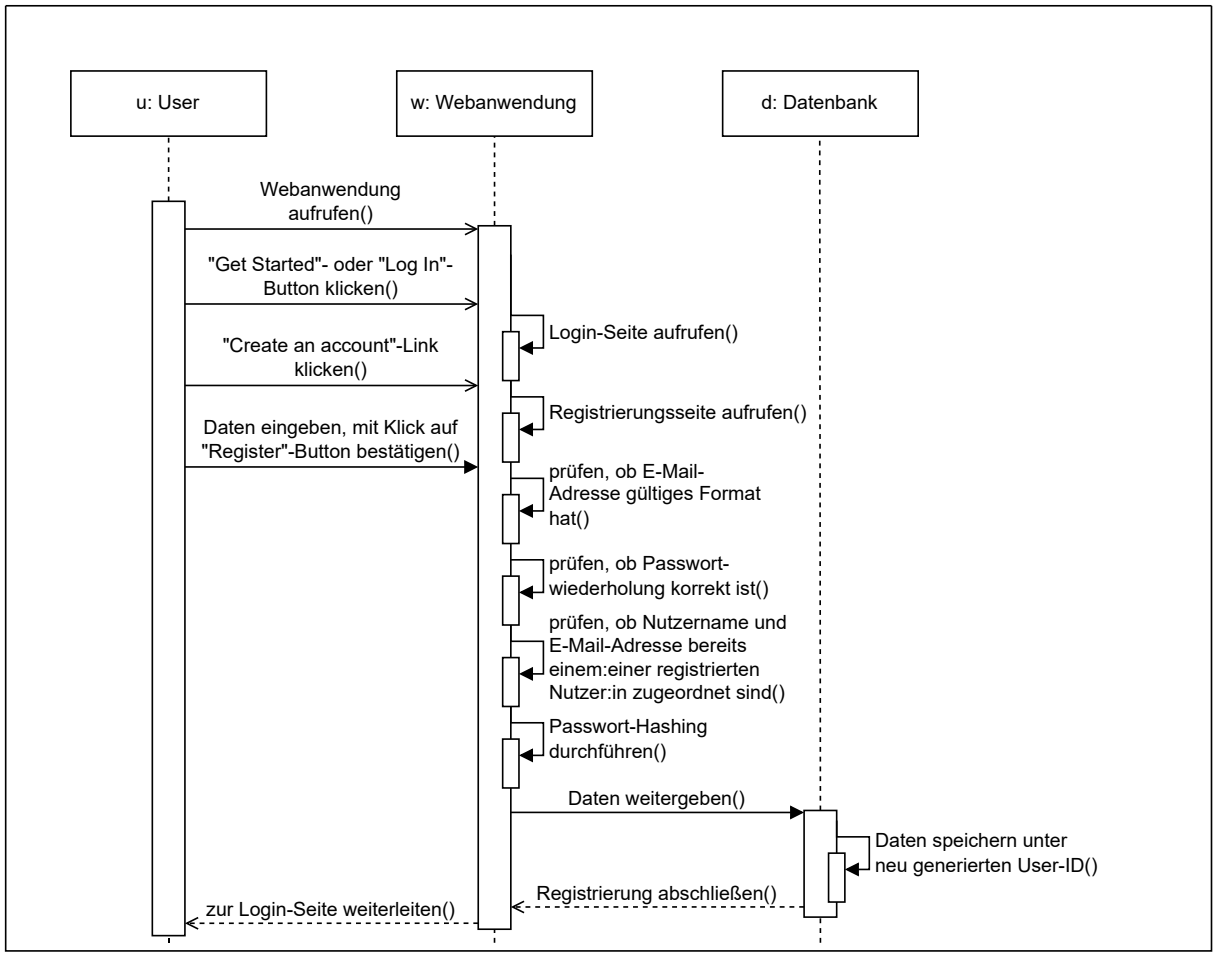

Abbildung 2.1: Sequenzdiagramm F10: Registrierung

Im Sequenzdiagramm zu der Funktion <F10> wird der Prozess der Erstellung eines Nutzerkontos zur erstmaligen Nutzung der Webanwendung ClockWise dargestellt. Zu Beginn wird die Webanwendung durch das Ausführen von app.py (lokal auf dem Endgerät) aufgerufen. Durch einen Klick auf den „Get Started“- oder „Log In“-Button gelangt der:die Nutzer:in zur Login-Seite, von wo aus über den Link „Create an account“ die Weiterleitung zur Registrierungsseite erfolgt. Dort gibt der:die Nutzer:in die für die Kontoerstellung erforderlichen Daten ein: Vorname, Nachname, Nutzername, E-Mail-Adresse, Passwort und Passwortwiederholung. Optional kann ein Profilbild ausgewählt werden.

Die Übermittlung der Daten an die Webanwendung erfolgt nach einem Klick auf den „Register“-Button. Die Anwendung prüft zunächst, ob das Format der E-Mail-Adresse gültig ist und ob die Passwortwiederholung korrekt ist. Wird dabei festgestellt, dass das E-Mail-Format oder die Passwortwiederholung inkorrekt sind, wird dem:der Nutzer:in eine entsprechende Fehlermeldung

angezeigt. Zusätzlich wird überprüft, ob der Nutzername oder die E-Mail-Adresse bereits einem:einer registrierten Nutzer:in zugeordnet sind, was ebenfalls die Ausgabe einer Fehlermeldung bewirken würde. Auf die Darstellung der Ausgabe der Fehlermeldungen an den:die Nutzer:in im Falle eines festgestellten Fehlers bei einer der Überprüfungen wird in diesem Diagramm aus Platzgründen verzichtet. Anschließend erfolgt das Hashing des Passworts und dann werden die vollständigen und validierten Daten an die Datenbank weitergegeben. In der Datenbank werden daraufhin die Nutzerdaten unter einer neu generierten User-ID gespeichert. Nach erfolgreicher Registrierung wird der:die Nutzer:in auf die Login-Seite weitergeleitet.

2.2 Analyse von Funktionalität <F20>: Login

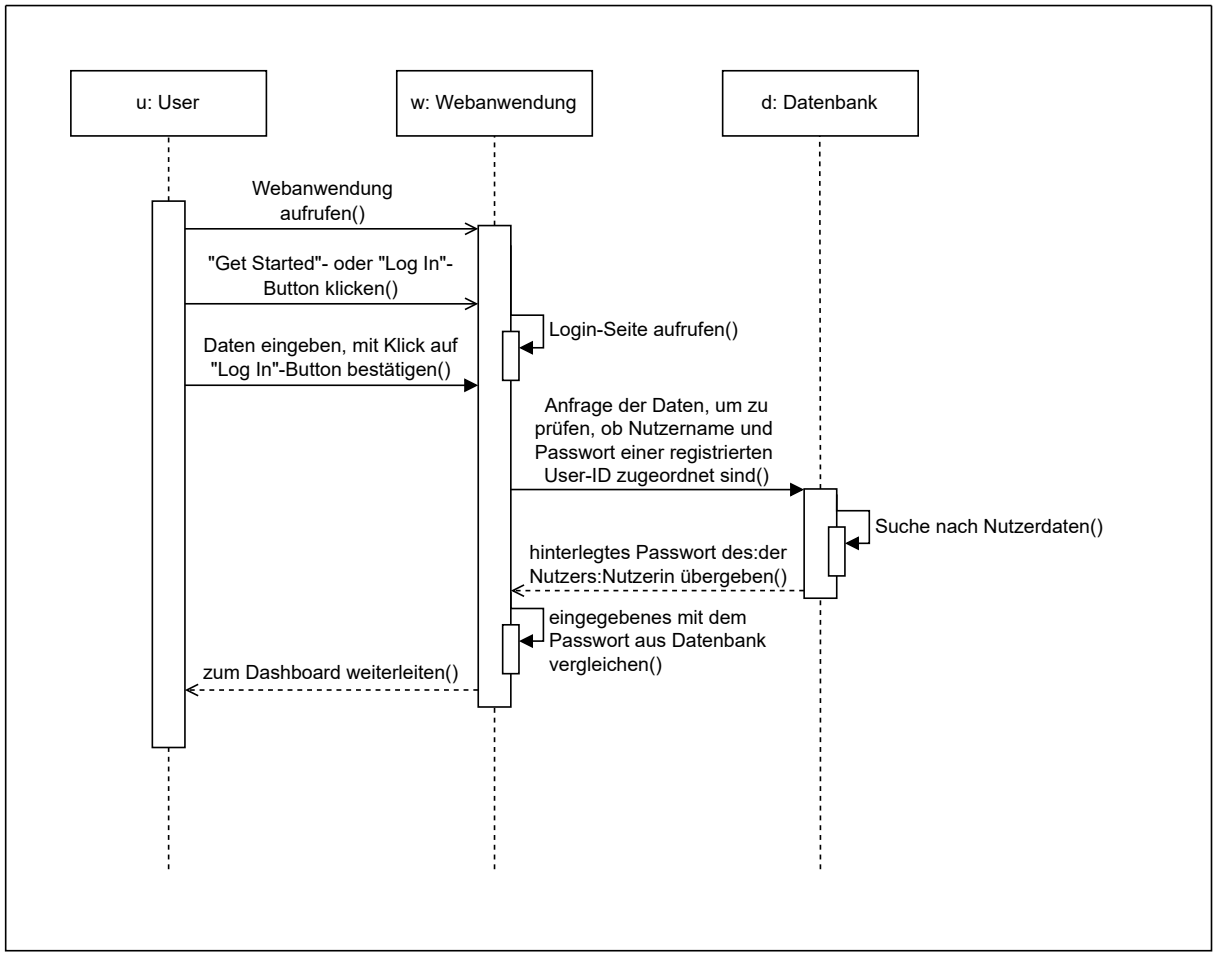

Abbildung 2.2: Sequenzdiagramm F20: Login

Im Sequenzdiagramm zur Funktion <F20> wird der Prozess der Anmeldung mit bereits bestehenden Zugangsdaten dargestellt. Nach dem Aufrufen der Webanwendung gelangt der:die Nutzer:in durch einen Klick auf den „Get Started“- oder „Log In“-Button zur Login-Seite. Dort gibt der:die Nutzer:in die für die Anmeldung erforderlichen Daten ein: Nutzername und Passwort. Mit einem Klick auf den „Log In“-Button werden die Daten an die Webanwendung übermittelt, die anschließend eine Datenbankabfrage durchführt, um zu prüfen, ob diese Daten einer bereits registrierten Nutzer-ID zugeordnet sind. Sind die Daten nicht in der Datenbank vorhanden, erhält der:die Nutzer:in eine Fehlermeldung, dass die eingegebenen Anmeldedaten inkorrekt sind. Werden die Daten gefunden, übermittelt die Datenbank das gespeicherte Passwort an die Webanwendung, die das eingegebene Passwort mit dem gespeicherten vergleicht. Stimmen beide überein, wird die Anmeldung erfolgreich ausgeführt und der:die Nutzer:in zum persönlichen Dashboard weitergeleitet. Stimmen sie nicht überein, erhält der:die Nutzer:in eine Fehlermeldung und der Zugriff wird verweigert. Auf die Darstellung der Fehlermeldungen, die dem:der

Nutzer:in im Falle eines auftretenden Fehlers beim Abrufen der Daten aus der Datenbank oder beim Passwortvergleich angezeigt werden, wird in diesem Diagramm aus Platzgründen verzichtet.

2.3 Analyse von Funktionalität <F30>: Zeiterfassung (Start/Stop)

2.3.1 Funktionalität <F30.1>: Zeiterfassung neu angelegter Aufgaben

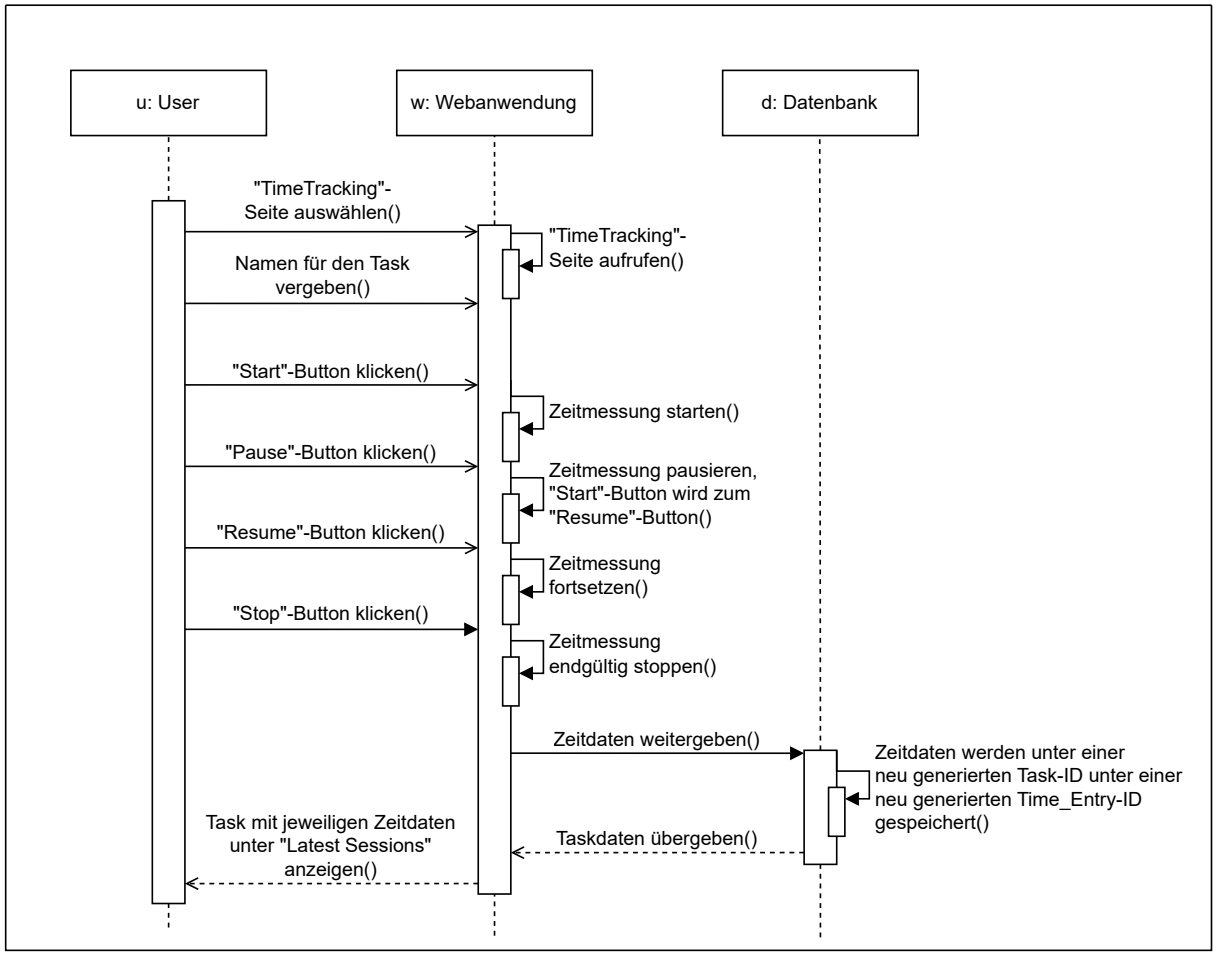

Abbildung 2.3: Sequenzdiagramm F30.1: Zeiterfassung (Start/Stop) neu angelegter Aufgaben

Im Sequenzdiagramm zu der Funktion <F30.1> wird der Prozess des Erfassens von Arbeitszeiten neu angelegter Aufgaben durch Starten, Pausieren und Stoppen eines Timers dargestellt. Auf der „TimeTracking“-Seite kann ein Name für den neuen Task eingegeben werden. Durch einen Klick auf den „Start“-Button wird die Zeitmessung begonnen, durch „Pause“ unterbrochen. Wird die Zeitmessung pausiert, erscheint anstelle des „Pause“-Buttons der „Resume“-Button, mit dem die Zeitmessung fortgesetzt werden kann.

Ein Klick auf den „Stop“-Button beendet die Zeitmessung endgültig. Die erfassten Daten werden anschließend an die Datenbank übermittelt und dort unter einer neu generierten Task-ID sowie unter einer neu generierten Time_Entry-ID gespeichert. Nach erfolgreicher Speicherung werden

die Daten an die Webanwendung zurückgegeben, sodass die aktualisierte Benutzeroberfläche den neuen Task auf der „TimeTracking“-Seite unter „Latest Sessions“ anzeigen kann.

2.3.2 Funktionalität <F30.2>: Zeiterfassung bereits existierender Aufgaben

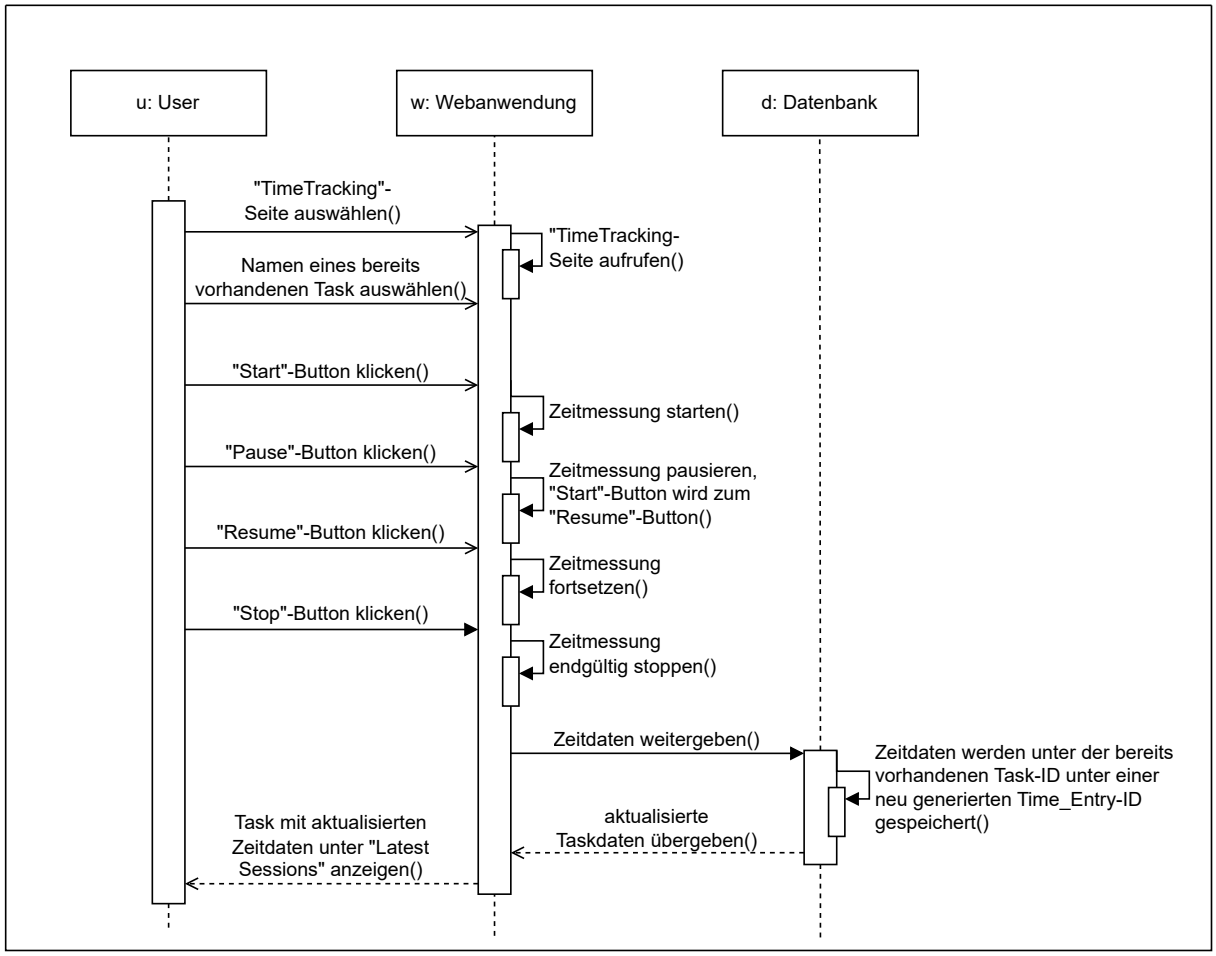

Abbildung 2.4: Sequenzdiagramm F30.2: Zeiterfassung (Start/Stop) bereits existierender Aufgaben

Im Sequenzdiagramm zu der Funktion <F30.2> wird der Prozess des Erfassens von Arbeitszeiten bereits existierender Aufgaben durch Starten, Pausieren und Stoppen eines Timers dargestellt. Auf der „TimeTracking“-Seite kann ein bereits angelegter Task ausgewählt werden. Die Zeitmessung erfolgt genauso wie bei neu angelegten Aufgaben.

Die erfassten Daten werden anschließend an die Datenbank übermittelt, wo der Task anhand seiner Task-ID aufgerufen wird und der neue Zeiteintrag unter einer neu generierten Time_Entry-ID gespeichert wird. Nach erfolgreicher Speicherung werden die Daten an die Webanwendung zurückgegeben, sodass die aktualisierte Benutzeroberfläche den überarbeiteten Task auf der „TimeTracking“-Seite unter „Latest Sessions“ anzeigen kann.

2.4 Analyse von Funktionalität <F40>: Zeiteinträge bearbeiten

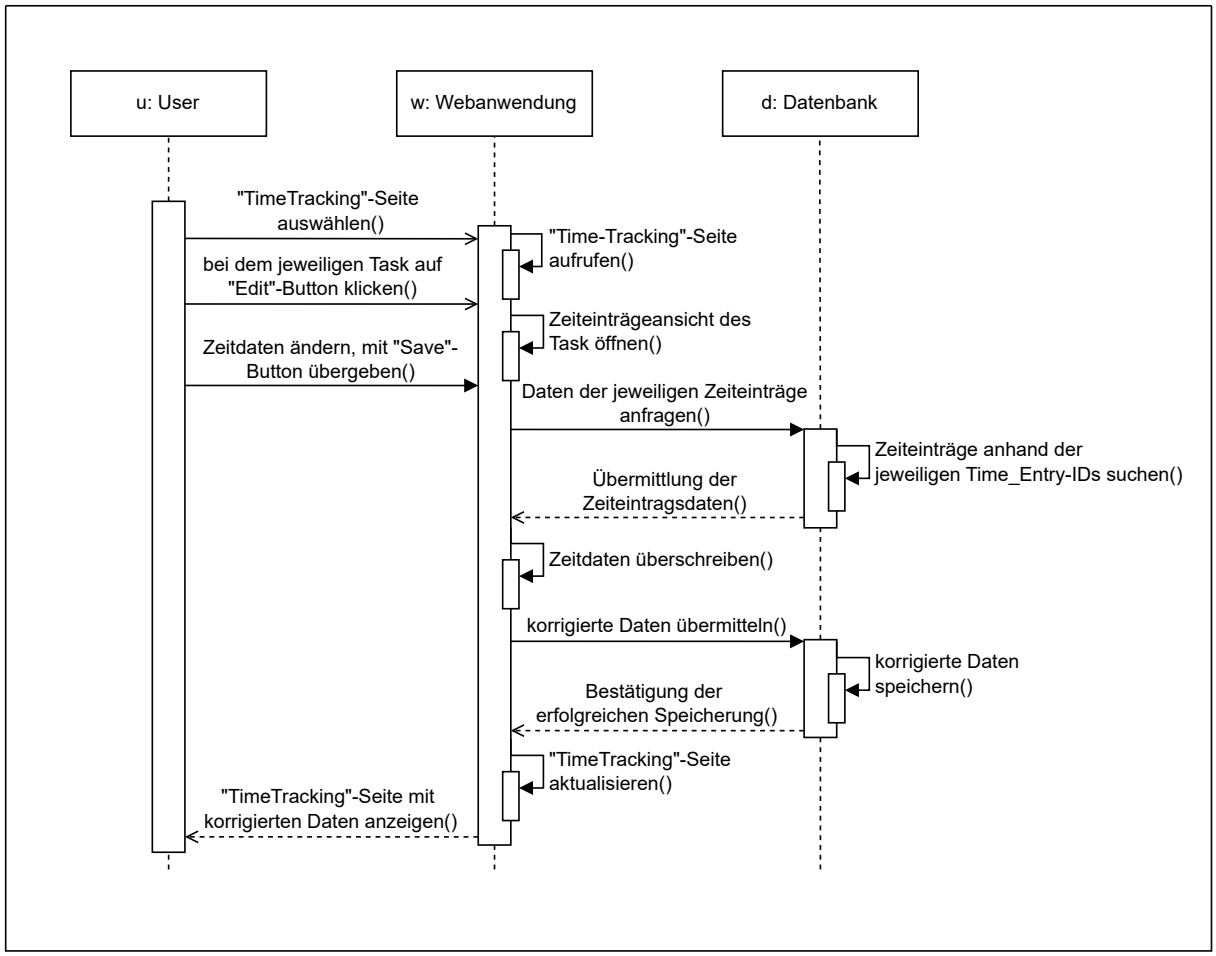

Abbildung 2.5: Sequenzdiagramm F40: Zeiteinträge bearbeiten

Im Sequenzdiagramm zur Funktion <F40> wird der Prozess der manuellen Anpassung oder Korrektur von bereits erfassten Zeitdaten durch den:die Nutzer:in dargestellt. Über die „TimeTracking“-Seite der Webanwendung wird zunächst eine Übersicht aller Tasks angezeigt. Der:Die Nutzer:in kann dort bei einem beliebigen Eintrag über den Button „Edit“ die Zeiteinträgeansicht öffnen. Dort lassen sich die erfassten Zeiteinträge des jeweiligen Tasks durch Veränderung des Datums sowie der Start- und Stopp-Zeit anpassen und neue Zeiteinträge erstellen. Über den „Save“-Button werden diese Änderungen übermittelt.

Die Webanwendung stellt daraufhin eine Anfrage der Zeitdaten der Zeiteinträge, an denen Änderungen vorgenommen werden sollen. Wenn die Datenbank diese Daten anhand der Time_Entry-IDs gefunden hat, übermittelt sie sie an die Webanwendung, die die Daten überschreibt. Die aktualisierten Daten werden anschließend zur Speicherung erneut an die Datenbank übergeben, welche bei erfolgreicher Speicherung eine Bestätigung an die Webanwendung zurückgibt.

Die „TimeTracking“-Seite wird daraufhin aktualisiert und dem:der Nutzer:in mit den korrigierten Daten angezeigt. Sollte es während der Speicherung oder beim Laden der aktualisierten Daten zu Fehlern kommen, werden entsprechende Fehlermeldungen ausgegeben. Auf deren Darstellung im Diagramm wird aus Platzgründen verzichtet.

2.5 Analyse von Funktionalität <F50>: Projektverwaltung

2.5.1 Funktionalität <F50.1>: Projekt erstellen

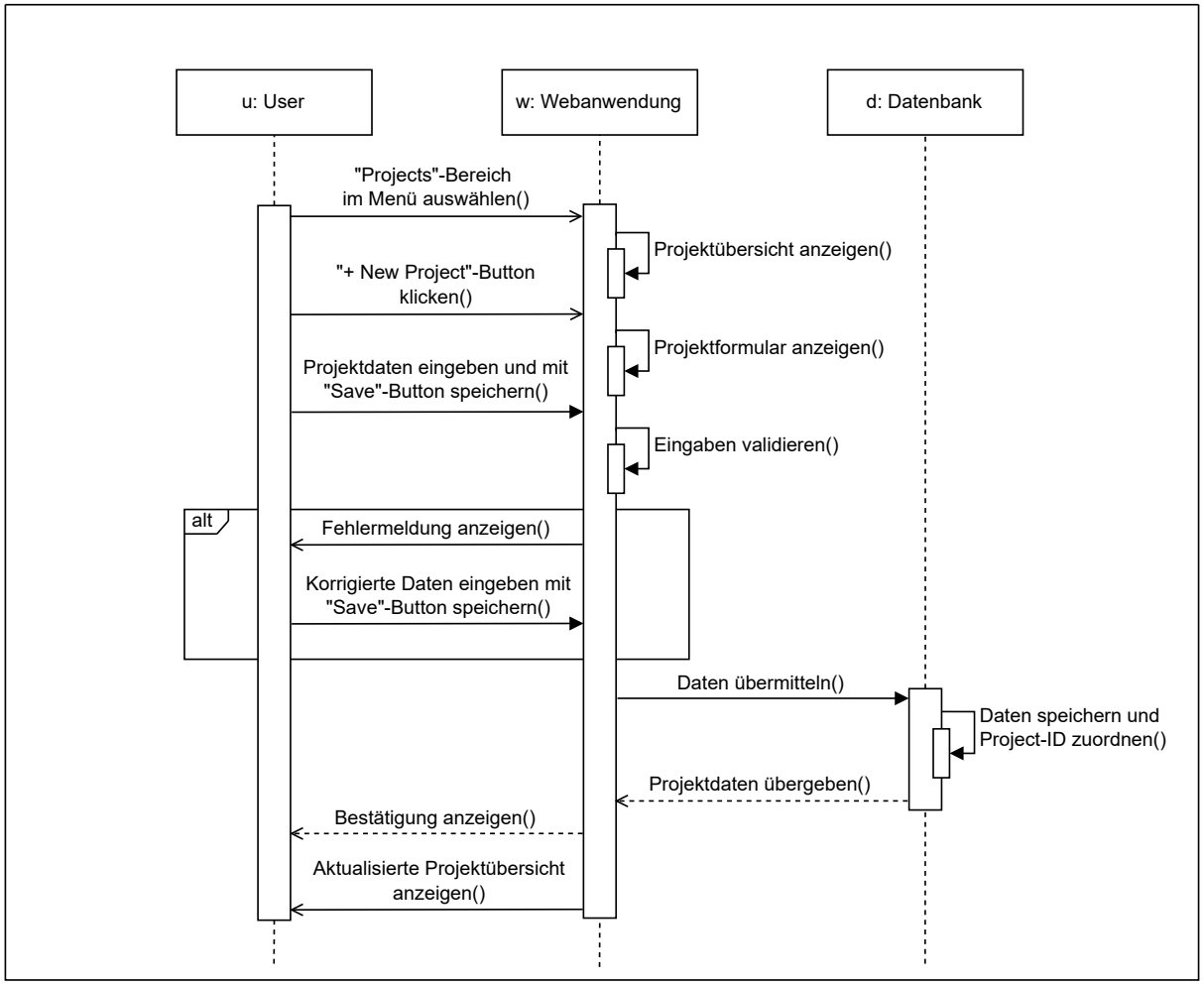

Abbildung 2.6: Sequenzdiagramm F50.1: Projekt erstellen

Die Funktion F50: Projektverwaltung wurde zur besseren Übersichtlichkeit in drei Teilfunktionen unterteilt: Projekte erstellen, Projekte bearbeiten und Projekte löschen.

Das Sequenzdiagramm zu F50.1 zeigt den Ablauf zur Erstellung eines neuen Projekts durch den:die Nutzer:in.

Nach dem Öffnen des Bereichs „Projects“ in der Webanwendung wird die Projektübersicht angezeigt. Über den Button „+ New Project“ kann ein Formular aufgerufen werden, in dem Projektdaten eingegeben werden. Nach dem Speichern werden die Eingaben zunächst von der Anwendung validiert.

Falls die Eingaben ungültig sind, zeigt die Anwendung eine Fehlermeldung an und der:die Nutzer:in kann die Daten korrigieren und erneut speichern (alt-Block). Sind die Eingaben gültig, werden die Projektdaten an die Datenbank übermittelt, dort gespeichert und mit einer eindeutigen ID versehen.

Anschließend zeigt die Anwendung eine Bestätigung und aktualisiert die Projektübersicht.

2.5.2 Analyse von Funktionalität <F50.2>: Projekt bearbeiten

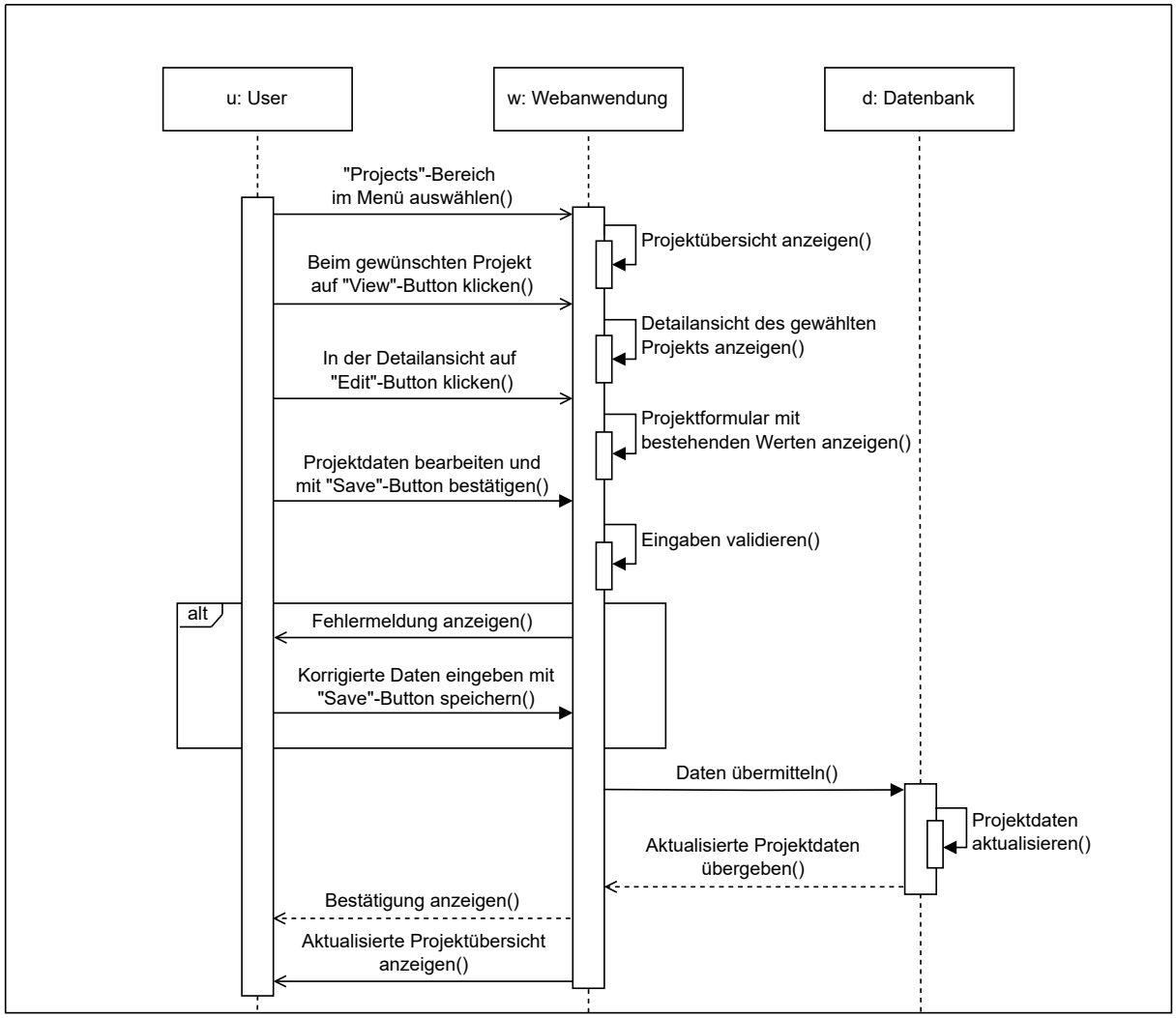

Abbildung 2.7: Sequenzdiagramm F50.2: Projekt bearbeiten

Das Sequenzdiagramm zu F50.2 zeigt den Ablauf zur Bearbeitung eines bestehenden Projekts durch den:die Nutzer:in..

In der Projektübersicht wird das gewünschte Projekt ausgewählt und über den „View“-Button geöffnet. In der Detailansicht kann es über den „Edit“-Button zur Bearbeitung aufgerufen werden. Die Webanwendung zeigt ein Formular mit den vorhandenen Projektdaten an, die angepasst werden können. Nach dem Klick auf den „Save“-Button werden die Eingaben zunächst validiert.

Falls dabei ungültige Eingaben erkannt werden, zeigt die Anwendung eine Fehlermeldung an. Der:die Nutzer:in kann daraufhin die Daten korrigieren und den Speichervorgang erneut auslö-

sen (alt-Block). Sind die Eingaben gültig, werden die überarbeiteten Daten an die Datenbank übermittelt und dort gespeichert.

Anschließend zeigt die Webanwendung eine Bestätigung sowie die aktualisierte Projektübersicht.

2.5.3 Analyse von Funktionalität <F50.3>: Projekt löschen

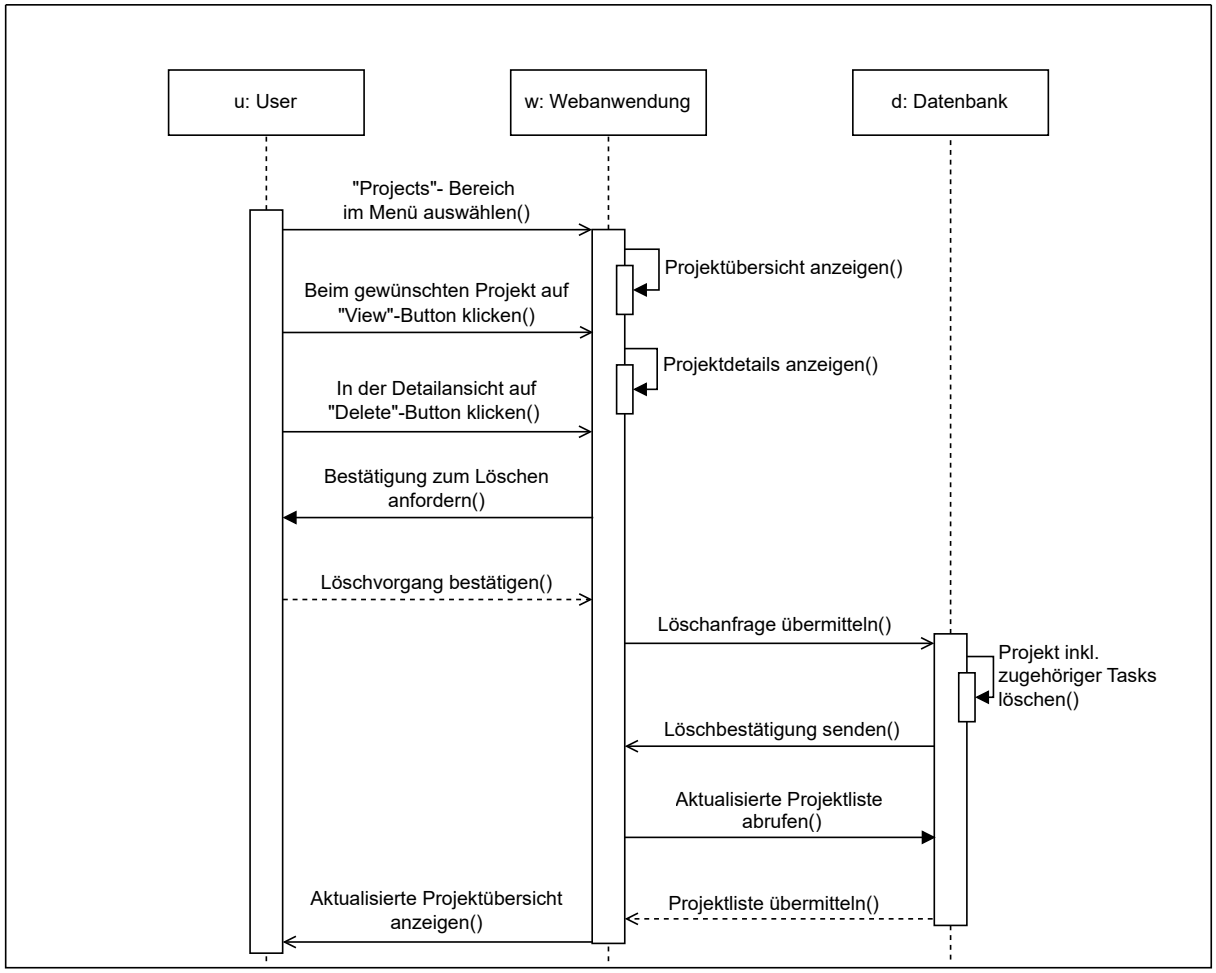

Abbildung 2.8: Sequenzdiagramm F50.3: Projekt löschen

Das Sequenzdiagramm zu F50.3 zeigt den Ablauf zum Löschen eines Projekts durch den:die Nutzer:in.

Nach Auswahl des gewünschten Projekts in der Übersicht wird in der Detailansicht der Löschvorgang über den „Delete“-Button gestartet. Die Anwendung fordert eine Bestätigung an, bevor die Löschanfrage an die Datenbank übermittelt wird. Dort werden das Projekt und die zugehörigen Tasks entfernt.

Abschließend wird die Projektübersicht aktualisiert angezeigt.

2.6 Analyse von Funktionalität <F60>: Taskverwaltung

2.6.1 Analyse von Funktionalität <F60.1>: Task erstellen auf der Projektseite

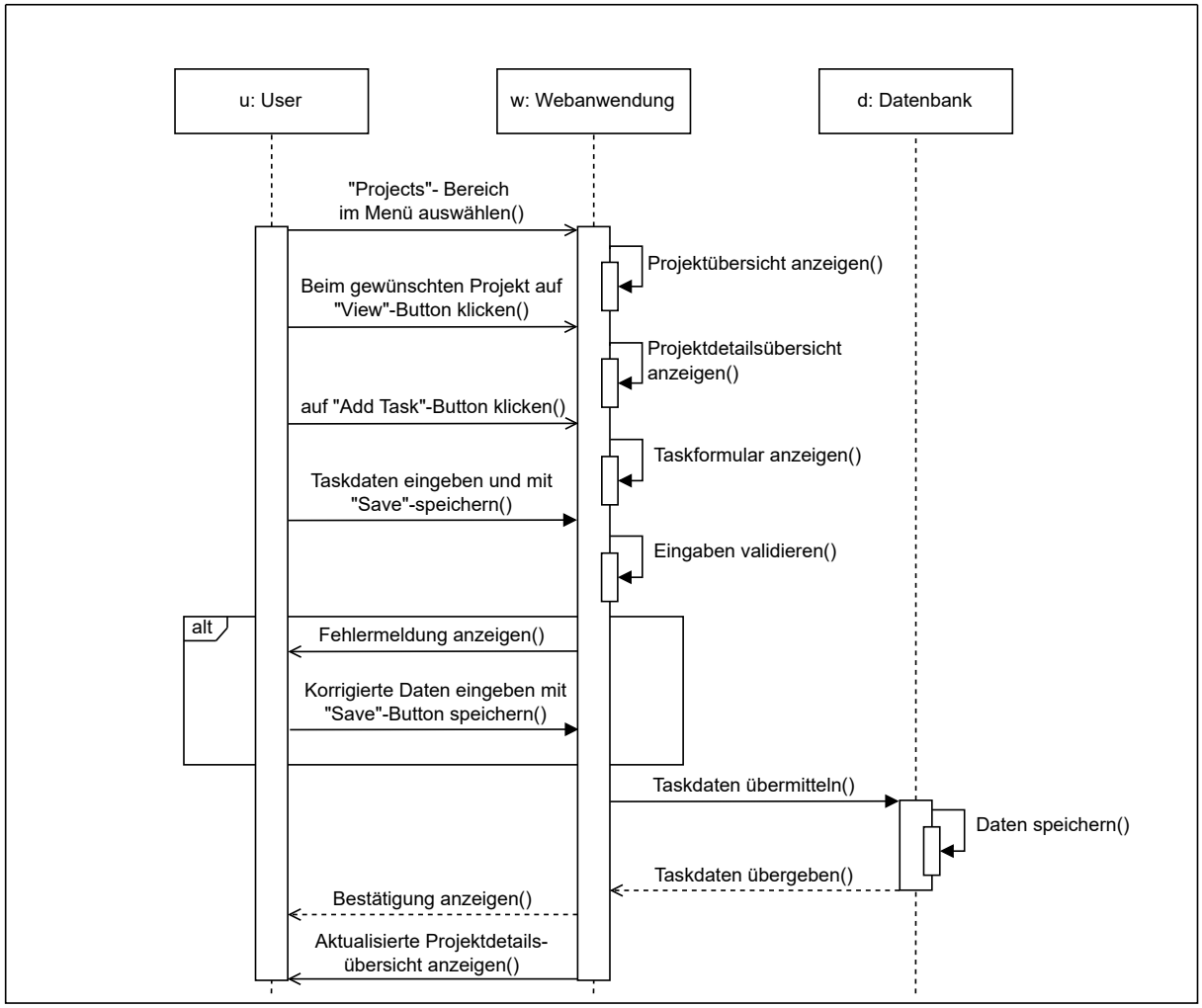

Abbildung 2.9: Sequenzdiagramm F60.1: Task erstellen auf der Projektseite

Die Funktion F60: Taskverwaltung wurde zur besseren Übersichtlichkeit in drei Teilfunktionen unterteilt: Tasks erstellen, Tasks bearbeiten und Tasks löschen.

Das Sequenzdiagramm zu F60.1 zeigt den Ablauf der Erstellung eines neuen Tasks innerhalb eines bestehenden Projekts durch den:die Nutzer:in.

Nach dem Öffnen des Bereichs „Projects“ klickt der:die Nutzer:in beim gewünschten Projekt auf „Add Task“. Ein Eingabeformular erscheint, in dem die Taskdaten (z.B. Name, Beschreibung, Kategorie, Fälligkeitsdatum) eingegeben und mit „Save“ bestätigt werden. Nach dem Speichern werden die Eingaben zunächst von der Anwendung validiert.

Wenn die Eingaben ungültig sind, zeigt die Anwendung eine Fehlermeldung an. Der:die Nutzer:in kann die Daten daraufhin korrigieren und den Speichervorgang erneut auslösen (alt-Block). Bei gültigen Eingaben werden die Projektdaten an die Datenbank übermittelt, dort gespeichert und mit einer eindeutigen ID versehen.

Anschließend zeigt die Anwendung eine Bestätigung sowie die aktualisierte Projektübersicht.

2.6.2 Analyse von Funktionalität <F60.2>: Task bearbeiten

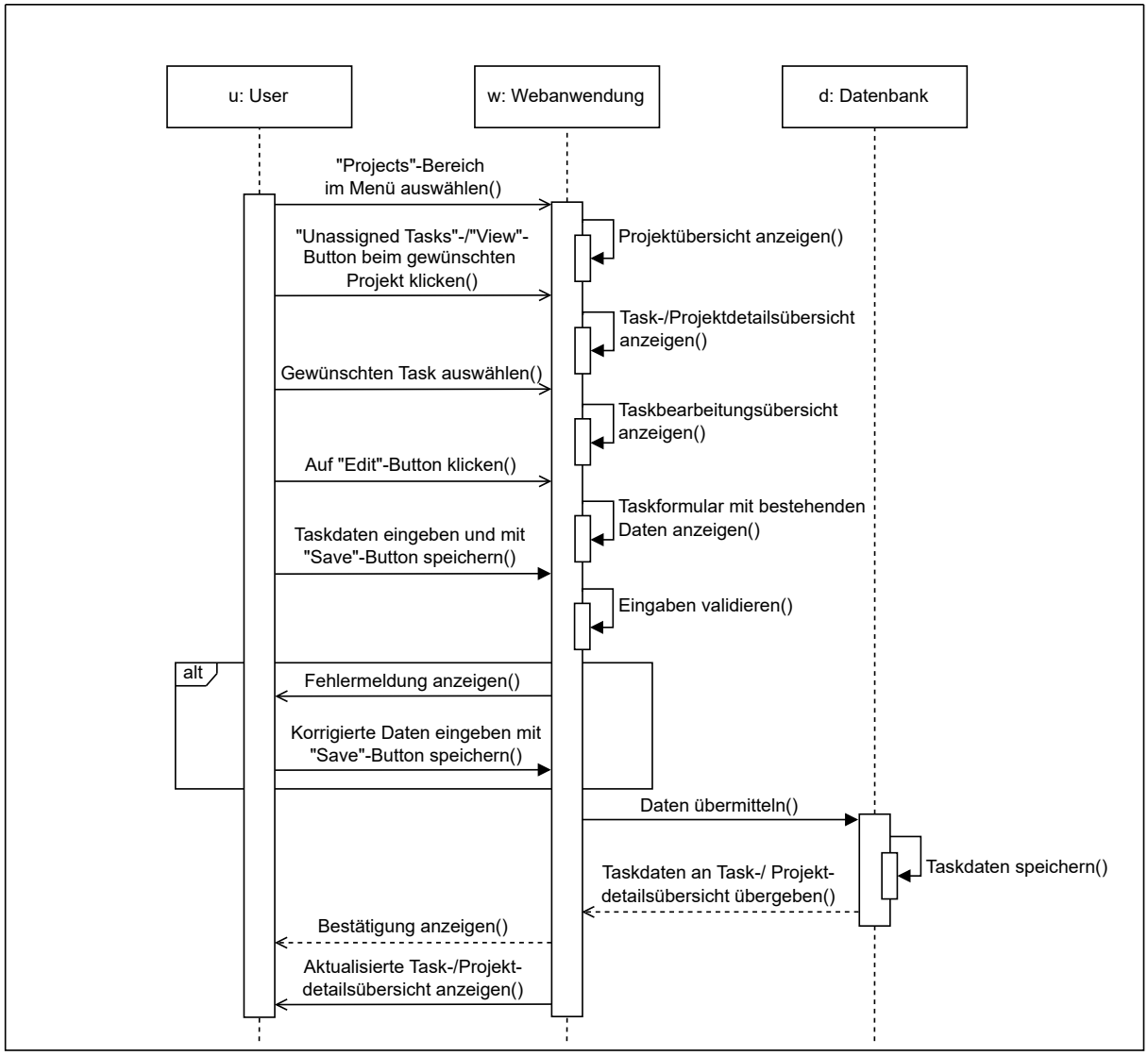

Abbildung 2.10: Sequenzdiagramm F60.2: Task bearbeiten

Das Sequenzdiagramm zu F60.2 beschreibt den Ablauf der Bearbeitung eines bestehenden Tasks durch den:die Nutzer:in.

Die Bearbeitung von Tasks erfolgt ausschließlich über die Projektseite im Bereich „Projects“ der Webanwendung. Der:die Nutzer:in wählt entweder einen nicht zugeordneten Task aus der Übersicht oder öffnet die Detailansicht eines Projekts über den „View“-Button, um einen zugehörigen Task zu bearbeiten.

Nach dem Klick auf „Edit“ erscheint ein Formular mit den vorhandenen Taskdaten, die angepasst werden können (z.B. Name, Beschreibung, Kategorie, Fälligkeitsdatum oder Projektzuordnung).

Nach dem Speichern werden die Eingaben validiert:

Wenn die Eingaben ungültig sind, zeigt die Anwendung eine Fehlermeldung an. Der:die Nutzer:in kann die Angaben daraufhin korrigieren und den Speichervorgang erneut auslösen (alt-Block). Bei gültigen Eingaben werden die Taskdaten an die Datenbank übermittelt und gespeichert.

Wenn eine Projektzuordnung geändert wurde, wird auch der zugehörige Projektdatensatz aktualisiert.

Anschließend wird die aktualisierte Task-/Projektdetailansicht angezeigt.

2.6.3 Analyse von Funktionalität <F60.3>: Task löschen

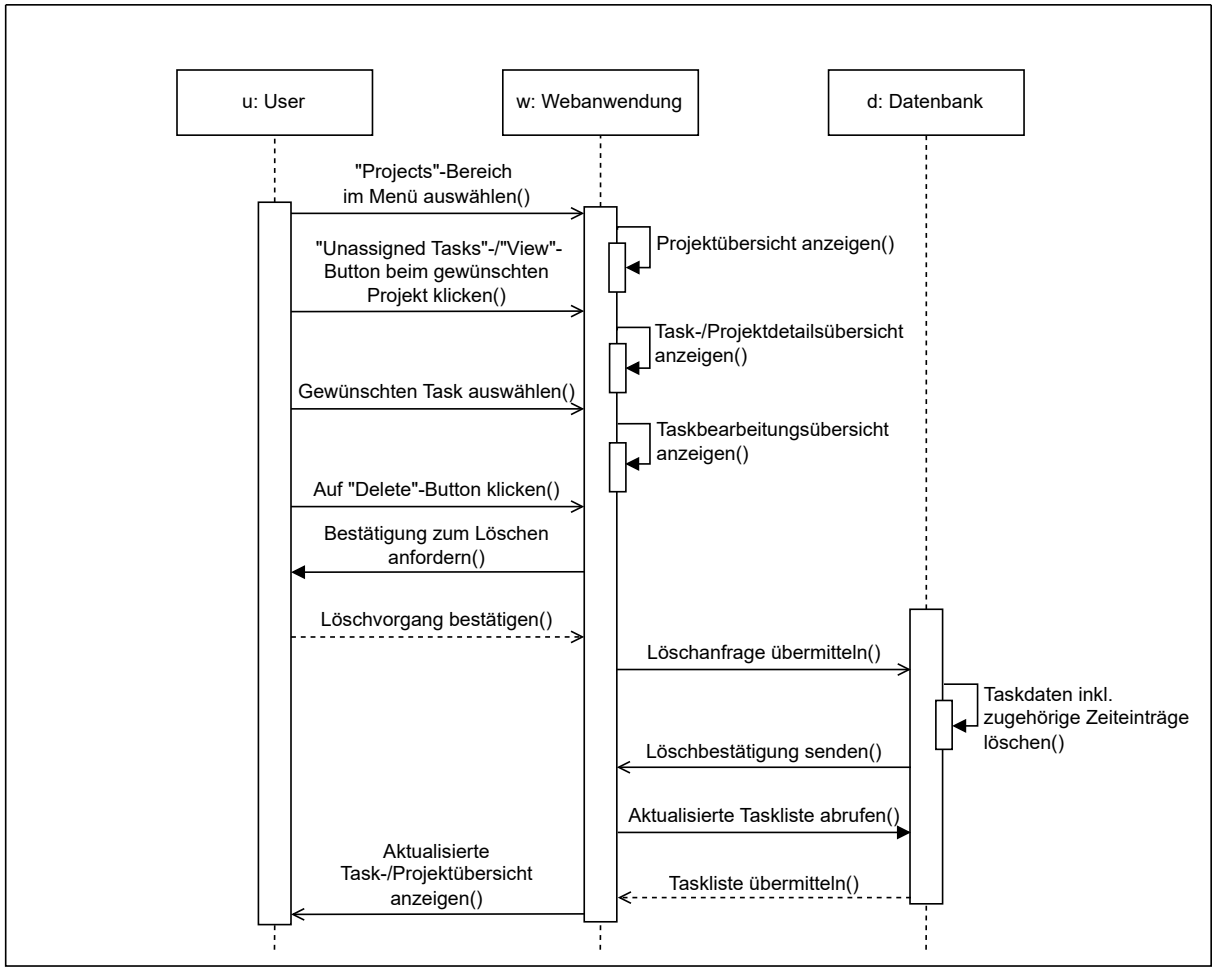

Abbildung 2.11: Sequenzdiagramm F60.3: Task löschen

Das Sequenzdiagramm zu F60.3 beschreibt den Ablauf des Löschens eines bestehenden Tasks durch den:die Nutzer:in.

Die Löschung erfolgt über die Projektseite. Nach dem Öffnen des Bereichs „Projects“ wird die Projektübersicht angezeigt. Dort hat der:die Nutzer:in zwei Möglichkeiten: Entweder wird der Task direkt im Abschnitt „Unassigned Tasks“ gefunden oder durch Klick auf den „View“-Button eines Projekts die zugehörige Projektdetailansicht geöffnet, in der ein projektbezogener Task ausgewählt werden kann. Beim gewünschten Task klickt der:die Nutzer:in auf „Delete“. Daraufhin wird eine Bestätigung angefordert. Nach Zustimmung übermittelt die Webanwendung die Löschanfrage an das Backend. Die Datenbank entfernt den Task sowie alle zugehörigen Zeiteinträge. Nach erfolgreicher Löschung wird eine Bestätigung gesendet, und die aktualisierte Taskliste wird angezeigt.

2.7 Analyse von Funktionalität <70>: Export von Zeitdaten

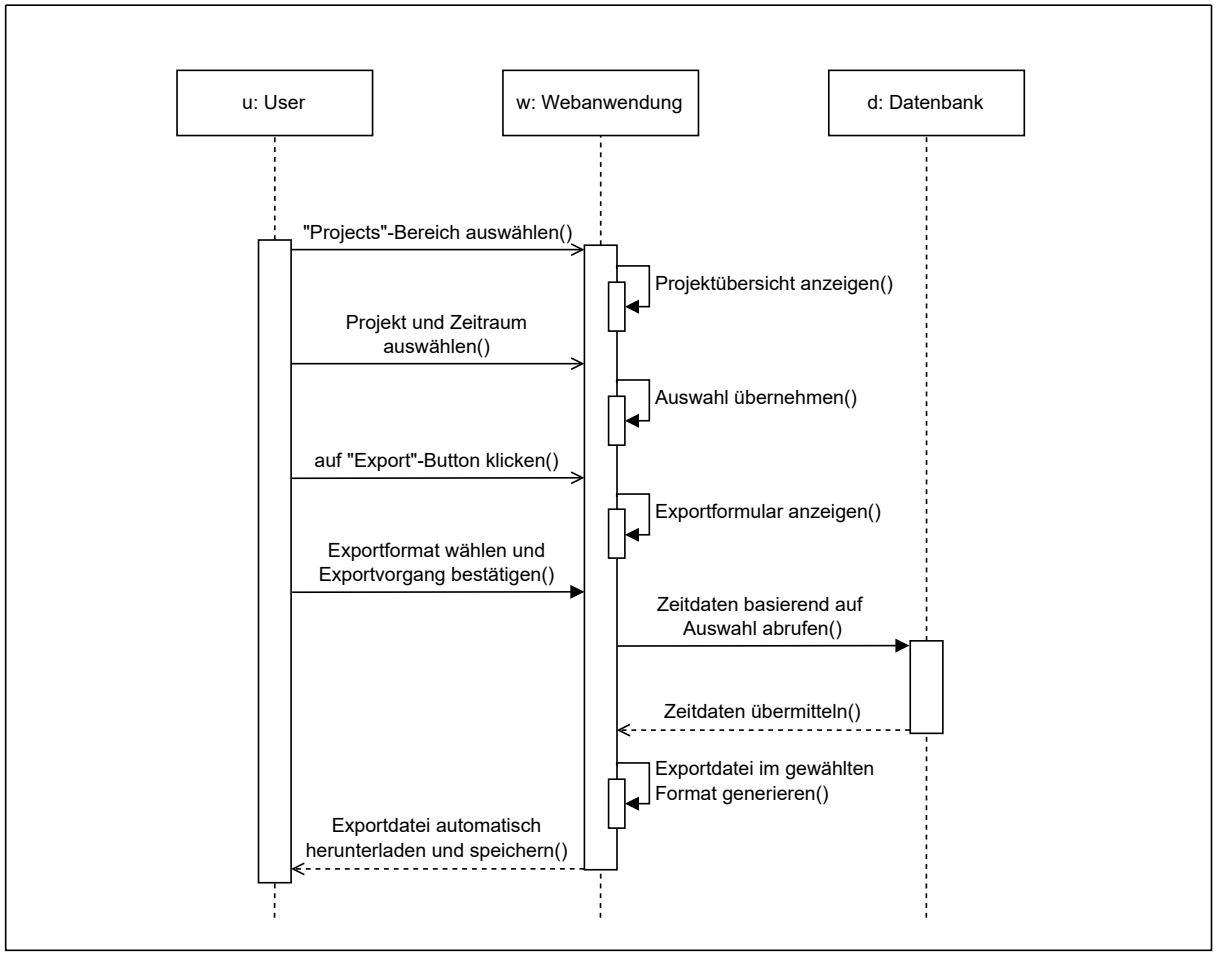

Abbildung 2.12: Sequenzdiagramm F70: Export von Zeitdaten

Das Sequenzdiagramm zu F70 zeigt den Ablauf zum Exportieren von erfassten Zeitdaten durch den:die Nutzer:in.

Zu Beginn wählt der:die Nutzer:in den Bereich „Projects“ in der Anwendung aus, woraufhin die Webanwendung die Projektübersicht lädt. Dort wird ein bestimmtes Projekt sowie ein Zeitraum ausgewählt. Diese Auswahl wird gespeichert und durch einen Klick auf den Button „Export“ wird der Exportvorgang gestartet. Die Webanwendung sendet automatisch eine Anfrage an das Backend, das die relevanten Zeitdaten abruft und eine Datei im gewählten Format (z.B. CSV oder PDF) generiert. Diese Datei wird direkt durch den Browser als Download ausgelöst und im Dateisystem des:der Nutzer:in gespeichert.

Diese Funktion ermöglicht es, Zeiteinträge strukturiert außerhalb der Anwendung zu sichern oder weiterzuverarbeiten.

2.8 Analyse von Funktionalität <80>: Kalenderansicht zur Anzeige projektbezogener Zeiteinträge

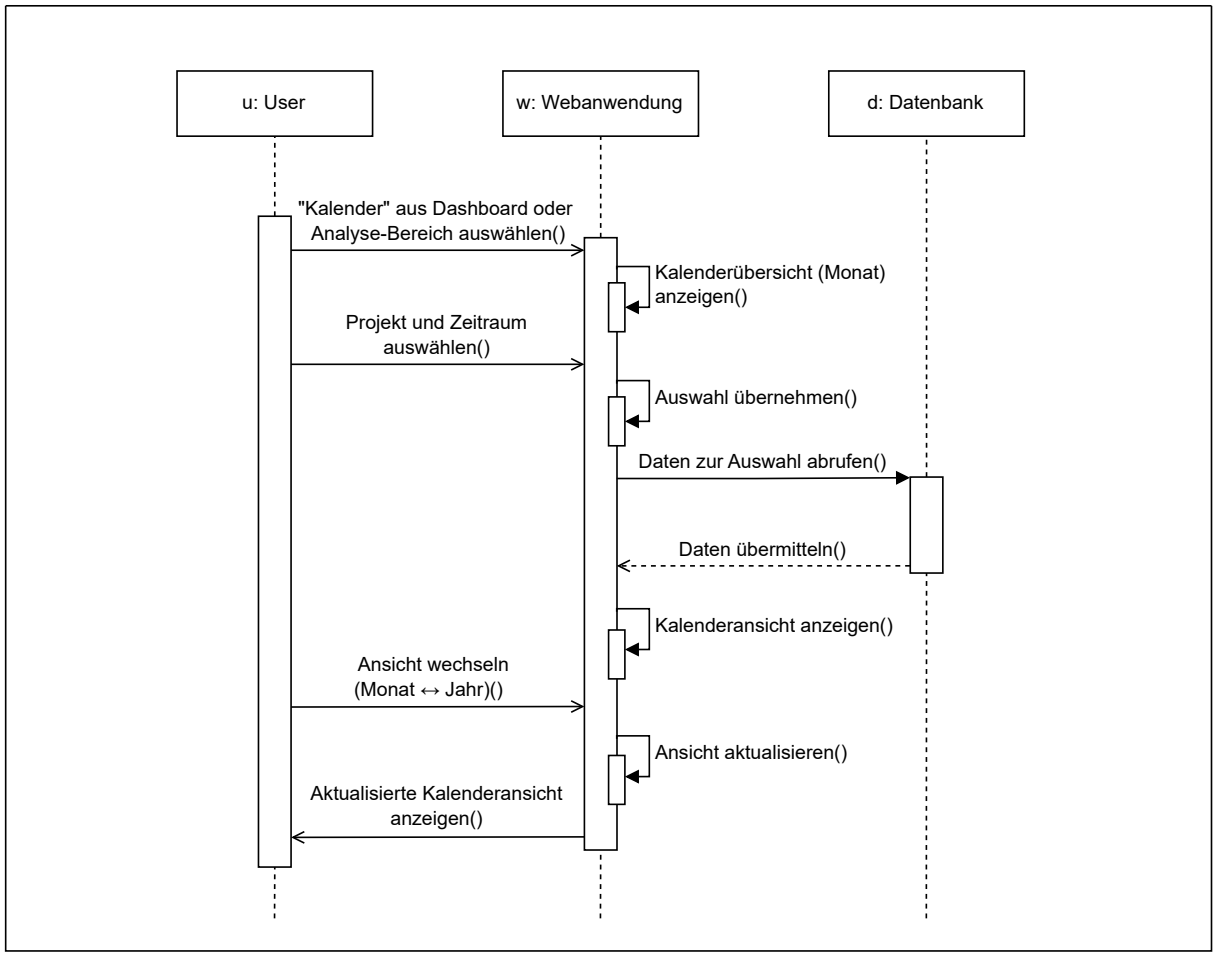

Abbildung 2.13: Sequenzdiagramm F80: Kalenderansicht zur Anzeige projektbezogener Zeiteinträge

Das Sequenzdiagramm zu F80 beschreibt den Ablauf beim Aufrufen und Nutzen der Kalenderansicht durch den:die Nutzer:in.

Der:Die Nutzer:in öffnet die Kalenderansicht über das Dashboard oder den Analysebereich. Die Webanwendung lädt automatisch die Monatsübersicht des Kalenders. Anschließend kann ein bestimmtes Projekt sowie ein gewünschter Zeitraum ausgewählt werden. Die Anwendung übernimmt die Auswahl und ruft die entsprechenden Daten aus der Datenbank ab. Nach erfolgreicher Übermittlung werden die relevanten Inhalte in der Kalenderansicht dargestellt. Optional kann der:die Nutzer:in in die Jahresansicht wechseln, woraufhin die Darstellung aktualisiert wird.

Diese Funktion dient der übersichtlichen Visualisierung dokumentierter Aktivitäten und unterstützt die zeitbezogene Auswertung und Planung.

2.9 Analyse von Funktionalität <F90>: Fortschritts- und Vergleichsdiagramme

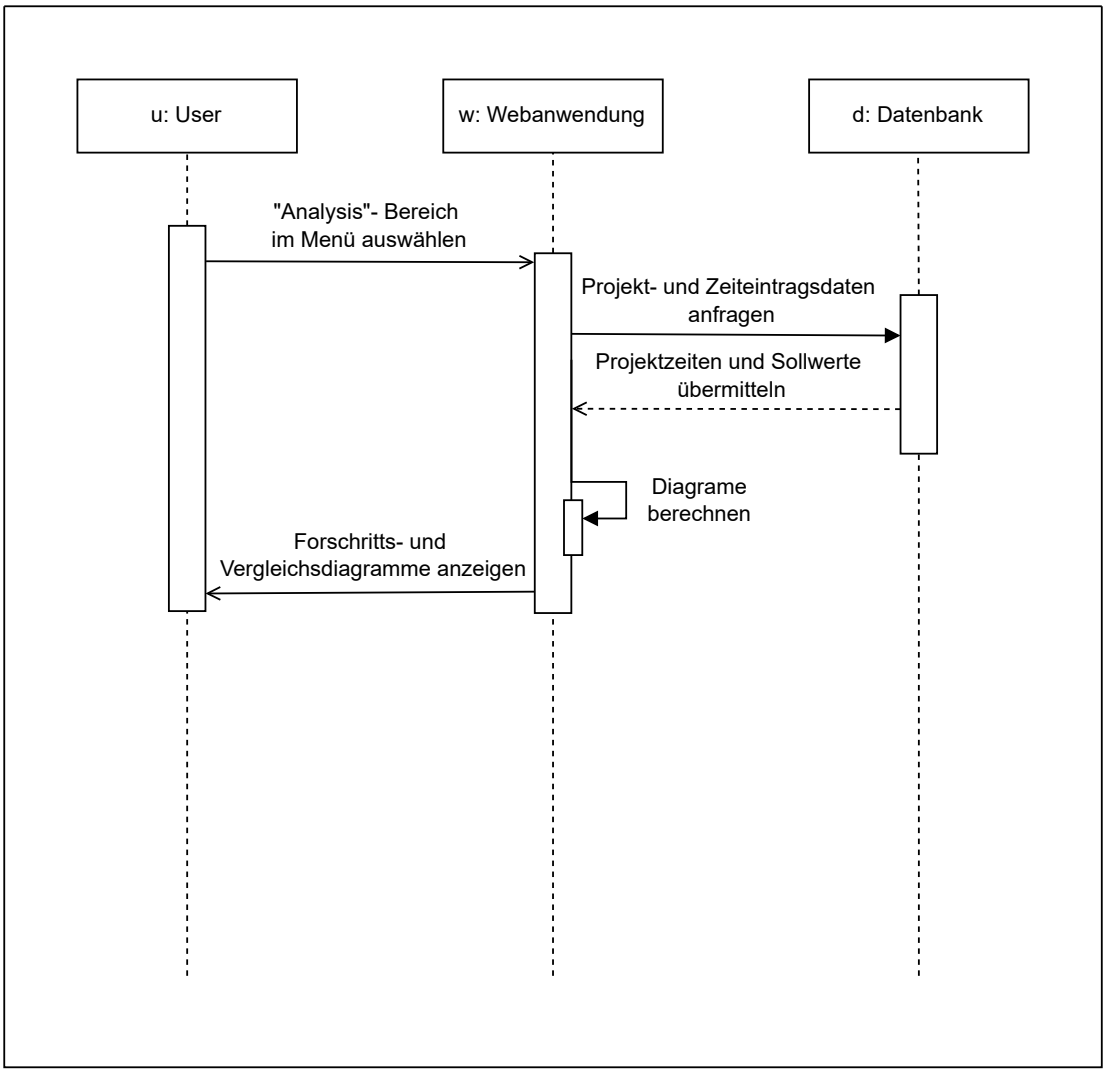

Abbildung 2.14: Sequenzdiagramm F90: Fortschritts- und Vergleichsdiagramme

Im Sequenzdiagramm zur Funktion <F90> wird der Ablauf zur Visualisierung von Projektfortschritt und Ist-Soll-Vergleich gezeigt. Nach Anmeldung und Aufruf des Analysebereichs wählt der:die Nutzer:in ein oder mehrere Projekte aus. Das System berechnet relevante Prozentwerte und erstellt Fortschritts- sowie Vergleichsdiagramme. Diese zeigen den Fortschritt einzelner Projekte und deren Zielerreichung im zeitlichen Vergleich. Bei fehlenden Sollwerten erscheint ein entsprechender Hinweis.

2.10 Analyse von Funktionalität <F100>: Teamverwaltung

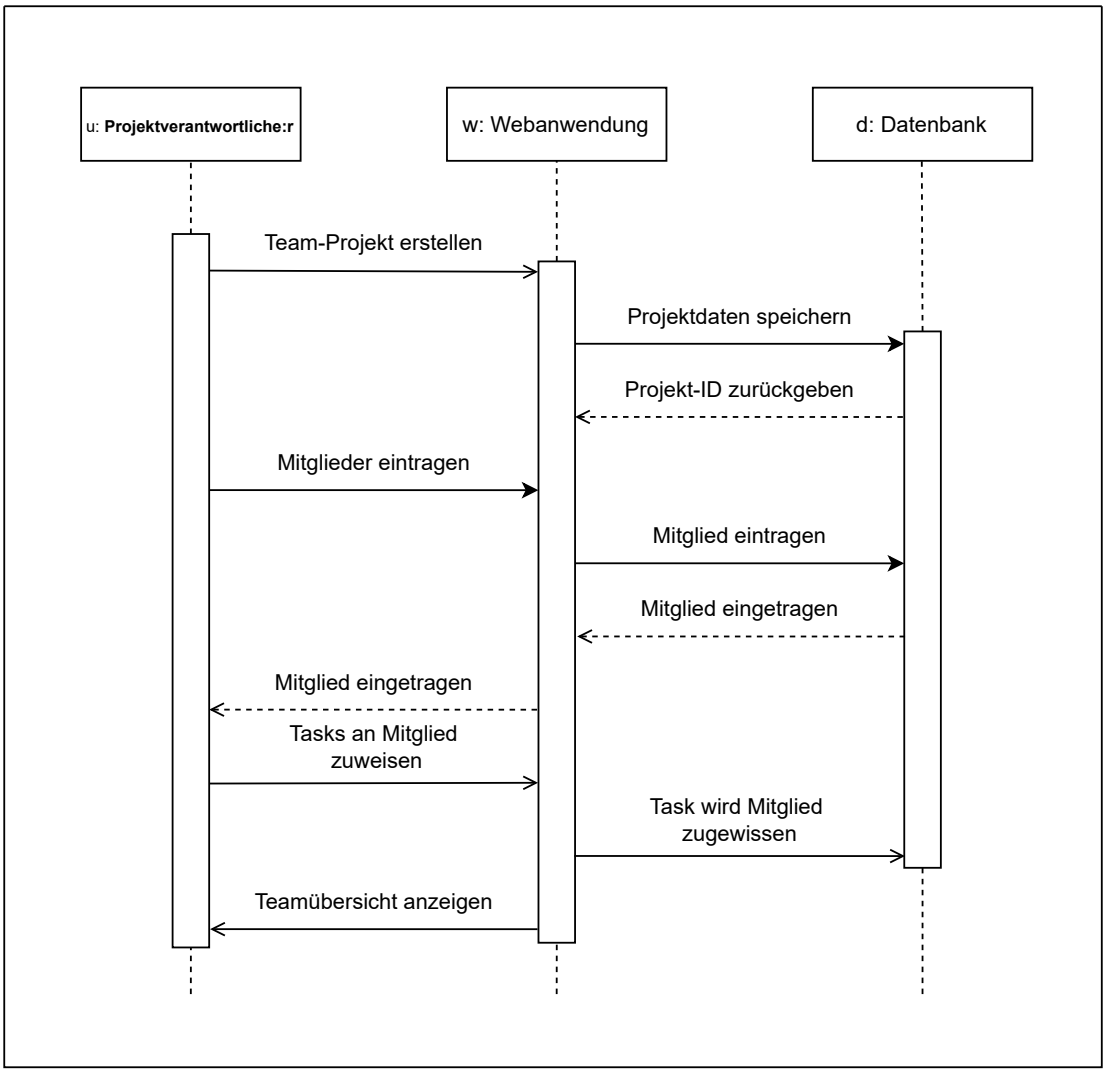

Abbildung 2.15: Sequenzdiagramm F100: Teamverwaltung

Im Sequenzdiagramm zur Funktion <F100> wird die Verwaltung von Teamprojekten dargestellt. Der:die Projektverantwortliche erstellt ein Projekt vom Typ „Team“ und weist anschließend Aufgaben gezielt einzelnen Teammitgliedern zu. Wird versucht, Aufgaben an nicht vorhandene oder nicht berechtigte Nutzer:innen zuzuweisen, erscheint eine entsprechende Fehlermeldung.

2.11 Analyse von Funktionalität <F110>: Benutzeroberfläche – Sprache

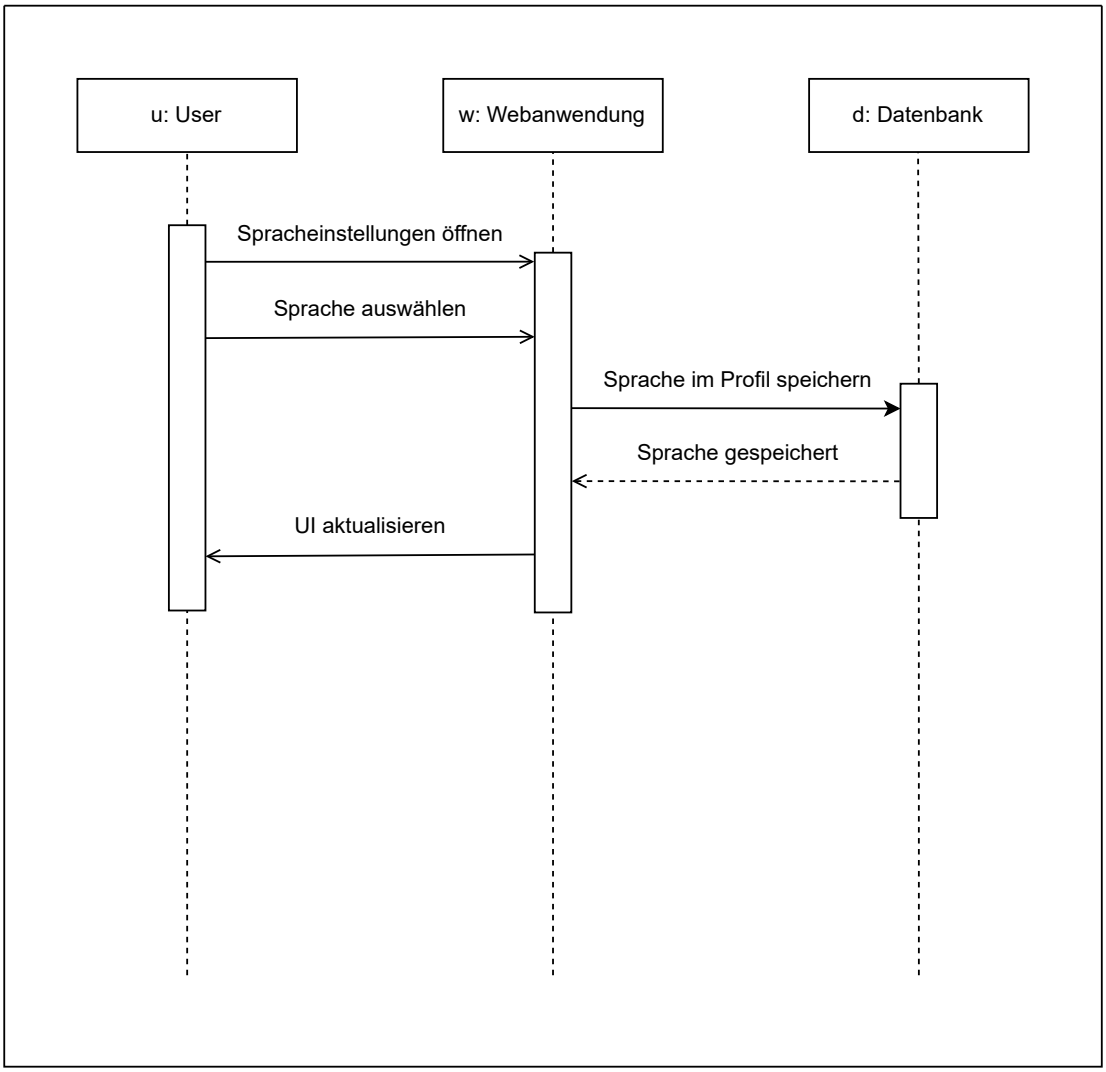

Abbildung 2.16: Sequenzdiagramm F110: Benutzeroberfläche – Sprache

Im Sequenzdiagramm zur Funktion <F110> wird der Ablauf zur Änderung der Spracheinstellung der Benutzeroberfläche dargestellt. Der:die Nutzer:in öffnet entweder das Einstellungsmenü oder nutzt einen sichtbaren Sprachumschalter. Nach Auswahl der gewünschten Sprache (standardmäßig Englisch) wird diese Einstellung im Profil der:des Nutzer:in gespeichert. Bei technischen Problemen bleibt die Standardsprache erhalten oder es erscheint eine Fehlermeldung. Zukünftig sind weitere Sprachen (z.B. Deutsch oder Spanisch) vorgesehen.

2.12 Analyse von Funktionalität <F120>: Plattformkompatibilität (Chrome)

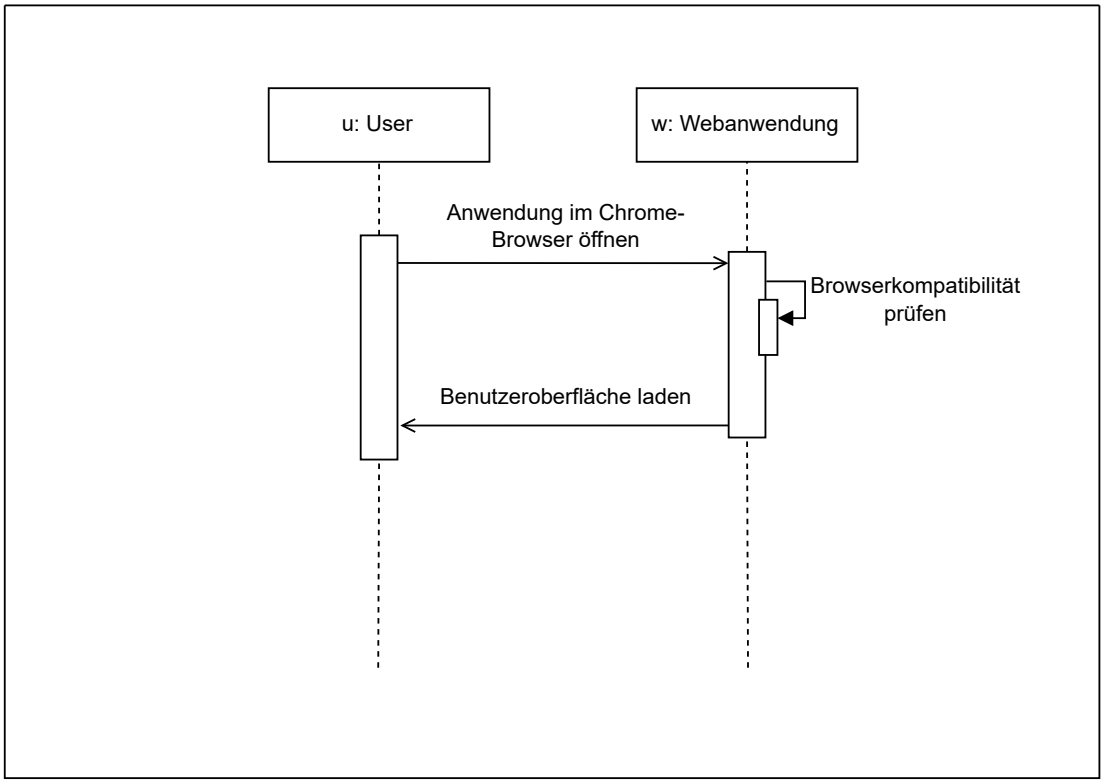

Abbildung 2.17: Sequenzdiagramm F120: Plattformkompatibilität (Chrome)

Im Sequenzdiagramm zu der Funktion <F120> wird die plattformunabhängige Nutzung der Webanwendung in Google Chrome dargestellt. Der:Die Nutzer:in öffnet die Anwendung im Chrome-Browser. Die Benutzeroberfläche wird daraufhin korrekt gerendert und alle Funktionen stehen uneingeschränkt zur Verfügung. Bei veralteten oder inkompatiblen Browsern kann es zu Einschränkungen kommen. Eine Erweiterung zur Unterstützung weiterer Browser (z.B. Firefox, Safari, Edge) ist möglich.

2.13 Analyse von Funktionalität <F130>: Datenpersistenz in SQLite

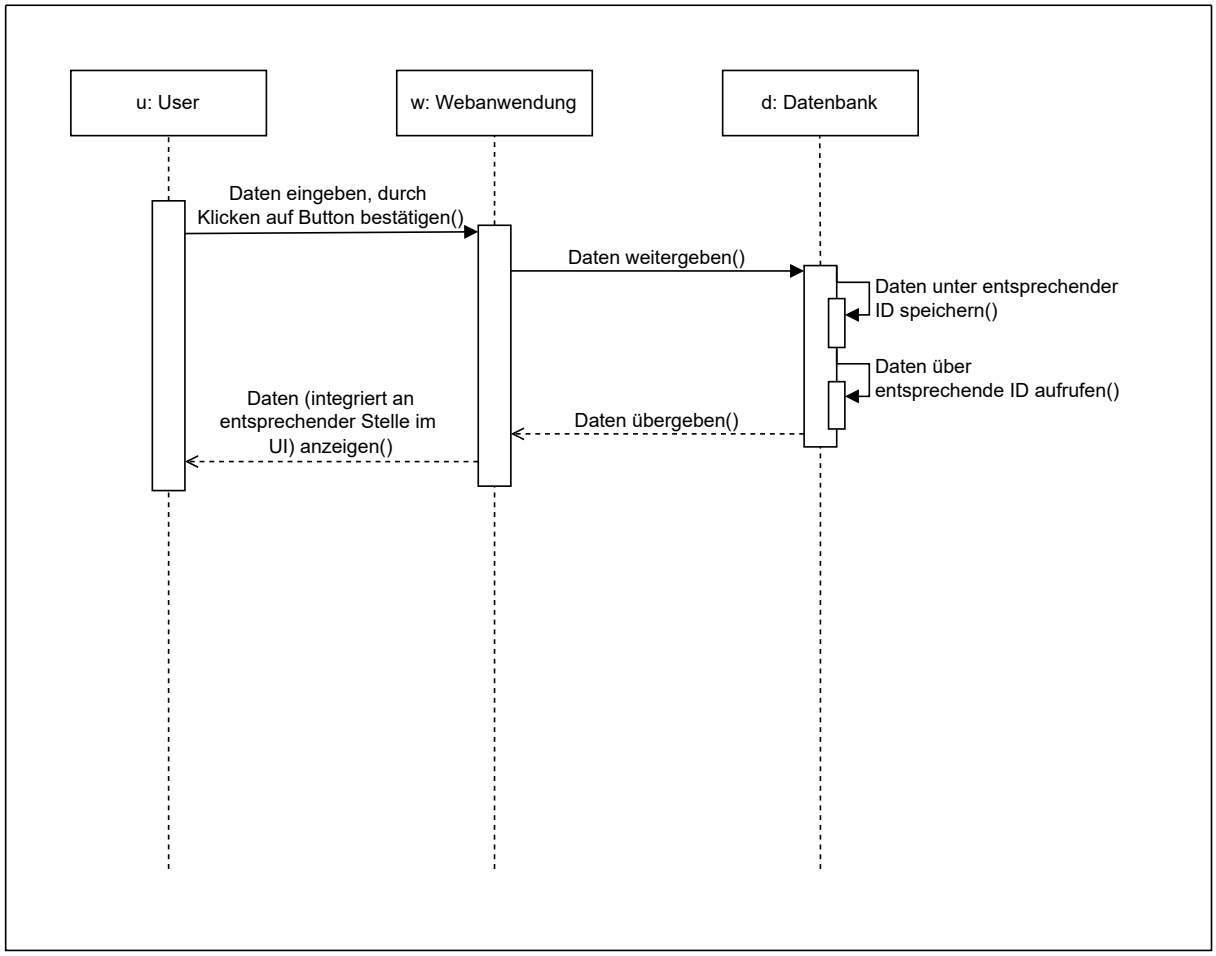

Abbildung 2.18: Sequenzdiagramm F130: Datenpersistenz in SQLite

Im Sequenzdiagramm zu der Funktion <F130> wird der Prozess der dauerhaften Speicherung aller erfassten Zeitdaten in einer zentralen SQLite-Datenbank dargestellt. Sobald Daten hinterlegt werden sollen, werden sie auf der Webseite eingetragen und durch das Klicken eines Buttons bestätigt. Daraufhin werden die Daten an die Datenbank übermittelt und im Anschluss unter einer eindeutigen ID gespeichert. Bereits vorhandene Einträge mit identischer ID werden in diesem Fall überschrieben, sodass Änderungen an bestehenden Daten zuverlässig berücksichtigt werden können. Die dauerhaft gespeicherten Daten können gezielt über ihre ID aus der Datenbank abgerufen werden und sind dementsprechend bei zukünftigen Zugriffen verfügbar. Sie werden anschließend an die Webanwendung zurückgegeben und dort in der Benutzeroberfläche an der vorgesehenen Stelle integriert. Durch diesen Ablauf wird eine konsistente und redundanzfreie Datenspeicherung gewährleistet.

3 Resultierende Softwarearchitektur

Dieser Abschnitt bietet eine strukturierte Darstellung der zentralen Komponenten des zu entwickelnden Systems. Es wird erläutert, wie diese Komponenten miteinander interagieren und welche spezifischen Funktionen sie erfüllen. Die Struktur des Systems wird durch ein Komponentendiagramm veranschaulicht. Dabei stehen sowohl die internen Abläufe als auch die Schnittstellen innerhalb des Systems im Fokus. Abschließend wird die Einbindung der einzelnen Komponenten in das Gesamtsystem nachvollziehbar dokumentiert.

3.1 Komponentenspezifikation

Die im Komponentendiagramm (Abbildung 3.1) dargestellten Komponenten des Systems werden in diesem Abschnitt anschließend einzeln beschrieben.

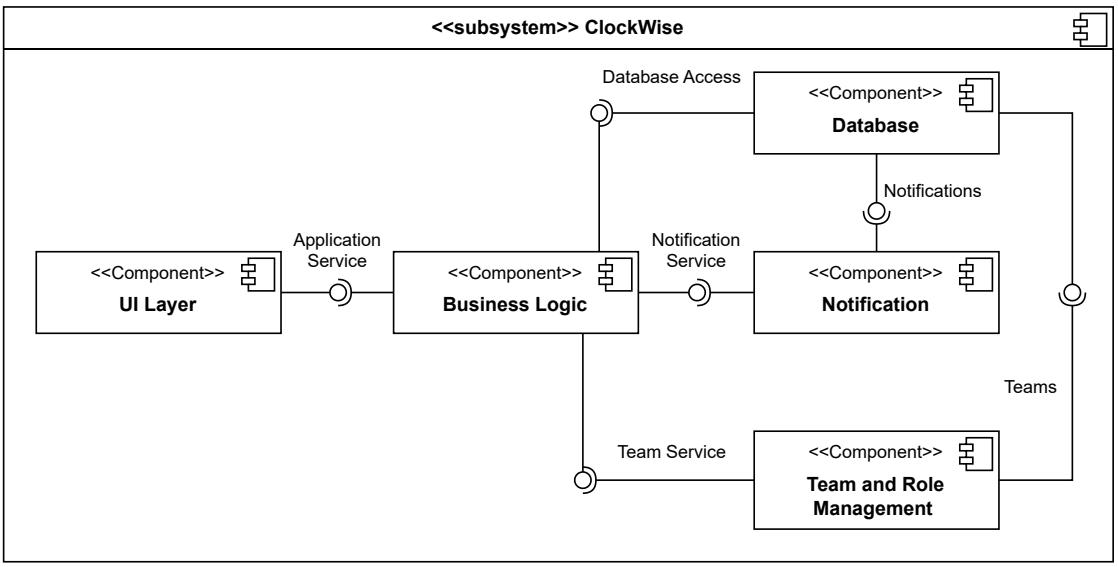

Abbildung 3.1: Komponentendiagramm.

Komponente $\langle C10 \rangle$: <UI Layer>
Die Komponente UI Layer stellt die grafische Interaktion mit dem System bereit und dient somit als Schnittstelle zu den Nutzer:innen. Sie nimmt dementsprechend Benutzereingaben entgegen und übermittelt diese an die Geschäftslogik. Zusätzlich validiert sie Eingaben clientseitig und sorgt für eine benutzerfreundliche und responsive Darstellung.

Komponente $\langle C20 \rangle$: <Business Logic>
Die Komponente Business Logic verarbeitet Benutzereingaben gemäß den definierten Regeln und Prozessen. Sie steuert Abläufe und koordiniert den Zugriff auf Datenbank, Rollenverwaltung und Benachrichtigungen.

Komponente $\langle C30 \rangle$: <Database>
Die Database übernimmt die strukturierte Speicherung aller relevanten Informationen, insbesondere zu Nutzern, Teams und Rollen, Projekten, Kategorien, Benachrichtigungen, Aufgaben sowie deren Zeiteinträgen. Darüber hinaus stellt sie die Daten effizient für Abfragen und Auswertungen bereit.

Komponente $\langle C40 \rangle$: <Team and Role Management>
Die Komponente Team and Role Management regelt den Zugang zum System über Benutzerkonten, Rollen und Berechtigungen. Sie erlaubt die Abbildung von Teams und legt Sichtbarkeiten und Aktionsrechte fest.

Komponente $\langle C50 \rangle$: <Notification>
Die Komponente Notification versendet automatisierte Systemnachrichten an Benutzer. Sie wird durch definierte Ereignisse ausgelöst, wie etwa durch die erfolgreiche Erstellung von Projekten, Aufgaben oder Zeiteinträgen – ebenso durch die Analyse der gesetzten Zeitziele.

3.2 Schnittstellenspezifikation

In diesem Abschnitt werden die Schnittstellen der zuvor beschriebenen Komponenten detailliert beschrieben. Dabei werden die von den Komponenten bereitgestellten Operationen dokumentiert, wie sie im Komponentendiagramm dargestellt sind.

Schnittstelle $\langle I10 \rangle$**: <Application Service>**

Operation	**Beschreibung**
dict register_user (str name, str email, str password)	Registriert eine:n neue:n Nutzer:in mit den angegebenen Daten. Gibt Status und ggf. Fehlermeldungen zurück.
dict login_user (str email, str password)	Authentifiziert eine:n Nutzer:in. Gibt bei Erfolg einen Authentifizierungstoken zurück.
bool reset_password (str email)	Sendet einen Link zum Zurücksetzen des Passworts an die angegebene E-Mail-Adresse.
bool submit_time_entry (int task_id, datetime start, datetime end)	Speichert einen neuen Zeiteintrag für eine bestimmte Task.
dict create_project (str name, str type, datetime deadline)	Erstellt ein neues Projekt und gibt die Projekt-ID zurück.
list[dict] get_user_projects (int user_id)	Ruft alle Projekte ab, bei denen der Benutzer teilnimmt oder die er selbst erstellt hat.
file export_time_data (int user_id, int project_id, str format)	Exportiert Zeitdaten eines Projekts im gewünschten Format (CSV/PDF).
list[dict] get_progress_diagrams (int user_id)	Liefert Daten zur Visualisierung von Fortschritt und Ist-Soll-Vergleich für Projekte.

Schnittstelle $\langle I20 \rangle$: <Database Access>

Operation	**Beschreibung**
bool create_user (dict user)	Speichert ein neues Benutzerobjekt mit Name, E-Mail und Passwort-Hash in der Datenbank.
dict | None get_user_by_email (str email)	Ruft einen Benutzer anhand der E-Mail-Adresse ab. Wird für Login und Registrierung verwendet.
bool insert_time_entry (dict entry)	Fügt einen neuen Zeiteintrag in die Datenbank ein.
list[dict] get_time_entries (int user_id, int project_id)	Gibt alle Zeiteinträge für ein bestimmtes Projekt und eine:n Nutzer:in zurück.
int create_project (dict project)	Speichert ein neues Projektobjekt und gibt die Projekt-ID zurück.
bool update_project (int project_id, dict changes)	Aktualisiert die Projektdaten basierend auf der ID.
bool delete_project (int project_id)	Löscht ein Projekt aus der Datenbank, inklusive zugehöriger Tasks und Zeiteinträge.
bool store_export_request (int user_id, int project_id, str format)	Protokolliert eine Exportanfrage zur späteren Verarbeitung.

Schnittstelle $\langle I30 \rangle$: <Team Service>

Operation	**Beschreibung**
list[dict] get_user_teams (int user_id)	Gibt alle Teams zurück, in denen ein Benutzer Mitglied ist, inklusive Teamname, Rolle und Erstellungsdatum.
dict create_new_team (str name, int user_id)	Erstellt ein neues Team mit dem angegebenen Namen und weist dem Ersteller automatisch die Rolle „Admin“ zu.
UserTeam | None check_admin (int user_id, int team_id)	Prüft, ob der Benutzer die Rolle Ädmin"des angegebenen Team hat.
UserTeam | None is_team_member (int user_id, int team_id)	Validiert, ob der angegebene Benutzer Mitglied des Teams ist. Gibt bei Erfolg die Beziehung zurück, sonst None.
list[dict] get_team_members (int team_id)	Gibt alle Mitglieder eines bestimmten Teams zurück, inklusive ihrer Benutzer-ID und Rolle.
bool delete_team_and_members (int team_id)	Löscht ein Team sowie alle zugehörigen Team-Mitgliedschaften aus der Datenbank.
list[dict] get_teams (int user_id)	Gibt alle Teams zurück, in denen ein Benutzer Mitglied ist, inklusive zugehöriger Projekte und der Benutzerrolle im Team.

Schnittstelle $\langle I40 \rangle$: Notification Service

Operation	**Beschreibung**
create_notification(user_id, message, type, project_id)	Erstellt eine Benachrichtigung für eine:n Nutzer:in zu einem bestimmten Projekt.
notify_task_assigned(user_id, task_name, project_name)	Erzeugt eine Benachrichtigung, wenn eine:r Nutzer:in eine neue Aufgabe zugewiesen wurde.
notify_task_reassigned(user_id, task_name, project_name)	Informiert den Benutzer über eine geänderte Aufgabenverteilung.
notify_user_added_to_team(user_id, team_name)	Benachrichtigt den:die Nutzer:in, dass er:sie einem neuen Team hinzugefügt wurde.
notify_progress_deviation(user_id, project_name, deviation_percentage)	Löst eine Benachrichtigung aus, wenn der Projektfortschritt deutlich vom Ziel abweicht.
notify_weekly_goal_achieved(user_id, project_name)	Informiert über das Erreichen des Wochenziels in einem Projekt.

Schnittstelle $\langle I50 \rangle$: <Teams>

Operation	**Beschreibung**
db.session.add(team)	Fügt eine neues Team in die Datenbank ein. Die Änderungen werden mit commit() gespeichert.
Team.query.filter_by(team_id=...) Team.query.get(team_id)	Findet Teams anhand der eindeutigen Team Id.
db.session.delete(team)	Entfernt ein Team dauerhaft aus der Datenbank.

Hinweis: Diese Schnittstelle wird durch SQLAlchemy bereitgestellt.

Schnittstelle $\langle I60 \rangle$: <Notifications>

Operation	**Beschreibung**
db.session.add(notification)	Fügt eine neue Benachrichtigung in die Datenbank ein. Die Änderungen werden mit commit() gespeichert.
Notification.query.filter_by(user_id=...)	Findet Benachrichtigungen für einen bestimmten Benutzer.
notification.is_read = True	Ändert den Status einer Benachrichtigung auf „gelesen“.
db.session.delete(notification)	Entfernt eine Benachrichtigung dauerhaft aus der Datenbank.

Hinweis: Diese Schnittstelle wird durch SQLAlchemy bereitgestellt.

3.3 Protokolle für die Benutzung der Komponenten

Im folgenden Abschnitt wird die korrekte Verwendung jeder Komponente durch ein Protokoll-Statechart sowie einen erläuternden Text beschrieben. Zusätzlich wird begründet, für welche Komponenten eine Wiederverwendung sinnvoll ist und für welche nicht.

3.3.1 Protokoll für $\langle C10 \rangle$ UI Layer

Das folgende Statechart beschreibt die zulässigen Zustände und Übergänge der Komponente **C10**. Es bildet typische Nutzungsabläufe ab – von der Anmeldung bis zur Navigation zwischen den Hauptansichten und dem Logout. Die Navigation zwischen den Ansichten erfolgt über die stets sichtbare Navbar, was durch bidirektionale Übergänge modelliert ist.

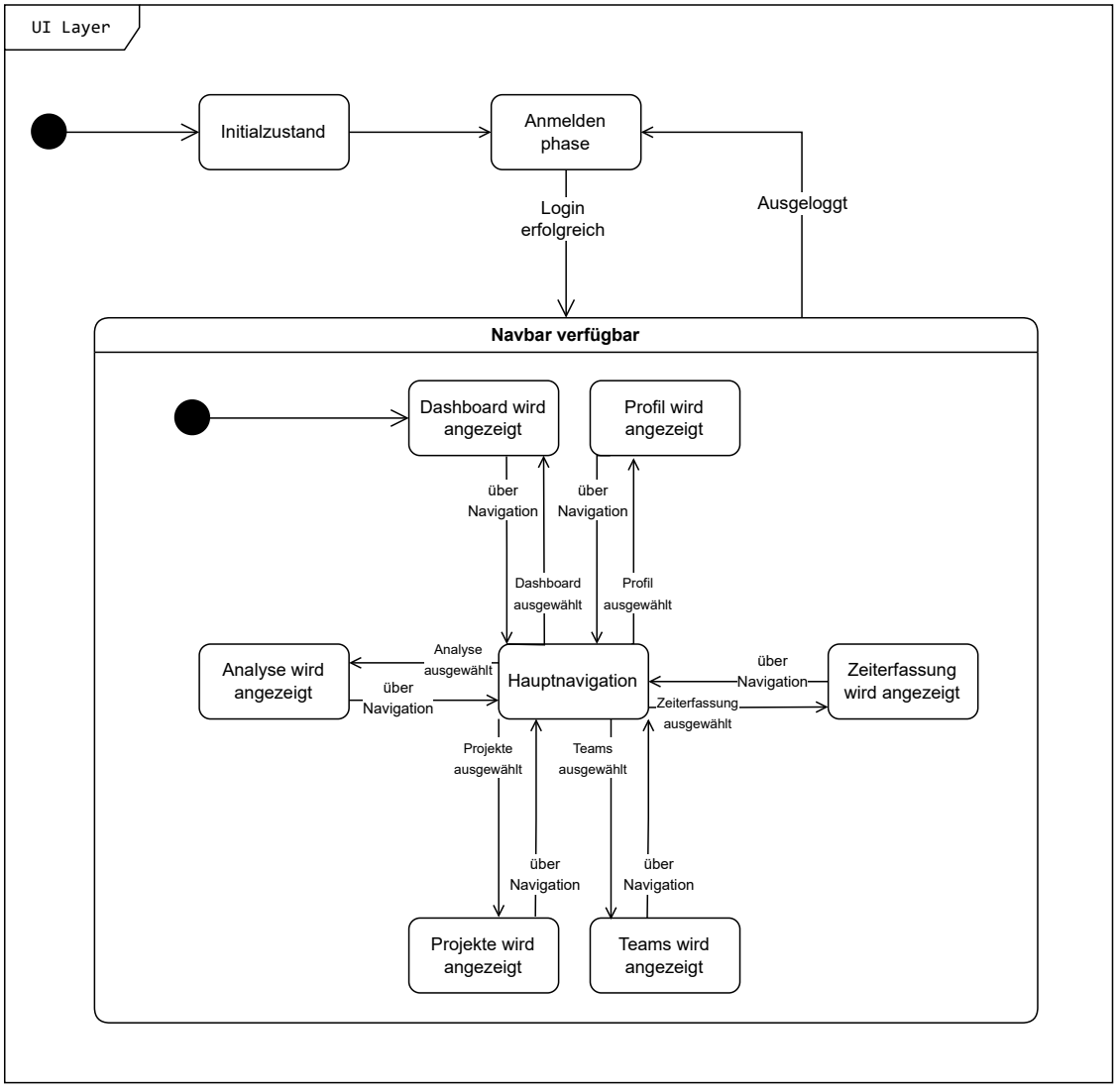

Abbildung 3.2: Protokoll-Statechart für Komponente $\langle C10 \rangle$: UI Layer

Zustände:

- **Anmeldenphase (Initialzustand)**: Startzustand der Anwendung; der:die Benutzer:in ist nicht authentifiziert.
- **Navbar verfügbar (komplexer Zustand)**: Oberzustand, in dem alle Hauptansichten über die stets sichtbare Navigationsleiste erreichbar sind.
- **Dashboard wird angezeigt**: Startübersicht nach erfolgreichem Login.
- **Hauptnavigation**: Zustand, in dem der:die Benutzer:in über die Navigationsleiste zwischen allen Ansichten wechselt.
- **Projekte wird angezeigt**: Darstellung und Bearbeitung von Projekten.
- **Zeiterfassung wird angezeigt**: Erfassung und Verwaltung von Zeiteinträgen.
- **Teams wird angezeigt**: Verwaltung von Teammitgliedern und Rollen.
- **Profil wird angezeigt**: Anzeige und Bearbeitung persönlicher Profildaten.
- **Analyse wird angezeigt**: Grafische Darstellung von Statistiken und Projektfortschritt.

Übergänge:

- `Login erfolgreich:` Wechsel in den Zustand **Navbar verfügbar**.
- `Dashboard ausgewählt:` Wechsel von Hauptnavigation zu **Dashboard wird angezeigt**.
- `Projekte ausgewählt:` Wechsel von Hauptnavigation zu **Projekte wird angezeigt**.
- `Zeiterfassung ausgewählt:` Wechsel von Hauptnavigation zu **Zeiterfassung wird angezeigt**.
- `Teams ausgewählt:` Wechsel von Hauptnavigation zu **Teams wird angezeigt**.
- `Profil ausgewählt:` Wechsel von Hauptnavigation zu **Profil wird angezeigt**.
- `Analyse ausgewählt:` Wechsel von Hauptnavigation zu **Analyse wir angezeigt**.
- `über Navigation:` Rückkehr von jeder Ansicht zur Hauptnavigation.
- `Ausloggen:` Wechsel zurück in die **Anmeldenphase**.

Bemerkung zur Wiederverwendung:
Die UI Layer ist modular aufgebaut und lässt sich leicht auf andere webbasierte Systeme übertragen. Ansichten wie Dashboard, Projektübersicht oder Analyse beruhen auf etablierten Designmustern, wodurch sich die Oberfläche flexibel in verschiedene Anwendungskontexte wie Aufgabenplaner, Produktivitätstools oder Projektplattformen integrieren lässt.

3.3.2 Protokoll für $\langle C20 \rangle$ Business Logic

Das folgende Statechart beschreibt die Ausführungslogik der Komponente **C20**. Es verdeutlicht die typischen Funktionsabläufe, welche durch Benutzeraktionen oder Systemereignisse ausgelöst werden.

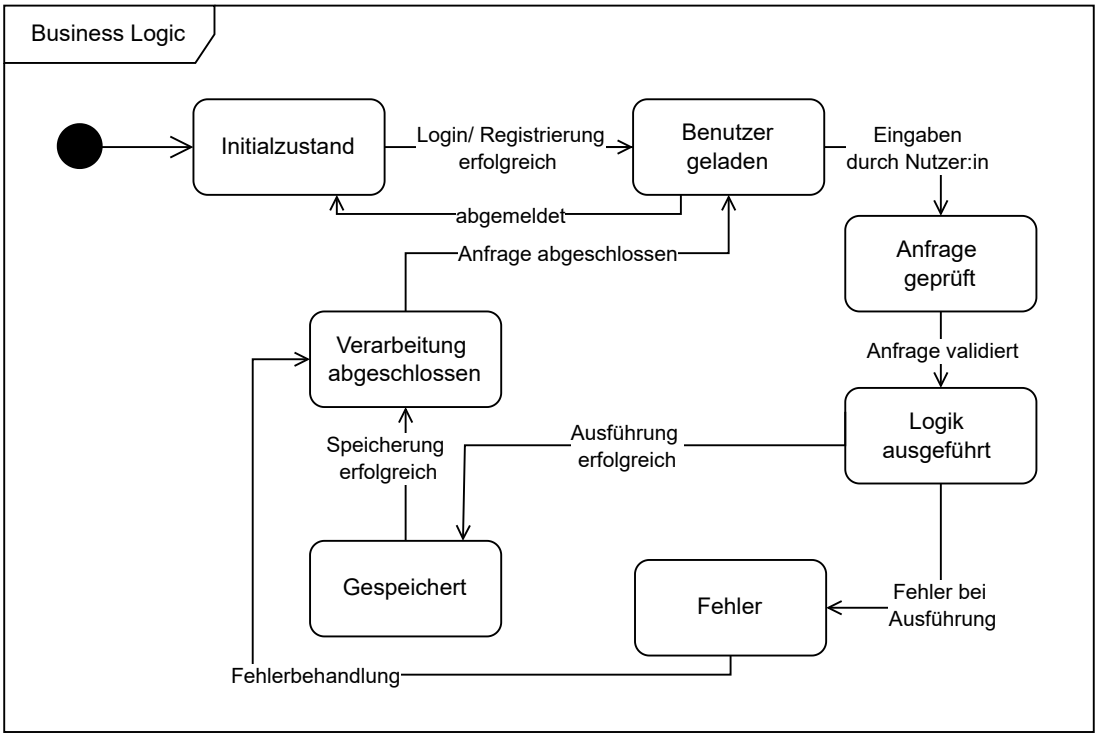

Abbildung 3.3: Protokoll-Statechart für Komponente $\langle C20 \rangle$: Business Logic

Zustände:

- **Initialzustand**: Keine aktive Verarbeitung.
- **Benutzer geladen**: Der Benutzer ist erfolgreich im System angemeldet und kann mit diesem arbeiten.
- **Anfrage geprüft**: Die Anfrage des Benutzers wurde auf Korrektheit und Sicherheit geprüft.
- **Logik ausgeführt**: Die Anfrage des Benutzer wurde in der Logik Struktur ausgeführt.
- **Fehler**: Es gab innerhalb der der Anfrage einen Logik Fehler, weshalb der Vorgang abgebrochen wurde.

- **Gespeichert**: Der Vorgang wurde erfolgreich zuende geführt und das Resulat der Anfrage wird gespeichert
- **Verarbeitung abgeschlossen**: Die Anfrage wurde vollends verarbeitet und endete in Speicherung des Resultats oder der Fehlerbehebung/meldung.

Übergänge:

- `Login/Registrierung erfolgreich:` Wechsel in Zustand **Benutzer geladen**.
- `Eingabe durch Nutzer:in:` Wechsel in Zustand **Anfrage geprüft**.
- `Anfrage validiert:` Wechsel in Zustand **Logik ausgeführt**.
- `Fehler bei Ausführung:` Wechsel in Zustand **Fehler**.
- `Ausführung erfolgreich:` Wechsel in Zustand **Gespeichert**.
- `Fehlerbehandlung:` Wechsel in Zustand **Verarbeitung abgeschlossen**.
- `Speicherung erfolgreich:` Wechsel in Zustand **Verarbeitung abgeschlossen**.
- `Anfrage abgeschlossen:` Wechsel in Zustand **Benutzer geladen**.
- `Abgemeldet:` Rückkehr in **Inintialzustand**.

Bemerkung zur Wiederverwendung:
Die Business Logic ist entkoppelt von der konkreten Oberfläche und arbeitet regelbasiert. Damit lässt sie sich einfach in Backend-Systeme mit vergleichbaren Funktionalitäten integrieren z.B. für andere Zeiterfassungssysteme, Projektmanagementlösungen oder rollenbasierte Portale.

3.3.3 Protokoll für $\langle C30 \rangle$ Database

Das folgende Statechart beschreibt die Zustände und Zugriffsabläufe innerhalb der Komponente **C30**. Es zeigt typische Datenbanktransaktionen im Kontext von ClockWise.

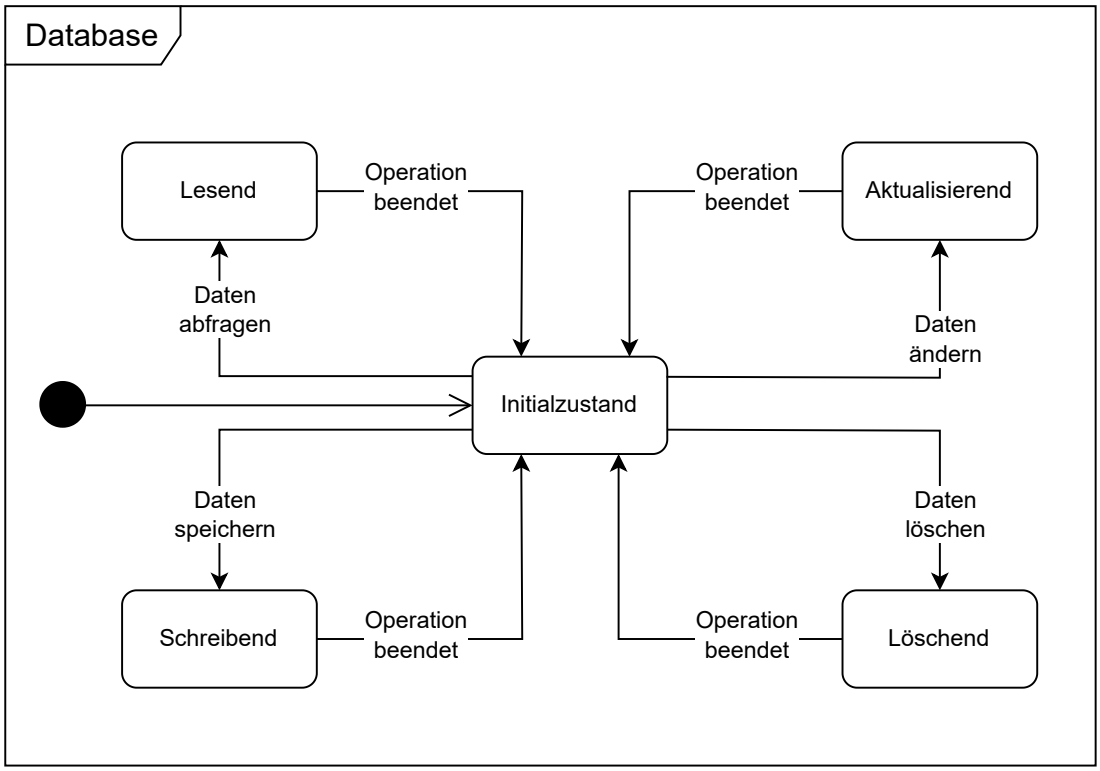

Abbildung 3.4: Protokoll-Statechart für Komponente $\langle C30 \rangle$: Database

Zustände:

- **Initialzustand**: Ausgangs- und Zwischenzustand, in dem keine aktive Operation ausgeführt wird.
- **Lesend**: Abfrage von Datenobjekten (z.B. Nutzer:innen, Projekte, Zeiteinträge).
- **Schreibend**: Erstellung neuer Objekte in der Datenbank.
- **Aktualisierend**: Modifikation bestehender Datensätze.
- **Löschend**: Entfernung von Objekten inklusive abhängiger Entitäten.

Übergänge:

- `Daten abfragen:` Wechsel in Zustand **Lesend**.

- `Daten speichern:` Wechsel in Zustand **Schreibend**.
- `Daten ändern:` Wechsel in Zustand **Aktualisierend**.
- `Daten löschen:` Wechsel in Zustand **Löschend**.
- `Operation beendet:` Rückkehr in **Initialzustand**.

Bemerkung zur Wiederverwendung:
Die Database basiert auf relationalen Grundprinzipien mit klaren Entitäten und Fremdschlüsselbeziehungen. Sie ist modular aufgebaut und damit leicht in vergleichbare Systeme integrierbar, bei denen Nutzer-, Projekt- oder Zeitdaten verwaltet werden müssen.

3.3.4 Protokoll für $\langle C40 \rangle$ Team and Role Management

Das folgende Statechart beschreibt die zulässigen Zustände und Übergänge eines Teams innerhalb der Komponente **C40**. Es bildet den Lebenszyklus eines Teams von der Erstellung, Bearbeitung bis Löschung systematisch ab.

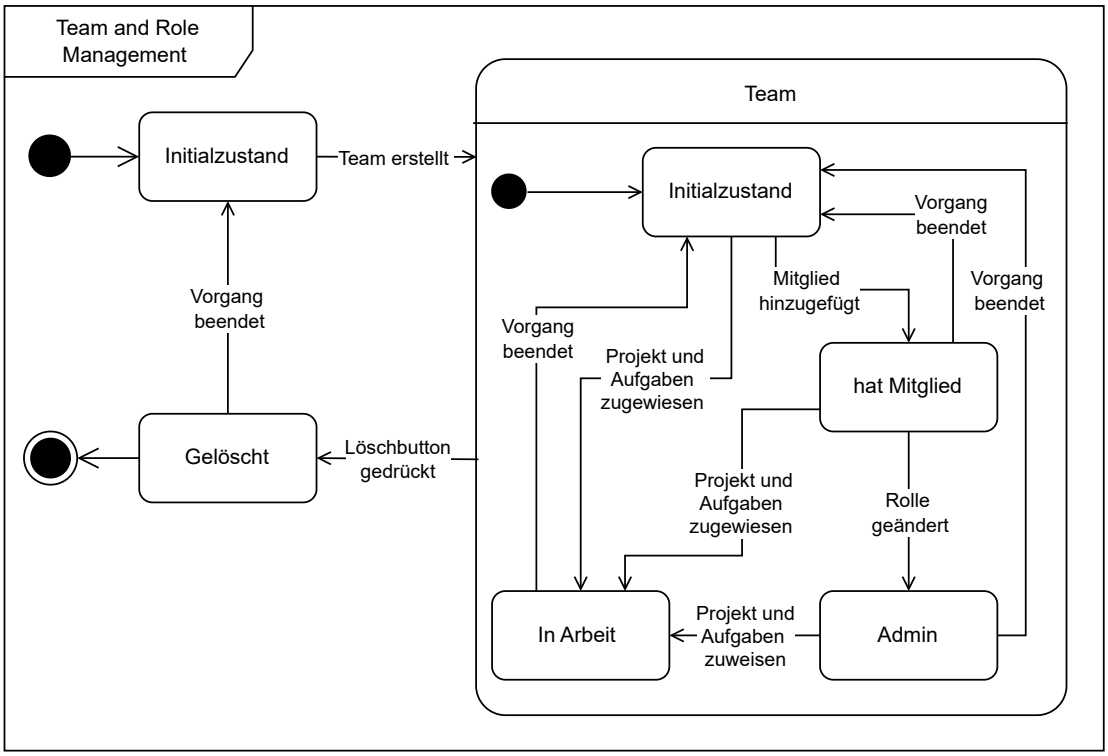

Abbildung 3.5: Protokoll-Statechart für Komponente $\langle C40 \rangle$: Team and Role Management

Zustände:

- **Initialzustand**: Die von der Benutzerobefläche gestellte Teamseite, ohne Teams
- **Team**: Es wurde von einem Nutzer ein Team erstellt, wodurch dieser zur Admin wurde. Der Nutzer ist das einzige Teammitglied.
  - **Initialszustand**: Die von der Benutzerobefläche gestellte Teamseite mit erstellten Teams.
  - **hat Mitglied**: Der Admin hat ein oder mehrere Mitglieder hinzugefügt.
  - **Admin**: Die Rolle von einem normalen Teammitglied wurde durch ein Admin geändert zu Rolle: Admin geändert. Dadurch hat der Nutzer, dessen Rolle geändert wurde andere Sichtbarkeiten und Aktionsrechte.

  - **In Arbeit**: Admins haben ein oder mehrere Teamprojekte erstellt, dem Team zugewiesen und Aufgaben erstellt und den einzelnen Mitgliedern zugeteilt, wodurch das Team zusammen an dem Projekt arbeiten können.

- **Gelöscht**: Das Team wurde von einem der Admins gelöscht, ist dadurch für alle nicht mehr sichtbar und wird mit seinen zugehörigen Projekten aus der Datenbank entfernt.

Übergänge:

- `Team erstellt:` Wechselt in Zustand: **Team**
- `Mitglied hinzugefügt:` Wechselt in Zustand: **hat Mitglied**
- `Rolle geändert:` Wechselt in Zustand: **Team**
- `Projekt und Aufgaben zugewiesen:` Wechselt in Zustand: **In Arbeit**
- `Projekt und Aufgaben zuweisen:` Wechselt in Zustand: **In Arbeit**
- `Löschbutton gedrückt:` Wechselt in Zustand: **Gelöscht** (möglich aus allen vorherigen Zuständen)
- `Vorgang beendet:` Wechselt in **Initialzustand**

Bemerkung zur Wiederverwendung:
Die Komponente lässt sich in jeder Webanwendung einsetzen die Benutzergruppen (Teams) und deren Berechtigungen (Rollen) verwalten muss. Kernfunktionen wie das Erstellen, Bearbeiten und Zuweisen von Rollen zu Benutzern oder Teams sind unabhängig von domänenspezifischer Logik umgesetzt. Die Rollen- und Rechtevergabe erfolgt über klar definierte Schnittstellen und ermöglicht eine einfache Integration in verschiedenste Einsatzszenarien – etwa in Content-Management-Systemen, Kollaborationsplattformen oder administrativen Backends.

3.3.5 Protokoll für $\langle C50 \rangle$ Notification

Das folgende Protokoll-Statechart beschreibt den Lebenszyklus einer einzelnen Benachrichtigung innerhalb der Komponente **C50**. Es dokumentiert die zulässigen Zustände und Übergänge im Verlauf der Nutzung durch das System oder den:die Nutzer:in. Die Zustände spiegeln die Interaktion des:der Nutzer:in mit dem Benachrichtigungssystem wider.

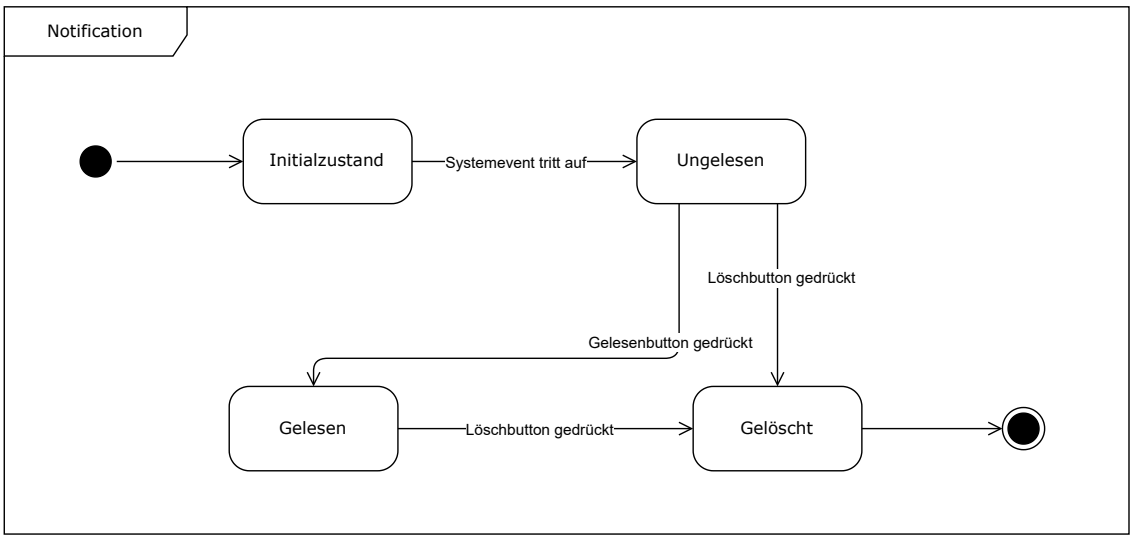

Abbildung 3.6: Protokoll-Statechart für Komponente $\langle C50 \rangle$: Notification

Zustände:

- **Ungelesen (Initialzustand)**: Die Benachrichtigung wurde vom System erzeugt, aber von dem:der Nutzer:in noch nicht gelesen.
- **Gelesen**: Der:die Nutzer:in hat die Benachrichtigung aktiv geöffnet oder bestätigt. Der Status `is_read` wird auf `true` gesetzt.
- **Gelöscht**: Die Benachrichtigung wurde aktiv von dem:der Nutzer:in entfernt. Sie ist nicht mehr sichtbar und wird aus der Datenbank entfernt.

Übergänge:

- `Systemevent tritt auf:` Erstellt eine neue Benachrichtigung -> Zustand: **Ungelesen**
- `Gelesenbutton gedrückt:` Wechselt in Zustand: **Gelesen**
- `Löschbutton gedrückt:` Wechselt in Zustand: **Gelöscht** (möglich aus beiden vorherigen Zuständen)

Bemerkung zur Wiederverwendung:
Die Komponente ist wiederverwendbar in jeder Webanwendung, die ereignisgesteuerte Benachrichtigungen benötigt. Die Zustandsübergänge sind klar definiert, vollständig abgeschlossen und unabhängig von der konkreten Geschäftslogik. Damit eignet sich das Benachrichtigungssystem als generische Infrastrukturkomponente für andere Projekte (z. B. Ticketsysteme, Kommunikationsplattformen, Projektmanagement-Tools).

4 Verteilungsentwurf

Bei der entwickelten Anwendung handelt es sich nicht um eine verteilte Anwendung im klassischen Sinne. Alle Softwarekomponenten sind zentral in einer einzigen Webanwendung integriert, die einen gemeinsamen Zugriffspunkt bietet.

Die Anwendung und all ihre Komponenten werden auf einem einzelnen Server oder Entwicklungsrechner betrieben. Sie ist logisch in Frontend und Backend unterteilt:

- **Frontend:** Bestehend aus der **UI Layer** <**C10**>.Die Benutzeroberfläche wird über einen Webbrowser bedient und fungiert als Frontend, bestehend aus HTML-, CSS- und JavaScript-Dateien, die vom Server statisch ausgeliefert und im Browser dynamisch ausgeführt werden. Die enthaltenen funktionalen Anteile werden clientseitig ausgeführt, ohne direkt mit dem Server zu kommunizieren.
- **Backend:** Bestehend aus der **Business Logic** <**C20**>, der **Database** <**C30**>, dem **Team and Role Management** <**C40**> und der **Notification** <**C50**>. Diese Komponenten bilden gemeinsam die zentrale Serverlogik.

Da alle Komponenten auf demselben System ausgeführt werden und keine Verteilung auf mehrere Maschinen erfolgt, wird auf eine explizite Darstellung mittels Verteilungsdiagramm verzichtet.

Sollte das System zukünftig auf mehrere Server verteilt oder containerisiert werden (z. B. durch den Einsatz von Docker oder einer externen Datenbank), kann ein entsprechendes Verteilungsdiagramm ergänzt werden.

5 Implementierungsentwurf

Dieses Kapitel beschreibt die technische Umsetzung der in Kapitel 3 definierten Systemkomponenten. Ziel ist es, die konkrete Implementierung auf Basis der zugrunde liegenden Softwarearchitektur nachvollziehbar darzustellen.

Jede Hauptkomponente wird in einem eigenen Abschnitt behandelt. Zur Veranschaulichung wird zunächst ein Klassendiagramm präsentiert, das den strukturellen Aufbau der jeweiligen Komponente zeigt. Anschließend erfolgt eine detaillierte Beschreibung der Aufgaben, der eingesetzten Technologien, der zentralen Operationen sowie der Kommunikation mit anderen Komponenten.

5.1 Implementierung von Komponente $\langle C10 \rangle$: UI Layer

Die Komponente **C10: UI Layer** bildet die zentrale Schnittstelle zur Interaktion mit der Anwendung. Sie ermöglicht es der:dem Nutzer:in, Projekte zu erstellen, Aufgaben zu verwalten, Zeit zu erfassen sowie Auswertungen über eine Analyseansicht einzusehen. Auch die Verwaltung von Gruppen erfolgt über eine eigene Teamansicht.

Obwohl die Anzeige von Benachrichtigungen funktional zur Benutzeroberfläche gehört, wurde deren technische Umsetzung im Rahmen der Komponente **C50** realisiert. Aus diesem Grund ist sie in der nachfolgenden Modulübersicht nicht enthalten.

Die `Dashboard`- und `Profile`-Seiten sind hingegen bewusst nicht modelliert, da sie keine eigene JavaScript-Logik enthalten und ausschließlich statische Inhalte oder einfache Anzeigeelemente darstellen.

Die Benutzeroberfläche ist als klassische Webanwendung mit **HTML5**, **CSS3** und **Vanilla JavaScript** umgesetzt. Für die Kommunikation mit der Geschäftslogik (C20) werden **REST-API-Endpunkte** asynchron über die **Fetch API** angesprochen.

Die grafische Oberfläche ist modular aufgebaut: Jede Hauptseite - beispielsweise Projektübersicht, Zeiterfassung, Analyseansicht oder Teamverwaltung - besteht aus einer HTML-Datei und einem zugehörigen JavaScript-Modul. Eine konsistente Navigation wird über die Datei `navbar.html` bereitgestellt.

Die einzelnen Views sind bewusst lose gekoppelt und voneinander getrennt modelliert, um die Modularität der Anwendung zu fördern. Aus diesem Grund enthält das Modulübersichtsdiagramm keine expliziten Assoziationen zwischen den View-Komponenten.

5.1.1 Modulübersicht

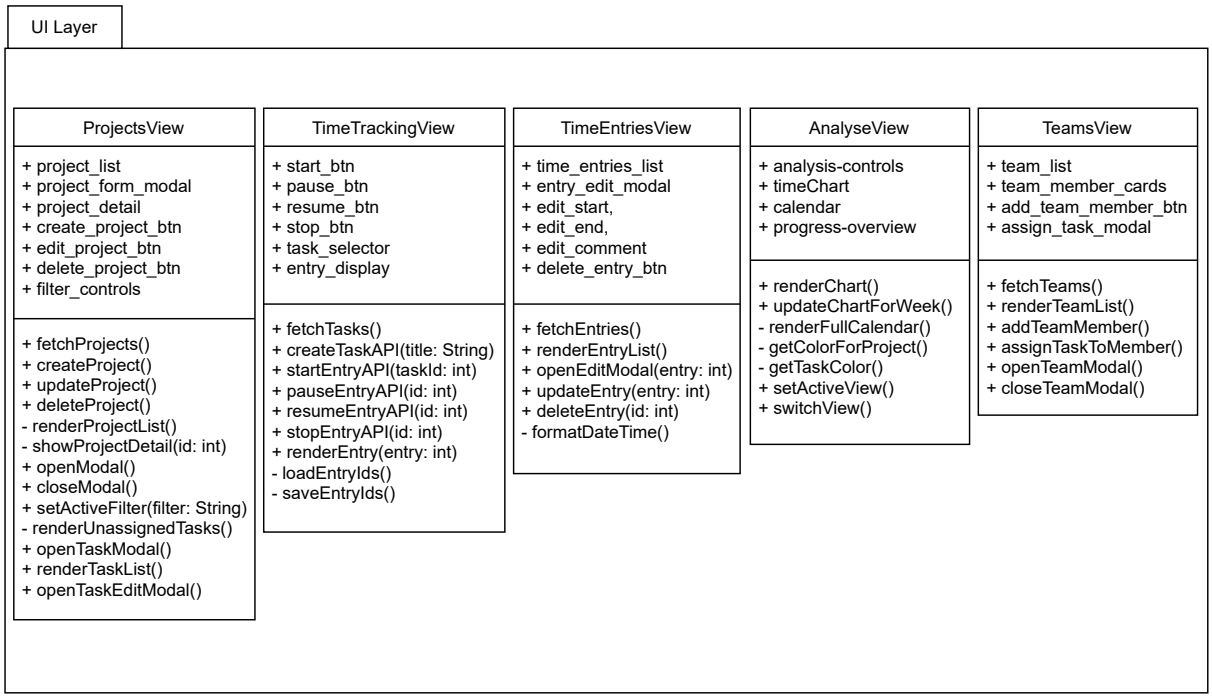

Abbildung 5.1: Modulübersicht der Komponente $\langle C10 \rangle$: UI Layer

5.1.2 Erläuterung

Projektübersicht$\langle CL11 \rangle$

Aufgabe

Ermöglicht das Erstellen, Bearbeiten, Löschen und Anzeigen von Projekten inklusive Detailansicht und zugehöriger Aufgaben. Die Seite bildet den zentralen Einstiegspunkt zur Projektverwaltung.

Elemente (HTML)

`#project_list` - Anzeige aller Projekte als Karten

`#project_form_modal` - Modal zur Projekterstellung und -bearbeitung

`#project_detail` - Detailansicht eines einzelnen Projekts mit zugehörigen Tasks

`#create_project_btn, #edit_project_btn, #delete_project_btn` - Interaktionsbuttons

`#filter_controls` - Buttongruppe zur Filterung nach Solo-/Teamprojekten und unzugeordneten Tasks

Operationen (JavaScript)

`fetchProjects()`, `createProject()`, `updateProject()`, `deleteProject()` - Kommunikation mit dem Backend

`renderProjectList()`, `showProjectDetail(id)` - Anzeige und Auswahl von Projekten

`openModal()`, `closeModal()` - Verwaltung des Projektformulars

`setActiveFilter(filter)`, `renderUnassignedTasks()` - Filterlogik

`openTaskModal()`, `renderTaskList()`, `openTaskEditModal()` - Aufgabenverwaltung innerhalb der Projektseite

Kommunikationspartner

Project Routes, Task Routes, DOM

Zeiterfassung$\langle CL12 \rangle$

Aufgabe

Ermöglicht die Live-Erfassung von Arbeits- oder Lernzeiten durch Start-, Stopp-, Pausier- und Fortsetzungsfunktionen. Der:Die Nutzer:in kann Aufgaben entweder aus bestehenden Tasks auswählen oder spontan neue Aufgaben anlegen. Die erfassten Zeiträume werden serverseitig gespeichert und in der Benutzeroberfläche angezeigt.

Operationen (JavaScript)

`fetchTasks()`, `createTaskAPI(title)` - Taskauswahl oder spontane Taskerstellung

`startEntryAPI(taskId)`, `pauseEntryAPI(id)`, `resumeEntryAPI(id)`, `stopEntryAPI(id)` - Kontrolle der Zeitmessung

`renderEntry(entry)` - Darstellung erfasster Einträge

`loadEntryIds()`, `saveEntryIds()`, `addEntryId()`, `removeEntryId()` - Verwaltung über LocalStorage

Kommunikationspartner

Time Entry Routes, Task Routes, LocalStorage, DOM

Zeiteinträge$\langle CL13 \rangle$

Aufgabe

Bietet eine Übersicht über erfasste Zeiteinträge inklusive Dauer, zugehörigem Task und Kommentaren. Der:Die Nutzer:in kann bestehende Einträge bearbeiten oder löschen.

Elemente (HTML)

`#time_entries_list` - Anzeige aller Zeiteinträge

`#entry_edit_modal` - Modal zur Bearbeitung von Einträgen

`#edit_start, #edit_end, #edit_comment` - Formulareingaben zur Bearbeitung

`#delete_entry_btn` - Button zum Entfernen eines Eintrags

Operationen (JavaScript)

`fetchEntries()`, `renderEntryList()` - Anzeige aller Einträge

`openEditModal(entry)`, `updateEntry(entry)`, `deleteEntry(id)` - Bearbeitung und Löschung

`formatDateTime()` - Umwandlung von Zeitformaten zur Darstellung

Kommunikationspartner

Time Entry Routes, DOM

Statistikansicht⟨$CL14$⟩

Aufgabe

Visualisiert die Zeiterfassung über mehrere Wochen hinweg und bietet eine Kalenderansicht. Der:Die Nutzer:in kann zwischen Wochen-, Kalender- und Fortschrittsansicht wechseln.

Elemente (HTML)

`#analysis-controls` - Steuerelemente zur Ansichtsumschaltung

`#timeChart` - Wochenansicht mit gestapeltem Balkendiagramm

`#calendar` - Kalenderansicht mit Zeit- und Fälligkeitsdaten

`#progress-overview` - Bereich für künftige Fortschrittsfunktionen

Operationen (JavaScript)

`renderChart()`, `updateChartForWeek()` - Visualisierung der Wochenzeit

`renderFullCalendar()` - Initialisierung und Darstellung des Kalenders

`getColorForProject()`, `getTaskColor()` - Farbzuweisung pro Projekt/Task

`setActiveView()`, `switchView()` - Umschalten zwischen Ansichten

Kommunikationspartner

Analysis Routes, DOM

Teamsverwaltung⟨$CL15$⟩

Aufgabe

Ermöglicht der:dem Nutzer:in die Verwaltung von Teamprojekten. Es können Teams erstellt, Mitglieder eingeladen oder entfernt sowie gemeinsame Projekte und Aufgaben eingesehen werden. Team-Administrator:innen können Aufgaben bestimmten Mitgliedern zuweisen.

Elemente (HTML)

`#team_list` - Übersicht aller Teams mit zugehörigen Projekten

`#create_team_form, #invite_user_form` - Formulare zur Erstellung von Teams und Einladung von Nutzer:innen

`#team_projects` - Anzeige aller Projekte innerhalb eines Teams

`#assign_task_form` - Formular zur Zuweisung von Tasks an Teammitglieder

Operationen (JavaScript)

`fetchTeams()`, `renderTeams()`, `createTeam()`, `inviteUserToTeam()` - Verwaltung und Anzeige von Teams

`removeMember()`, `showTeamProjects()`, `assignTaskToUser()` - Mitglieder- und Aufgabenverwaltung

Kommunikationspartner

Team Routes, Task Routes, User-Modul, DOM

5.2 Implementierung von Komponente $\langle C20 \rangle$: Business Layer

Die Komponente **C20: Business Layer** bildet das funktionale Kernstück der Anwendung. Sie verarbeitet Anfragen der:des Nutzer:in, steuert fachliche Abläufe und stellt die Verbindung zwischen der UI Layer (C10), Database (C30) und externen Systemen wie der Notification (C50) sicher.

Die Business Logic wurde mit dem **Flask-Framework** in **Python** implementiert. Für die Datenpersistenz kommt **Flask-SQLAlchemy** zum Einsatz. Authentifizierungsprozesse werden über **Flask-Login** sowie ergänzende Services wie Token- und Mail-Handling realisiert. Die REST-API-Endpunkte werden modular in Blueprints strukturiert und ermöglichen eine saubere Trennung der Anwendungsbereiche.

Für jede zentrale Domäne der Anwendung - darunter Projekte, Tasks, Zeiteinträge und Nutzer:innen - existieren ein Datenmodell, eine Service-Schicht mit der Business Logic sowie ein zugehöriges API-Modul zur Kommunikation mit der UI Layer.

5.2.1 Paket-/Modulübersicht

5.2.2 Erläuterung

Authentifizierung und Benutzerverwaltung$\langle CL21 \rangle$

Aufgabe

Ermöglicht der:dem Nutzer:in die Registrierung, Anmeldung, Bearbeitung des eigenen Profils sowie das Zurücksetzen des Passworts. Zusätzlich wird das Hochladen eines Profilbildes unterstützt.

Service-Funktionen

`register_user()`, `login_user()`, `edit_user()`, `delete_user()`, `password_forget()`, `upload_profile_picture()`, `generate_reset_token()`

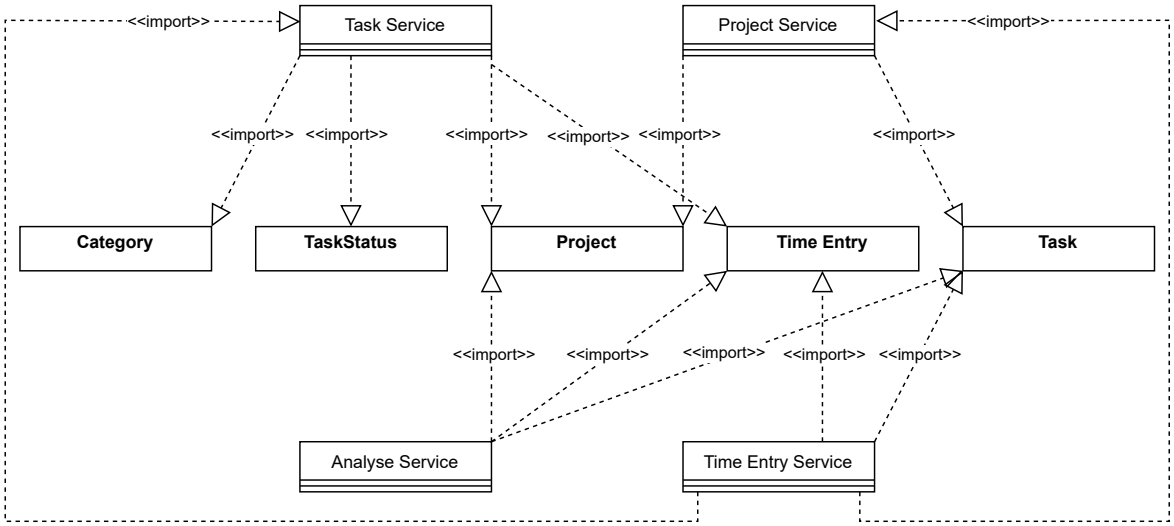

Abbildung 5.2: Modulübersicht der Komponente `C20` (Teil 1): Business Logic

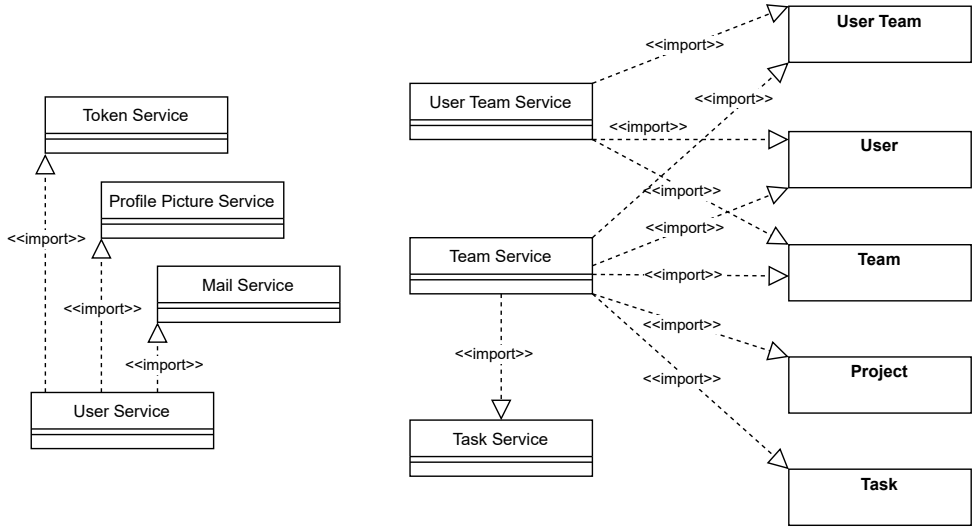

Abbildung 5.3: Modulübersicht der Komponente `C20` (Teil 2): Business Logic

Kommunikationspartner

UI Layer (C10), Mail-Service, Token-Service, Profile-Picture-Service, Database (C30)

Projektverwaltung$\langle CL22 \rangle$

Aufgabe

Ermöglicht das Anlegen, Bearbeiten, Löschen und Abrufen von Projekten. Projekte sind Träger für Tasks, Zeiteinträge und Zeitziele. Zeitlimits lassen sich manuell oder aus ECTS-Punkten berechnen.

Service-Funktionen

`create_project()`, `get_project()`, `update_project()`, `delete_project()`, `update_total_duration_for_project()`, `calculate_time_limit_from_credits()`

Kommunikationspartner

Task-Modul, TimeEntry-Modul, UI Layer (C10), Database (C30)

Taskverwaltung⟨$CL23$⟩

Aufgabe

Ermöglicht die Erstellung, Bearbeitung, Zuweisung und Löschung von Aufgaben. Tasks können optional Kategorien oder Projekten zugeordnet werden. Auch spontane Aufgaben ohne Projektbindung sind möglich.

Service-Funktionen

`create_task()`, `get_task_by_id()`, `get_tasks_by_project()`, `update_task()`, `delete_task()`, `get_tasks_without_time_entries()`, `get_task_with_time_entries()`, `get_unassigned_tasks()`, `get_tasks_assigned_to_user()`, `unassign_tasks_for_user_in_team()`, `update_total_duration_for_task()`

Kommunikationspartner

Project-Modul, TimeEntry-Modul, Category-Modul, UI Layer (C10), Database (C30)

Zeiterfassungsmodul⟨$CL24$⟩

Aufgabe

Ermöglicht Live-Zeiterfassung per Start-, Pause-, Fortsetzungs- und Stoppfunktionen. Neue Tasks können direkt aus der Zeiterfassung erzeugt werden. Die Daten werden fortlaufend gespeichert.

Service-Funktionen

`start_time_entry()`, `pause_time_entry()`, `resume_time_entry()`, `stop_time_entry()`, `create_time_entry()`, `update_durations_for_task_and_project()`, `get_latest_time_entries_for_user()`

Kommunikationspartner

Task-Modul, Project-Modul, UI Layer (C10), Database (C30)

Zeiteinträge verwalten⟨$CL25$⟩

Aufgabe

Bietet Funktionen zum Abrufen, Bearbeiten und Löschen bestehender Zeiteinträge. Jeder Eintrag enthält Startzeit, Endzeit, Kommentar und Taskbezug.

Service-Funktionen

`get_time_entry_by_id()`, `get_time_entries_by_task()`, `update_time_entry()`, `delete_time_entry()`

Kommunikationspartner

TimeEntry-Modul, Task-Modul, UI Layer (C10), Database (C30)

Statistik- und Kalenderauswertung⟨$CL26$⟩

Aufgabe

Aggregiert und filtert Zeiteinträge für Kalender- und Statistikvisualisierungen. Vergleicht Ist-Zeiten mit Planwerten und ermöglicht Wochenübersichten, Kalenderansichten sowie Projektfortschrittsanalysen.

Service-Funktionen

`load_time_entries()`, `load_projects()`, `load_tasks()`, `filter_time_entries_by_date()`, `aggregate_weekly_time()`, `calendar_events()`, `calendar_worked_time()`, `progress_per_project()`, `actual_target_comparison()`, `load_target_times()`, `export_time_entries_pdf()`, `export_time_entries_csv()`, `calendar_due_dates()`, `tasks_in_month()`, `aggregate_time_by_day_project_task()`, `worked_time_today()`

Kommunikationspartner

TimeEntry-Modul, Project-Modul, UI Layer (C10), Database (C30)

Team- und Rollenverwaltung⟨$CL27$⟩

Aufgabe

Ermöglicht das Erstellen von Teams, das Hinzufügen und Entfernen von Mitgliedern sowie die Rollenzuweisung (z.B. „admin“). Die Mitgliedschaften werden über eine separate Zuordnungstabelle gepflegt.

Service-Funktionen

`create_new_team()`, `add_member()`, `delete_member()`, `remove_member_from_team()`, `get_user_teams()`, `get_team_members()`, `get_teams()`, `delete_team_and_members()`, `check_admin()`, `is_team_member()`

Kommunikationspartner

User-Modul, Notification-Modul (C50), Task-Modul, UI Layer (C10), Database (C30)

5.3 Implementierung von Komponente $\langle C30 \rangle$: Database

Die Komponente **C30: Database** ist für die dauerhafte Speicherung und Verwaltung aller anwendungsrelevanten Entitäten zuständig. Sie bildet das Rückgrat der ClockWise Anwendung, indem sie Nutzer:innen, Teams, Projekte, Aufgaben, Zeiteinträge, Benachrichtigungen und Kategorien strukturiert abbildet.

Die Persistenz erfolgt über das ORM Framework **Flask SQLAlchemy**. Alle Datenbankmodelle sind als Python Klassen implementiert und spiegeln die fachlichen Beziehungen durch geeignete Primär und Fremdschlüssel sowie deklarative Beziehungen wider. Zusätzlich werden Enumerationen zur Repräsentation fester Wertebereiche eingesetzt, z. B. für Projekt oder Aufgabenstatus.

5.3.1 Paket-/Klassendiagramm

Die Enumerationen `ProjectType` und `TaskStatus` werden im Diagramm mit `«uses»` markiert, da es sich um fest definierte Wertebereiche handelt, die innerhalb der jeweiligen Klassen referenziert, aber nicht als eigenständige Komponenten importiert oder instanziiert werden.

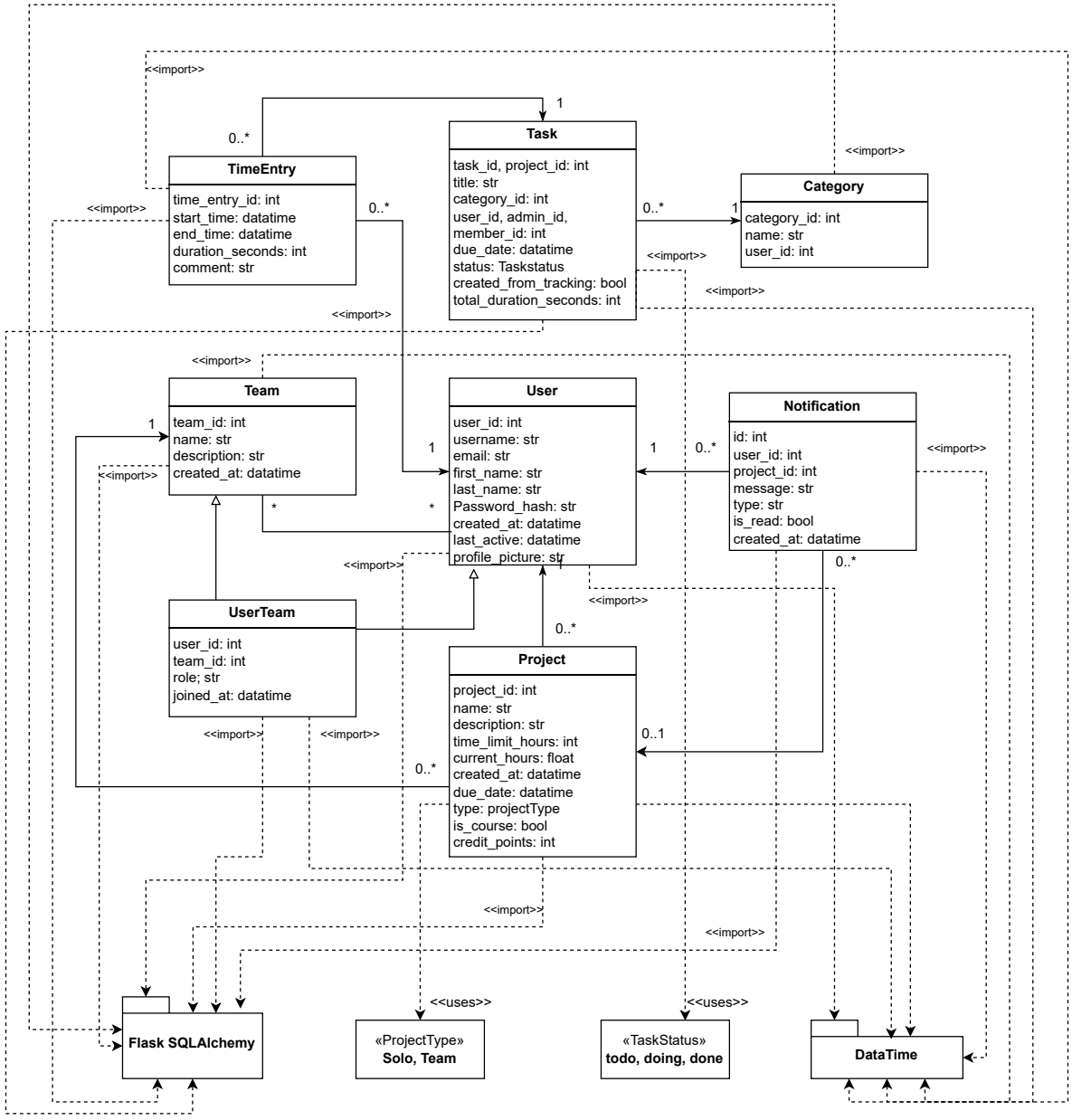

Abbildung 5.4: Klassendiagramm der Komponente ⟨$C30$⟩: Database

5.3.2 Erläuterung

User⟨$CL30$⟩

Aufgabe

Repräsentiert einen registrierten Nutzer bzw. eine registrierte Nutzerin des Systems. Dient als zentrale Bezugsklasse für Projekte, Aufgaben, Teams, Zeiteinträge, Benachrichtigungen und Kategorien. Die Authentifizierung erfolgt über Flask-Login.

Attribute

`user_id`: int – Primärschlüssel

`username`, `email`: String – Eindeutige Login-Daten

`first_name`, `last_name`: String – Personenbezogene Daten

`password_hash`: String – Gesicherter Passwort-Hash

`profile_picture`: String – Optionales Profilbild

`created_at`, `last_active`: DateTime Zeitstempel

Operationen

Authentifizierungsmethoden (z. B. `get_id()` für Flask Login), sowie Serialisierungsmethoden und Debug Ausgabe

Kommunikationspartner

Project, Task (`assigned_task`, `admin_tasks`, `member_tasks`), UserTeam (für Teamzugehörigkeit), TimeEntry, Notification, Category

Team⟨$CL31$⟩

Aufgabe

Stellt eine organisatorische Einheit dar, in der mehrere Nutzer:innen zusammenarbeiten können.

Attribute

`team_id`: int – Primärschlüssel

`name`, `description`: String – Teaminformationen

`created_at`: DateTime – Erstellungszeitpunkt

Operationen

Validierungsmethoden, `is_valid()`

Kommunikationspartner

UserTeam, Project

UserTeam⟨*CL*32⟩

Aufgabe

Implementiert eine Beziehung zwischen Nutzer:in und Team mit Zusatzinformationen wie Rolle und Beitrittsdatum.

Attribute

`user_id`, `team_id`: int – Primärschlüssel

`role`: String – Rolle im Team (Member oder admin)

`joined_at`: DateTime – Zeitpunkt des Teambeitritts

Operationen

`__repr__()` zur Debug Ausgabe

Kommunikationspartner

User, Team

Project⟨*CL*33⟩

Aufgabe

Verwaltet ein Projekt oder Lernvorhaben, inklusive Zeitzielen, Fortschritt und Verknüpfung mit Nutzer:in oder Team.

Attribute

`project_id`, `name`, `description`

`time_limit_hours`, `current_hours`: Integer/Float

`created_at`, `due_date`: DateTime

`type`: Enum – Projektart (Solo/Team)

`is_course`, `credit_points`: Kursbezogene Felder

Operationen

`duration_readable()` zur menschenlesbaren Ausgabe der investierten Zeit

Kommunikationspartner

User, Team, Task, Notification

Task⟨$CL34$⟩

Aufgabe

Repräsentiert eine konkrete Tätigkeit innerhalb eines Projekts.

Attribute

`task_id`: int – Primärschlüssel

`project_id`: int – Zugehöriges Projekt

`user_id`, `admin_id`, `member_id`: int – Nutzer-Zuweisung je nach Projekttyp

`category_id`: int – Thematische Kategorie

`title`, `description`: String – Inhaltsangaben

`due_date`: DateTime – Frist

`status`: Enum – Aufgabenstatus (todo, in_progress, done)

`created_at`: DateTime – Erstellungszeitpunkt

`created_from_tracking`: bool – via Zeiterfassung erzeugt

`total_duration_seconds`: int – akkumulierte Zeitdauer

Kommunikationspartner

Project, User, TimeEntry, Category

TimeEntry⟨$CL35$⟩

Aufgabe

Zeichnet Arbeitszeit für Aufgaben auf und verknüpft sie mit Nutzer:in und Aufgabe.

Attribute

`time_entry_id`: int – Primärschlüssel

`start_time`, `end_time`: DateTime

`duration_seconds`: int – Zeitspanne in Sekunden

`comment`: String – Optionaler Kommentar

Operationen

`to_dict()`, `__repr__()`

Kommunikationspartner

Task, User

Notification⟨*CL*36⟩

Aufgabe

Benachrichtigt Nutzer:innen über projekt- oder aufgabenbezogene Ereignisse.

Attribute

`id`: int – Primärschlüssel

`user_id`, `project_id`: int – Zielnutzer und Projektbezug

`message`, `type`: String – Inhalt und Typ

`is_read`: Boolean – Gelesen-Status

`created_at`: DateTime – Zeitstempel

Kommunikationspartner

User, Project

Category⟨*CL*37⟩

Aufgabe

Dient zur thematischen Klassifikation von Aufgaben.

Attribute

`category_id`: int – Primärschlüssel

`name`: String – Bezeichnung

`user_id`: int – Besitzer der Kategorie

Kommunikationspartner

Task

5.4 Implementierung von Komponente $\langle C40 \rangle$: Team and Role Management

Die Komponente **C40: Team and Role Management** ist für die strukturierte Organisation von Nutzer:innen in Teams sowie die Zuweisung von Rollen innerhalb dieser Teams verantwortlich. Sie stellt sicher, dass Aktionen und Sichtbarkeiten systematisch nach Rollenrechten differenziert werden.

Die Komponente wurde mit dem **Flask Framework** implementiert. Sie verwendet sowohl klassische Webrouten mit HTML Templates als auch REST-konforme API-Endpunkte zur Interaktion mit dem Frontend. Die klassischen Routen dienen primär der Anzeige von Teamübersichten, Detailseiten und Verwaltungsansichten innerhalb der Anwendung und nutzen `Jinja2` Templates zur serverseitigen Darstellung. Die REST-API Endpunkte werden hingegen für dynamische Interaktionen wie das Erstellen, Bearbeiten und Löschen von Teams sowie das Zuweisen und Verwalten von Mitgliedern und Rollen verwendet, insbesondere zur nahtlosen Integration mit dem JavaScript basierten Frontend-Dashboard. Die serverseitige Logik ist in Python Services gekapselt und trennt klar zwischen Präsentations- und Verarbeitungsschicht.

Für Authentifizierung und Autorisierung wird **Flask Login** verwendet, wobei insbesondere auf rollenbasierten Zugriff innerhalb der Teamstruktur geachtet wird. Die Persistenz erfolgt über **Flask SQLAlchemy**, wobei die Zwischentabelle `UserTeam` eine zentrale Rolle bei der Verwaltung der Rollenbeziehungen zwischen Nutzer:innen und Teams spielt.

Zur besseren Veranschaulichung der Komponente werden im Folgenden zwei UML Diagramme präsentiert:

5.4.1 Paket-/Klassendiagramm: Services und Routen

- Das erste Diagramm stellt die **Service- und Routenschicht** dar und zeigt, welche Funktionalitäten für das Erstellen, Verwalten und Interagieren mit Teams zur Verfügung stehen.

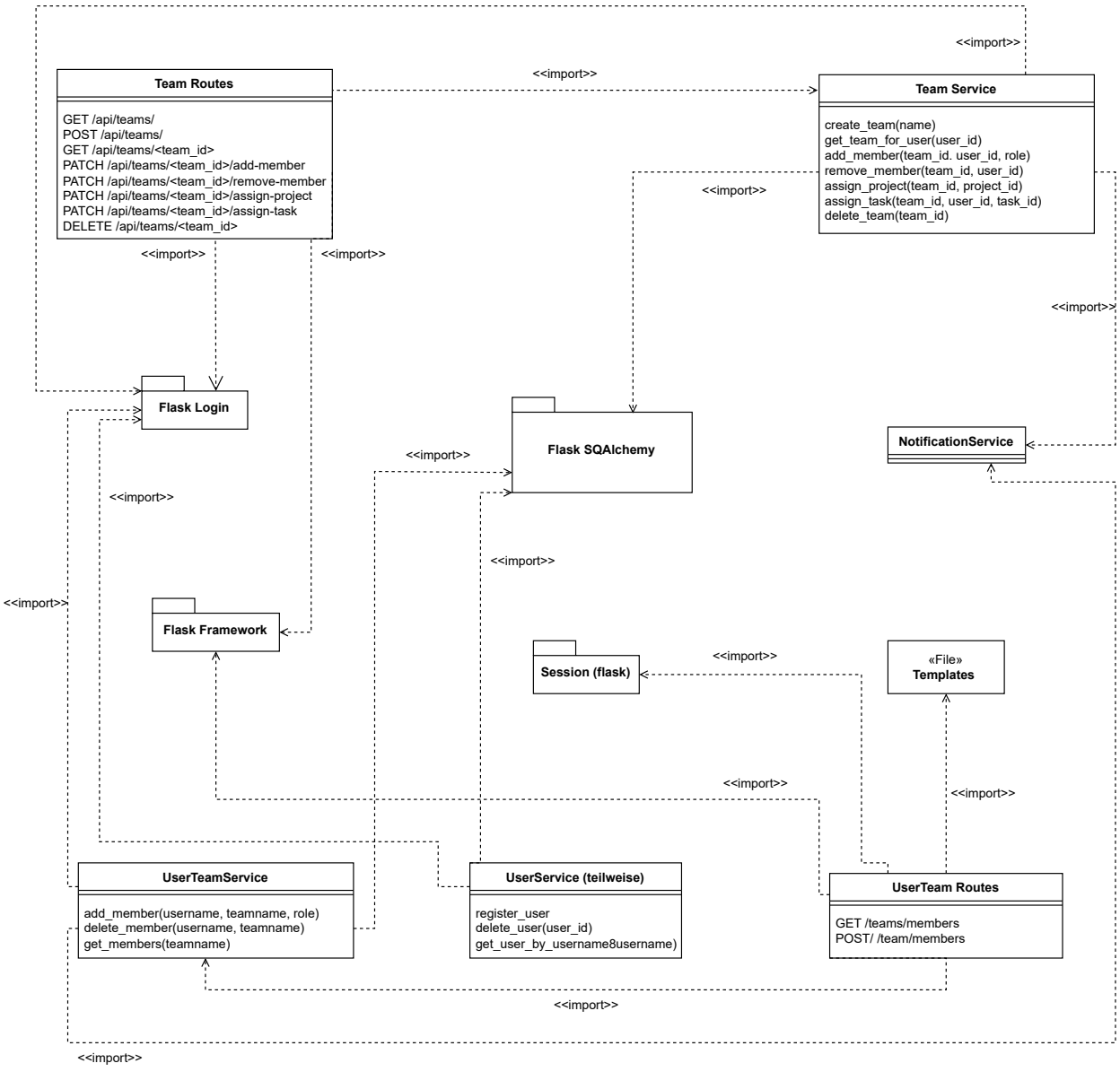

Abbildung 5.5: Klassendiagramm der Service- und Routenschicht der Komponente $\langle C40 \rangle$: Team and Role Management

5.4.2 Paket-/Klassendiagramm: Datenmodelle und Abhängigkeiten

- Das zweite Diagramm fokussiert sich auf die **Struktur der zugrunde liegenden Datenmodelle** sowie deren Beziehungen untereinander und zu den entsprechenden Services.

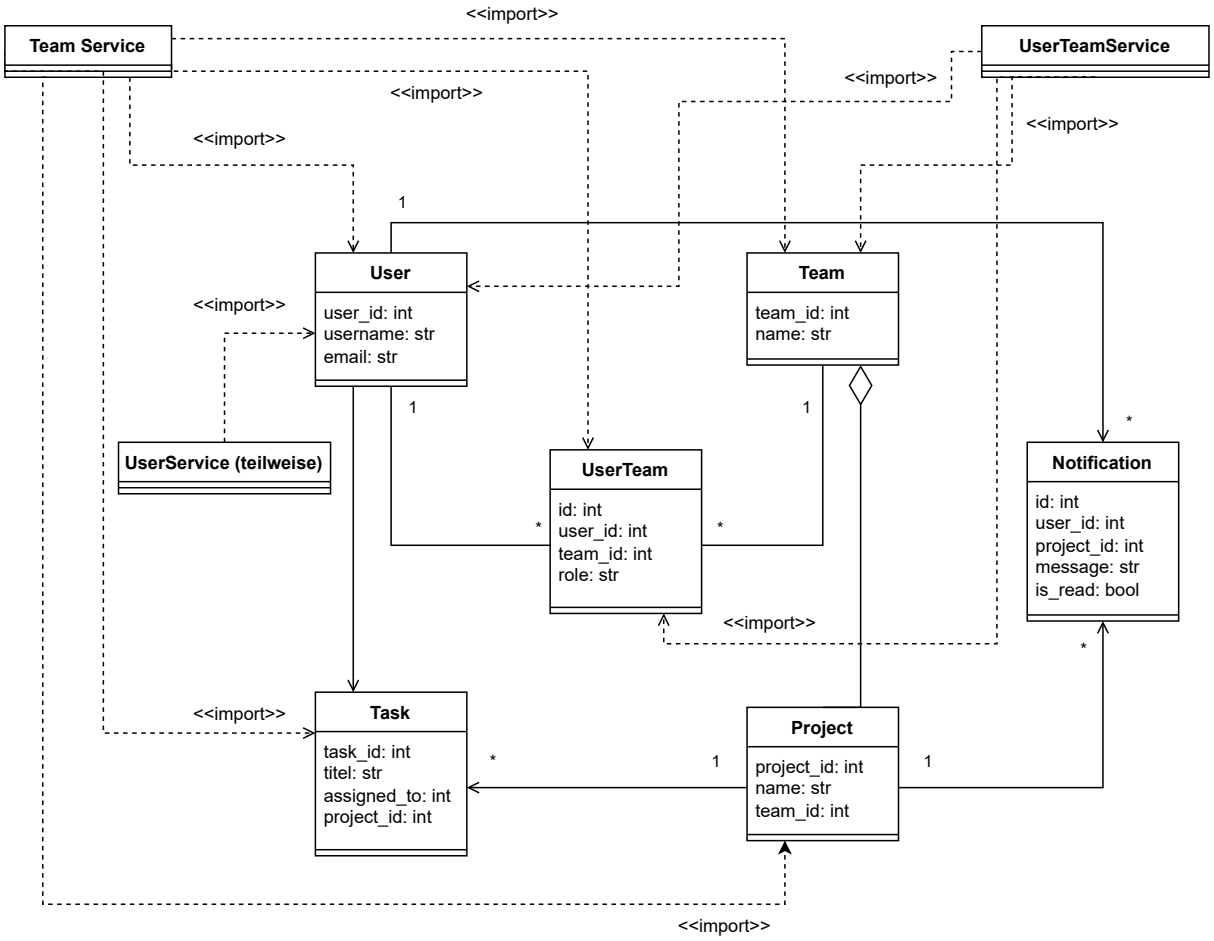

Abbildung 5.6: Klassendiagramm der Datenmodelle und ihrer Nutzung durch Services der Komponente $\langle C40 \rangle$: Team and Role Management

5.4.3 Erläuterung

Team Routes⟨$CL41$⟩

Aufgabe

Stellt REST-API Endpunkte für die Verwaltung von Teams, Teammitgliedern, Rollen und der Projektzuweisung bereit.

Operationen

`GET /api/teams/`: Gibt alle Teams zurück, zu denen der/die Nutzer:in gehört
`POST /api/teams/`: Erstellt ein neues Team
`GET /api/teams/<team_id>`: Gibt Informationen über ein bestimmtes Team zurück
`PATCH /api/teams/<team_id>/add-member`: Fügt ein:e Nutzer:in dem Team hinzu
`PATCH /api/teams/<team_id>/remove-member`: Entfernt ein:e Nutzer:in aus dem Team
`PATCH /api/teams/<team_id>/assign-project`: Weist dem Team ein Projekt zu
`PATCH /api/teams/<team_id>/assign-task`: Weist einem Mitglied eine Aufgabe zu
`DELETE /api/teams/<team_id>`: Löscht ein Team

Kommunikationspartner

Frontend, `TeamService`, Modelle: `Team`, `UserTeam`, `Project`, `Task`, `Notification`

TeamService⟨$CL42$⟩

Aufgabe

Stellt zentrale Backendlogik für das Erstellen, Verwalten und Löschen von Teams sowie für Mitglieder- und Aufgabenverwaltung bereit.

Operationen

`create_team(name)`, `get_teams_for_user(user_id)`,
`add_member(team_id, user_id, role)`, `remove_member(team_id, user_id)`,
`assign_project(team_id, project_id)`, `assign_task(team_id, user_id, task_id)`,
`delete_team(team_id)`

Kommunikationspartner

Modelle: `Team`, `UserTeam`, `User`, `Project`, `Task`, `NotificationService`

UserTeamService⟨$CL43$⟩

Aufgabe

Ermöglicht die Teamverwaltung über Benutzernamen und Teamnamen. Wird vorrangig für klassische Formularinteraktionen verwendet.

Operationen

`add_member(username, teamname, role)`: Fügt ein:e Nutzer:in anhand des Namens einem Team hinzu
`delete_member(username, teamname)`: Entfernt ein:e Nutzer:in anhand des Namens aus

einem Team

`get_members(teamname)`: Gibt alle Mitglieder eines bestimmten Teams zurück

Kommunikationspartner

Modelle: `User`, `Team`, `UserTeam`, Service: `NotificationService`

UserTeam Routes⟨$CL44$⟩

Aufgabe

Bietet eine klassische Webschnittstelle (Formularansicht) zur Anzeige und Verwaltung von Teammitgliedern.

Operationen

`GET /team/members`: Zeigt alle Mitglieder des ausgewählten Teams

`POST /team/members`: Verarbeitet Formular zum Hinzufügen oder Entfernen von Mitgliedern

Kommunikationspartner

Templates: `teams.html`, Service: `user_team_service`, Session (Flask)

UserService (Teilweise)⟨$CL46$⟩

Aufgabe

Verwaltet grundlegende Nutzeroperationen, die auch Auswirkungen auf Teamstrukturen haben können.

Operationen

`register_user(...)`, `delete_user(user_id)`, `get_user_by_username(username)`

Kommunikationspartner

Modelle: `User`, indirekt `UserTeam`

5.5 Implementierung von Komponente ⟨$C50$⟩: Notification

Die Komponente **C50: Notification** ist dafür zuständig, den:die Nutzer:in über relevante Ereignisse innerhalb der Anwendung zu informieren. Dies erfolgt serverseitig über REST-Endpunkte zur Abfrage und Aktualisierung von Benachrichtigungen.

Die Komponente wurde mit dem **Flask-Framework** implementiert. Für Authentifizierung und Autorisierung wird **Flask-JWT-Extended** genutzt. Die Persistenz erfolgt über **Flask-SQLAlchemy**.

5.5.1 Paket-/Klassendiagramm

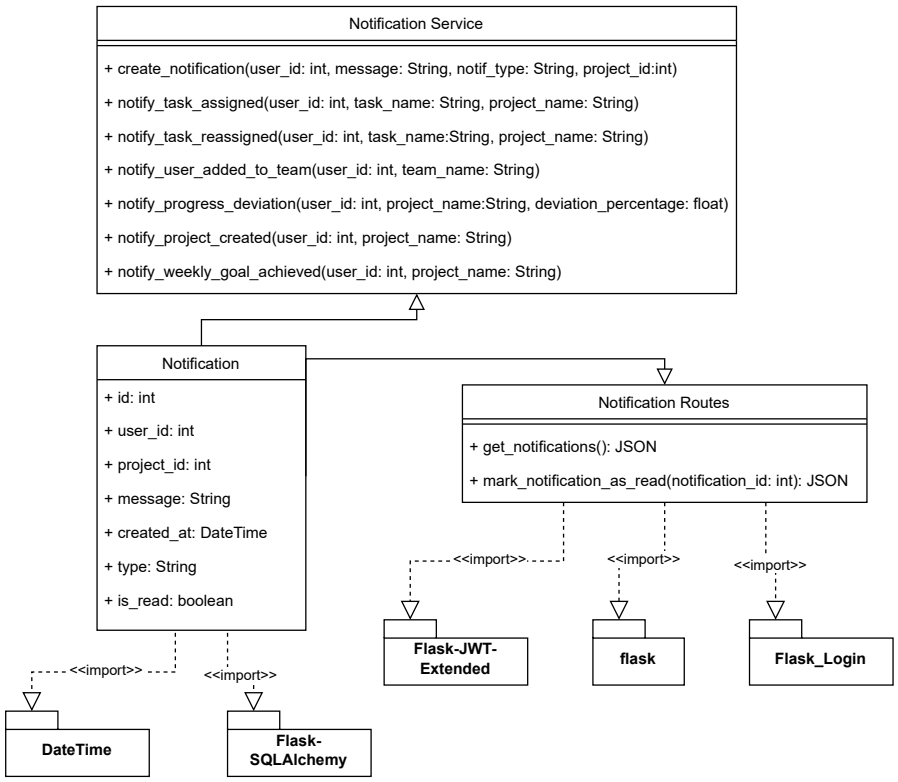

Abbildung 5.7: Klassendiagramm der Komponente $\langle C50 \rangle$: Notification

5.5.2 Erläuterung

Notification⟨$CL51$⟩

Aufgabe

Repräsentiert eine einzelne Systembenachrichtigung, die einem:einer bestimmten Nutzer:in und Projekt zugeordnet ist.

Attribute

`id`: int – Primärschlüssel der Benachrichtigung
`user_id`: int – Referenz auf den:die betroffene:n Nutzer:in
`project_id`: int – Referenz auf das zugehörige Projekt
`message`: String – Inhalt der Benachrichtigung
`created_at`: DateTime – Erstellungszeitpunkt
`type`: String – Typ der Benachrichtigung (z. B. „Info“, „Team“)
`is_read`: boolean – Gelesen-Status

Operationen

Automatisch verwaltete Methoden durch SQLAlchemy (CRUD)

Kommunikationspartner

Notification Service, Notification Routes, Datenbank

Notification Service⟨$CL52$⟩

Aufgabe

Zentrale Komponente zur Erzeugung und Verwaltung von Benachrichtigungen aus systeminternen Ereignissen.

Operationen

```
create_notification(user_id, message, notif_type, project_id)
notify_task_assigned(user_id, task_name, project_name)
notify_task_reassigned(user_id, task_name, project_name)
notify_user_added_to_team(user_id, team_name)
notify_progress_deviation(user_id, project_name, deviation_percentage)
notify_project_created(user_id, project_name)
notify_weekly_goal_achieved(user_id, project_name)
```

Kommunikationspartner

Notification (Modell), Projektverwaltung, Task-Management, Teamverwaltung

Notification Routes⟨$CL53$⟩

Aufgabe

Bietet REST-API-Endpunkte für den Zugriff auf und die Interaktion mit Benachrichtigungen.

Operationen

`get_notifications()`: Gibt alle (oder nur ungelesene) Benachrichtigungen eines:einer Nutzer:in zurück

`mark_notification_as_read(notification_id)`: Markiert eine Benachrichtigung als gelesen

Kommunikationspartner

Frontend (Benutzeroberfläche), Notification Service, Authentifizierungssystem (Flask-JWT)

6 Datenmodell

In diesem Kapitel wird der Aufbau und die Funktionsweise des Datenmodels genauer beschrieben. Die Speicherung der Daten erfolgt dabei in einer SQLite-Datenbank. Ziel ist es eine möglichst stabile und fehlerfrei Datenbank den Nutzer:innen zu bieten, um eine reibungslose Nutzung der Anwendung zu gewährleisten.

6.1 Diagramm

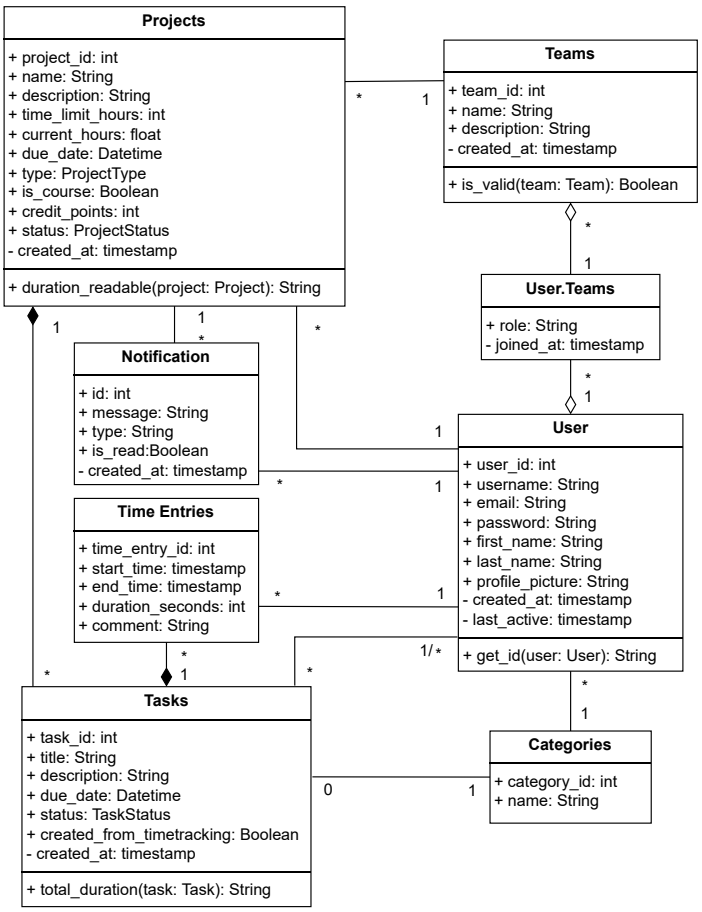

Abbildung 6.1: Klassendiagramm der langfristig zu speichernden Daten

6.2 Erläuterung

Die folgenden Tabelleneinträge dienen dazu, alle Entitäten im obigen Klassendiagramm weiter zu erläutern. Die grundlegenden Funktionalitäten werden von SQLAlchemy bereitgestellt, weitere Funktionalitäten wie to_dict( ) oder __repr__( ) wurden aufgrund von Redundanzen ausgelassen.

User $\langle E10 \rangle$

Beziehung	**Kardinalität**	**Erwartete Datenmenge**	**Beschreibung**
UserTeam <E20>	1:N	10–500 KB	Verknüpft Nutzer:in mit Teams über Zwischentabelle
TimeEntry <E30>	1:N	1–50 MB	Erfasste Arbeitszeiten des:der Nutzers:Nutzerin
Task <E50>	1/N:N	100–1.000 KB	Aufgaben, die dem:der Nutzer:in zugewiesen sind
Project <E60>	1:N	50–500 KB	Nutzer:in kann mehrere Projekte erstellen
Notification <E70>	1:N	50–500 KB	Benachrichtigungen für Aktionen von Nutzer:innen
Category <E80>	1:N	1–50 MB	Nutzer:in kann mehrere selbst definierte Projekte erstellen

UserTeam ⟨*E*20⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
User <E10>	N:1	10–500 KB	Verbindung zu genau einem:einer Nutzer:in
Team <E40>	N:1	10–500 KB	Verbindung zu genau einem Team

TimeEntry ⟨*E*30⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
User <E10>	N:1	10–500 KB	Arbeitszeit eines:einer Nutzer:in
Task <E50>	N:1	1–50 MB	Zeit wird konkreter Aufgabe zugeordnet

Team ⟨*E*40⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
UserTeam <E20>	1:N	10–500 KB	Verknüpfung zu Mitgliedern
Project <E60>	1:N	50–500 KB	Team kann mehrere Projekte besitzen

Task ⟨*E*50⟩

Beziehung	**Kardinalität**	**Erwartete Datenmenge**	**Beschreibung**
TimeEntry <E30>	1:N	1–50 MB	Erfasste Zeiten zu Aufgabe
User <E10>	N:1	10–500 KB	Aufgabe ist einer Person zugeordnet, kann im Team auch mehreren Nutzer:innen zugwiesen sein.
Project <E60>	N:1	50–500 KB	Aufgabe gehört zu Projekt
Category <E80>	0:1	10–100 KB	Optionale Zuordnung zu Kategorie

Project ⟨*E*60⟩

Beziehung	**Kardinalität**	**Erwartete Datenmenge**	**Beschreibung**
Task <E50>	1:N	1–50 MB	Enthält zu erledigende Tasks
Team <E40>	N:1	50–500 KB	Projekt wird von Team betreut
User <E10>	N:1	10–500 KB	Projektverantwortliche:r
Notification <E70>	0:N	50–500 KB	Benachrichtigung zu Projektaktionen

Notification ⟨*E*70⟩

Beziehung	**Kardinalität**	**Erwartete Datenmenge**	**Beschreibung**
User <E10>	N:1	10–500 KB	Info über relevante Ereignisse
Project <E60>	N:1	10–500 KB	Verknüpfung zu Auslöser-Projekt

Category ⟨*E*80⟩

Beziehung	**Kardinalität**	**Erwartete Datenmenge**	**Beschreibung**
User <E10>	N: 1	50-100 KB	Nutzer:in kann mehrere selbst definierte Kategorien erstellen
Task <E50>	0:N	10–500 KB	Aufgaben können einer Kategorie zugeordnet sein

7 Konfiguration

In diesem Kapitel werden alle erforderlichen Konfigurationen beschrieben, welche für einen reibungslosen Betrieb der Anwendung notwendig sind. Die Anwendung basiert auf dem Python-Framework Flask und setzt sich aus verschiedenen Komponenten zusammen, die modular miteinander kommunizieren. Die Konfigurationsdateien dienen sowohl zur Anpassung an verschiedene Umgebungen als auch zur Festlegung von sicherheitsrelevanten und infrastrukturellen Parametern.

Zentraler Bestandteil der Konfiguration ist die Datei `.env`, welche sich im Hauptverzeichnis der Anwendung befindet. Diese Datei enthält eine Vielzahl von Umgebungsvariablen, die für den Betrieb der Applikation notwendig sind. Zu den in der `.env`-Datei enthaltenen Einstellungen gehören unter anderem der geheime Schlüssel der Anwendung, sowie Pfade zu Ressourcen wie zum Beispiel zu den Profilbildern. Darüber hinaus wird über diese Datei auch der Zugriff auf den verwendeten Mailserver geregelt. Hierzu gehören beispielsweise der Hostname des Maildienstes, die Portnummer, Authentifizierungsdaten sowie die Absenderadresse. Diese Angaben sind insbesondere für Funktionen wie Passwortzurücksetzung oder Benachrichtigungen relevant.

Ein weiterer wichtiger Aspekt der Konfiguration ist die Datenbankanbindung. Die Anwendung verwendet SQLAlchemy als ORM-Bibliothek. Der Pfad zur Datenbank wird dynamisch aus dem Projektverzeichnis heraus generiert und über eine Umgebungsvariable bereitgestellt. Die Migration von Datenbankschemata wird mithilfe von Alembic durchgeführt. Die hierfür erforderlichen Einstellungen sind ebenfalls an die Umgebungsvariablen gekoppelt und werden bei jeder Migration automatisch berücksichtigt.

Die installierten Abhängigkeiten der Anwendung sind in der Datei `requirements.txt` festgehalten. Diese Datei befindet sich im Hauptverzeichnis des Projekts und listet alle für die Ausführung notwendigen Python-Pakete mit genauer Versionsangabe auf. Dazu gehören neben Flask selbst auch Erweiterungen für Authentifizierung, Autorisierung, E-Mail-Versand, Datenbankintegration, Migration sowie Testautomatisierung (`pytest`). Durch die Angabe fester Versionen wird eine einheitliche Umgebung gewährleistet, was insbesondere für die Nachvollziehbarkeit und Wartbarkeit des Systems von Bedeutung ist. Die Pakete können typischerweise automatisiert über einen Paketmanager installiert werden.

Die Anwendung ist so aufgebaut, dass sie sowohl auf einem lokalen Entwicklungsrechner als auch auf einem Produktionsserver betrieben werden kann. Zur lokalen Entwicklung genügt ein

Rechner mit Python-Unterstützung, während für den produktiven Betrieb zusätzliche Konfigurationen wie beispielsweise eine sichere Transportverschlüsselung, ein externer Maildienst sowie persistente Datenbankverbindungen notwendig sind.

Neben der serverseitigen Anwendung kommen im Frontend kleinere JavaScript-Dateien zum Einsatz. Diese übernehmen grundlegende Aufgaben wie die Validierung von Formulareingaben. Die Dateien befinden sich im Verzeichnis `frontend/static/js` und werden direkt in die entsprechenden HTML-Templates eingebunden.

8 Änderungen gegenüber Fachentwurf

Im Verlauf der Implementierung wurden keine grundlegenden Änderungen gegenüber dem im Fachentwurf beschriebenen Systemdesign erforderlich.

Das Feedback zum Fachentwurf wurde im weiteren Verlauf berücksichtigt und entsprechend umgesetzt. Insbesondere die Logik der Sequenzdiagramme im Kapitel 2 wurde im Hinblick auf die Aufgabenverteilung zwischen der Webanwendung und der Datenbank noch einmal überarbeitet und an die tatsächliche Implementierung angepasst.

Aufgrund von Änderungen in der Datenbank wurde Kapitel 6 entsprechend überarbeitet, um die neuen Gegebenheiten korrekt widerzuspiegeln.

Alle geplanten Komponenten und Abläufe konnten wie vorgesehen umgesetzt werden. Eine Anpassung des ursprünglichen Entwurfs war darüber hinaus nicht notwendig.

9 Erfüllung der Kriterien

In diesem Kapitel wird dargelegt, wie die im Pflichtenheft definierten Muss-, Soll- und Kann-Kriterien durch das entworfene System erfüllt bzw. nicht erfüllt werden. Dabei wird jeweils die konkrete Komponente aus Kapitel 3 dieses Dokuments benannt, die die Umsetzung ermöglicht.

9.1 Musskriterien

Die im Folgenden aufgeführten Kriterien stellen essentielle Anforderungen dar, die von unserem Produkt zwingend erfüllt werden müssen:

- **RM1**: Die Registrierung und Authentifizierung mit Passwortänderung und -wiederherstellung wird durch die Komponenten <C10> UI Layer, <C20> Business Logic und <C30> Database umgesetzt. <C10> stellt die Eingabeformulare bereit, <C20> verarbeitet die Anfragen und <C30> speichert und aktualisiert die Zugangsdaten persistent.
- **RM2**: Die Zeiterfassung mit Start-/Stopp-Funktion wird durch die Komponenten <C10> UI Layer, <C20> Business Logic und <C30> Database umgesetzt. <C10> stellt die Bedienelemente für Start, Pause, Fortsetzen und Stop zur Verfügung. <C20> verarbeitet die Zeiterfassungslogik und übergibt die Daten zur Speicherung an <C30>.
- **RM3**: Die manuelle Erfassung und nachträgliche Bearbeitung von Zeiteinträgen wird durch die Komponenten <C10> UI Layer, <C20> Business Logic und <C30> Database umgesetzt. Über die Oberfläche kann der:die Nutzer:in neue Einträge manuell hinzufügen oder bestehende bearbeiten. <C20> übernimmt die Validierung und leitet die Daten an <C30> weiter, wo sie gespeichert oder aktualisiert werden.
- **RM4**: Das Anlegen, Bearbeiten und Löschen von Projekten sowie das Festlegen des Projekttyps (Solo- oder Teamprojekt) wird durch die Komponenten <C10> UI Layer, <C20> Business Logic und <C30> Database ermöglicht. <C10> stellt Eingabemasken bereit und erlaubt die Auswahl des Projekttyps. Die eingegebenen Daten werden durch <C20> verarbeitet und anschließend in <C30> gespeichert oder bei Bedarf aktualisiert bzw. gelöscht.

- **RM5**: Die eindeutige Zuordnung eines Zeiteintrags zu einem Task und die Möglichkeit, mehrere Zeiteinträge pro Task zu speichern, wird durch <C20> Business Logic ermöglicht und in <C30> der Database gesichert.
- **RM6**: Das Anlegen und Verwalten mehrerer Tasks pro Projekt wird durch die Komponenten <C10> UI Layer, <C20> Business Logic und <C30> Database ermöglicht. Über <C10> kann der:die Nutzer:in neue Tasks erstellen, bearbeiten oder löschen. <C20> übernimmt die Prüfung der Eingaben und organisiert die korrekte Zuordnung zum jeweiligen Projekt. Die Informationen zu den Tasks werden schließlich in <C30> hinterlegt.
- **RM7**: Die Darstellung aller erfassten Zeiten in einer Kalenderansicht wird durch die Komponente <C10> UI Layer ermöglicht, welche mithilfe der Open-Source-Bibliothek FullCalendar die in <C30> Database gespeicherten, projektbezogenen Zeiteinträge visuell strukturiert.
- **RM8**: Fortschrittsdiagramme, die den Bearbeitungsstand pro Projekt visuell in Prozent darstellen, werden durch die Komponenten <C20> Business Logic und <C10> UI Layer umgesetzt. <C20> berechnet die Fortschrittswerte auf Basis der in <C30> Database gespeicherten Ist- und Soll-Zeiten. Die visuelle Darstellung erfolgt in <C10>.
- **RM9**: Ein visueller Ist-Soll-Vergleich der prozentualen Zeitverteilung über mehrere Projekte hinweg wird durch die Komponente <C20> Business Logic auf Basis der in <C30> Database gespeicherten Zeitdaten berechnet und in <C10> UI Layer visuell dargestellt.
- **RM10**: Die Möglichkeit, Teamprojekte mit mehreren Mitgliedern eines bestehenden Teams zu teilen, sofern jedes Mitglied registriert ist, wird durch die Komponente <C40> Team and Role Management ermöglicht, wobei Nutzer:innen zu Projekten zugewiesen und entsprechende Rechte vergeben werden.
- **RM11**: Die Zuweisung von Tasks an einzelne Mitglieder innerhalb eines Teamprojekts wird durch die Komponente <C40> Team and Role Management ermöglicht, die die Rollenlogik verwaltet und die Zuweisung bereitstellt.
- **RM12**: Das Exportieren von Zeitdaten (z. B. im PDF- oder CSV-Format) wird ermöglicht, indem die Komponente <C10> UI Layer die Exportbuttons bereitstellt. Die serverseitige Generierung der Dateien erfolgt durch <C20> Business Logic auf Basis der in <C30> Database gespeicherten Projekt- und Zeiteinträge.
- **RM13**: Die Bereitstellung der Benutzeroberfläche in englischer Sprache wird durch die Komponente <C10> UI Layer gewährleistet. Alle UI-Elemente wie Buttons, Menüs und Beschriftungen wurden in englischer Sprache implementiert.

- **RM14**: Die plattformunabhängige Lauffähigkeit der Anwendung in Google Chrome ab Version 136.0.7103.93 wird durch die Komponente <C10> UI Layer sichergestellt. Die UI-Elemente wurden gezielt so implementiert, dass sie in diesem Browser korrekt dargestellt werden und sämtliche Funktionalitäten zuverlässig nutzbar sind.
- **RM15**: Das sichere und dauerhafte Speichern der Zeitdaten in einer Datenbank (SQLite) wird durch die Komponente <C30> Database gewährleistet, welche SQLite als persistente Datenbank nutzt.

9.2 Sollkriterien

Die Umsetzung der nachfolgenden Kriterien wird für das Produkt angestrebt, auch wenn sie nicht zwingend erforderlich sind:

- **RS1**: Die Kategorisierung von Tasks wird durch die Komponenten <C10> UI Layer, <C20> Business Logic und <C30> Database ermöglicht. <C10> erlaubt das Festlegen von Kategorien, während <C20> die Verknüpfung der Kategorien mit Tasks verarbeitet und an <C30> zur Speicherung übergibt.
- **RS2**: Die Bereitstellung einer Such- und Filterfunktion zur Filterung von Projekten und Tasks nach Eigenschaften wird nicht umgesetzt.
- **RS3**: Die Filterung systeminterner Benachrichtigungen nach dem Benachrichtigungstyp wird durch die Komponenten <C10> UI Layer und <C30> Database umgesetzt. <C10> stellt die Auswahl und Anwendung von Filteroptionen bereit. Die Benachrichtigungstypen, nach denen gefiltert wird, sind in <C30> gemeinsam mit den Benachrichtigungen gespeichert.
- **RS4**: Das Senden systeminterner Benachrichtigungen bei Teamprojekten (z. B. bei Task-Zuweisungen, Teambeitritten oder neuen Kommentaren) wird durch die Komponente <C50> Notification ermöglicht. Sie erkennt relevante Ereignisse und generiert automatisch passende Nachrichten. Die Ereignisse, die Benachrichtigungen auslösen, entstehen durch Aktionen innerhalb der Komponente <C40> Team and Role Management.
- **RS5**: Die Berechnung von Zielwerten für den wöchentlichen Lernaufwand durch Eingabe der Leistungspunkte für ein Projekt wird nicht umgesetzt.
- **RS6**: Das Anmelden von Nutzer:innen über OAuth (z. B. Google-Account) wird nicht umgesetzt.
- **RS7**: Die Unterscheidung aktiver und inaktiver Projekte wird durch die Komponenten <C10> UI Layer, <C20> Business Logic und <C30> Database umgesetzt. <C10> stellt die Schaltflächen zur Auswahl des Projektstatus bereit, <C20> verarbeitet die Statusänderung und überträgt sie zur Speicherung an <C30>.
- **RS8**: Die Lauffähigkeit der Anwendung auf verschiedenen Endgeräten wird nicht umgesetzt.
- **RS9**: Die Konfigurierbarkeit der Frequenz der Benachrichtigungen zum Ist-Soll-Vergleich (z. B. täglich, wöchentlich) wird nicht umgesetzt.

9.3 Kannkriterien

Die folgenden Kriterien stellen optionale Erweiterungen dar, die für das Erreichen des Projektziels nicht zwingend erforderlich sind:

- **RC1**: Eine Farbanpassung für Projekte, Kalender und Diagramme wird nicht umgesetzt.
- **RC2**: Die Bereitstellung eines automatischen Backups der Zeitdaten wird nicht umgesetzt.
- **RC3**: Die Bereitstellung eines Balkendiagramms zur Auswertung des Zeitaufwands pro Projekt wird durch die Komponenten <C10> UI Layer und <C30> Database ermöglicht. Die Visualisierung erfolgt in <C10> mithilfe der Open-Source-Bibliothek Chart.js. Die dafür benötigten Zeitdaten werden direkt aus <C30> geladen.
- **RC4**: Die korrekte, ortsunabhängige Darstellung von Zeiten wird durch die Komponenten <C10> UI Layer, <C20> Business Logic und <C30> Database grundsätzlich ermöglicht, sofern die Serverzeit korrekt konfiguriert ist. Zeiteinträge werden mithilfe der Standardfunktion `datetime.now()` aus der Python-Bibliothek zum Zeitpunkt der Erfassung mittels <C20> in <C30> gespeichert und durch <C10> angezeigt. Diese Funktion basiert auf der lokalen Systemzeit des Servers. Eine automatische, zeitzonenunabhängige Zeiterfassung wird nicht umgesetzt.
- **RC5**: Die Bereitstellung der Benutzeroberfläche in deutscher Sprache wird nicht umgesetzt.
- **RC6**: Das Versenden wöchentlicher Benachrichtigungen zum Ist-Soll-Vergleich, wenn individuell festgelegte Zeitziele für Projekte deutlich unterschritten, erreicht oder überschritten werden, wird nicht umgesetzt.
- **RC7**: Die Darstellung innerhalb eines Teamprojekts, welche Mitglieder an welchen Tasks arbeiten und wie viel Zeit sie investiert haben, wird durch die Komponenten <C10> UI Layer und <C40> Team and Role Management ermöglicht. Über <C40> werden den Teammitgliedern Tasks zugewiesen, und in <C10> werden die Taskzuweisungen sowie die investierte Zeit angezeigt.
- **RC8**: Die Bereitstellung einer Umschaltfunktion zwischen Dark- und Light-Mode wird durch die Komponente <C10> UI Layer umgesetzt. Der:Die Nutzer:in kann zwischen beiden Designs wechseln; die vollständige Ausarbeitung des Light-Modes steht jedoch noch aus.

10 Glossar

Benutzeroberfläche Der Teil einer Software, den Nutzer:innen sehen und bedienen können.

CPU-Kern(e) Ein Verarbeitungseinheit innerhalb eines Prozessors (CPU), die eigenständig Rechenoperationen ausführen kann. Mehrere Kerne ermöglichen parallele Verarbeitung.

CSS3 Cascading Style Sheets – eine Stylesheet-Sprache zur Gestaltung von HTML-Inhalten.

CSV „Comma-Separated Values“ – ein Dateiformat zur Speicherung tabellarischer Daten in Textform.

Datenbanksystem Software zur Speicherung, Verwaltung und Abfrage strukturierter Daten.

Deployment Der Prozess der Veröffentlichung und Bereitstellung einer Softwareanwendung auf einem Server oder in der Cloud zur Nutzung durch Endnutzer:innen.

Export Funktion zur Ausgabe und Speicherung von Daten aus der Anwendung in verschiedenen Dateiformaten, um sie weiterzuverwenden oder zu archivieren.

Flask Leichtgewichtiges Web-Framework für Python zur Entwicklung von Webanwendungen (siehe `https://flask.palletsprojects.com/`).

HTML5 Die fünfte Version der Hypertext Markup Language zur Strukturierung von Webseiteninhalten.

HTTPS „Hypertext Transfer Protocol Secure“ – eine verschlüsselte Version des HTTP-Protokolls, die für eine sichere Datenübertragung im Web sorgt. Erkennbar an der URL mit „https://“.

JavaScript Eine Programmiersprache zur Umsetzung interaktiver Funktionen im Webbrowser.

OAuth Ein offener Standard zur sicheren Autorisierung, der es Nutzern ermöglicht, sich über Drittanbieter-Dienste wie Google oder GitHub bei Anwendungen anzumelden, ohne eigene Zugangsdaten zu hinterlegen.

ORM „Object-Relational Mapping“ – eine Technik, die Objekte in Programmiersprachen mit Datenbanktabellen verknüpft, um Datenbankoperationen objektorientiert durchzuführen.

PDF „Portable Document Format“ – ein weit verbreitetes Dateiformat zur originalgetreuen Darstellung und Weitergabe von Dokumenten, unabhängig vom verwendeten Betriebssystem oder Gerät.

RAM „Random Access Memory“ – der Arbeitsspeicher eines Computers, in dem laufende Programme und Daten temporär gespeichert werden.

SMTP-Dienst „Simple Mail Transfer Protocol“ – ein Protokoll zum Versenden von E-Mails über das Internet.

SQLAlchemy Ein in Python geschriebenes Toolkit zur Datenbankprogrammierung, das die Arbeit mit relationalen Datenbanken vereinfacht (siehe `https://www.sqlalchemy.org/`).

SQLite Eine leichtgewichtige, serverlose Datenbank, die direkt in Anwendungen eingebettet wird (siehe `https://www.sqlite.org/`).

Systeminterne Benachrichtigung Eine Nachricht, die direkt innerhalb der Anwendung angezeigt wird – z. B. als Hinweis, Pop-up oder Symbol in der Benutzeroberfläche. Sie informiert den Nutzer über relevante Ereignisse, ohne eine externe E-Mail oder Push-Nachricht zu verwenden.

VM-Server Ein Server, der mehrere virtuelle Maschinen (VMs) auf einer physischen Hardware betreibt. Jede VM läuft wie ein eigener Computer mit eigenem Betriebssystem.

Webanwendung Eine über den Webbrowser nutzbare Software, die keine Installation auf dem eigenen Gerät erfordert.

Webbrowser Ein Programm zur Anzeige und Nutzung von Webseiten (z.B. Chrome, Firefox, Edge).