Abschlussprüfung Sommer 2024

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Portal zur Eintragung von Urlaub- und Krankheitsabwesenheiten

Weiterentwicklung von TIME+PLUS

Abgabetermin: Stuttgart, den 04.06.2024

Prüfungsbewerber:
Luis Scheurenbrand
Friedrichstraße 39
73770 Denkendorf

Ausbildungsbetrieb:
Ruthardt Softwaretechnik GmbH
Friedrich-List-Straße 34
70771 Leinfelden-Echterdingen

Inhaltsverzeichnis

Abbildungsverzeichnis **III**

Tabellenverzeichnis **IV**

Listings **V**

Abkürzungsverzeichnis **VI**

1 Einleitung **1**
1.1 Projektumfeld . . . 1
1.2 Projektziel . . . 1
1.3 Projektbegründung . . . 1
1.4 Projektschnittstellen . . . 1

2 Projektplanung **2**
2.1 Projektphasen . . . 2
2.2 Abweichungen vom Projektantrag . . . 2
2.3 Ressourcenplanung . . . 2
2.4 Entwicklungsprozess . . . 3

3 Analysephase **4**
3.1 Ist-Analyse . . . 4
3.2 Wirtschaftlichkeitsanalyse . . . 5
3.2.1 Projektkosten . . . 5
3.2.2 Amortisationsdauer . . . 6

4 Entwurfsphase **6**
4.1 Zielplattform . . . 6
4.2 Architekturdesign . . . 7
4.2.1 UI-Layer . . . 7
4.2.2 Business Logic-Layer . . . 7
4.2.3 Data Access-Layer . . . 7
4.2.4 Model-Layer . . . 7
4.3 Entwurf der Benutzeroberfläche . . . 7
4.4 Auswahl der Kalenderansicht . . . 8
4.4.1 Vertrautheit mit der Technologie . . . 8
4.4.2 Dokumentation . . . 8
4.4.3 Codeverständlichkeit . . . 8
4.4.4 Nutzerfreundlichkeit . . . 9
4.5 Datenmodell . . . 9
4.6 Data Access-Layer . . . 9

4.7 Geschäftslogik 9

5 Implementierungsphase 11
5.1 Maßnahmen zur Qualitätssicherung 11
5.2 Anpassung der Datenbankmodelle 11
5.3 Erweiterung des Data Access-Layers 11
5.4 Erweiterung der Geschäftslogik im Business-Layers 11
5.5 Implementierung der Web-Endpunkte 12

6 Dokumentation 13

7 Fazit 13
7.1 Abnahme 13
7.2 Ausblick 13

Eidesstattliche Erklärung 14

A Anhang i
A.1 Klasse: AbwesenheitenLogic i
A.2 Entwicklerdokumentation iv
A.3 Endnutzerdokumentation iv
A.4 Abwesenheitsübersicht vi
A.5 Abwesenheitsantrag vi
A.6 Abwesenheitsdetails vii
A.7 Unit-Tests viii

Abbildungsverzeichnis

1 Urlaubsauswertung der SQL Server Reporting Services 4
2 Urlaubsauswertung der SQL Server Reporting Services 5
3 Firmeninterne Referenz-Architektur ohne Fachbezug 8
4 Prozess des Einreichens einer Abwesenheit 10
5 Entwicklerdokumentation iv
6 Endnutzerdokumentation v
7 Abwesenheitsübersicht vi
8 Abwesenheitsantrag vi
9 Abwesenheitsdetails für Vorgesetzen vii

Tabellenverzeichnis

1 Zeitplanung 2
2 Entscheidungsmatrix 9

Listings

1 Klasse: AbwesenheitenLogic i
2 Klasse: AbwesenheitenTests viii

Abkürzungsverzeichnis

API	Application Programming Interface
CRM	Customer-Relationship-Management
CDO	cobra Domainobject
CSS	Cascading Style Sheets
IDE	Integrated Development Environment
MVC	Model View Controller
JSON	JavaScript Object Notation
ORM	Object-Relational Mapping
SQL	Structured Query Language
AU	Arbeitsunfähigkeitsbescheinigung
CI	Continuous Integration

1 Einleitung

1.1 Projektumfeld

Die Ruthardt Softwaretechnik GmbH ist ein mittelständiger Betrieb, spezialisiert auf die Digitalisierung von Geschäftsprozessen. Sie bildet zusammen mit der Ruthardt Systemhaus GmbH und der Ruthardt CRM GmbH unter der Ruthardt Holding verschiedene Aspekte der Digitalisierung ab. Bei der Ruthardt Softwaretechnik GmbH wird meist die Programmiersprache `C#` verwendet. Als Auftraggeber agiert die Ruthardt Holding GmbH.

1.2 Projektziel

Zur Erfassung von Abwesenheiten soll eine Webseiten-Portal erstellt werden, bei dem die Mitarbeiter ihre Urlaube und Arbeitsunfähigkeiten einreichen können. Dadurch werden bestehende, mehrschrittige Prozesse abgelöst und Vorgänge teilautomatisiert und vereinfacht. Vorgesetzte sollen über die Abwesenheiten zeitnah informiert werden und Urlaubsanträge verwalten können. Bei anstehenden Abwesenheiten soll im Postfach des Mitarbeiters automatisch eine Abwesenheitsnotiz hinterlegt werden.

1.3 Projektbegründung

Die Erfassung der Abwesenheiten soll durch das neue Portal vereinfacht und Doppelstrukturen vermieden werden. Außerdem soll sowohl den Mitarbeitern, als auch den Vorgesetzten eine bessere Übersicht der vergangenen und anstehenden Abwesenheiten gewährt werden. Durch Teilautomatisierung sollen auch Fehler verhindert werden. So muss ein Mitarbeiter bei Antreten des Urlaubs aktuell manuell eine Abwesenheitsnotiz in seinem Outlook hinterlegen und für jeden Tag manuell die genommenen Urlaubsstunden vermerken.

1.4 Projektschnittstellen

Das Portal soll als Erweiterung der existierenden TIME+PLUS Anwendung realisiert werden. Im Vordergrund steht die Webseite der Anwendung, welche mit dem Framework ASP.NET Core, auf Basis der Programmiersprache `C#`, implementiert ist. Die Nutzeroberflächen sind in Razor-Views, JavaScript und CSS gestaltet. Die Programmdaten werden auf einem Microsoft SQL-Server gespeichert, und sind mit der Anwendung über das ORM linq2db verbunden. Nutzer der Erweiterung sind alle Mitarbeiter und Vorgesetzen, welche bereits Zugriff zur Webseite haben.

2 Projektplanung

2.1 Projektphasen

Die Umsetzung des Projekt fand vom 15.05.2024 bis zum 03.06.2024 statt. Hierbei standen mir insgesammt 80 Stunden zur Verfügung, welche in verschiedene Phasen unterteilt wurden.

Projektphase	**Geplante Zeit**
Analysephase	4 h
Entwurfsphase	10 h
Implementierungsphase	52 h
Abnahmetest der Fachabteilung	4 h
Erstellen der Dokumentation	10 h
Gesamt	**80 h**

Tabelle 1: Zeitplanung

2.2 Abweichungen vom Projektantrag

In dem Projektantrag wurde nur die Aspekte der Abwesenheit bei Urlaub und Arbeitsunfähigkeit beschrieben. Hinzu kam bei der Umsetzung die Abbildung der Ausbildungsstunden. Abwesenheiten durch Arbeitsunfähigkeit können zudem auch mit einem Vermerk versehen werden, welcher angibt, ob eine Arbeitsunfähigkeitsbescheinigung (AU) vorliegt. Zudem ist es auch möglich nur Teile eines Arbeitstages (Stunden) als Abwesenheit zu deklarieren. Die Prüfung der Software-Code-Qualität wird auf die Nutzung des Analyzers StyleCop.Analyzer beschränkt, da dieser Firmenweit priorisiert wird.

2.3 Ressourcenplanung

In der folgenden Übersicht werden die verwendeten Ressourcen aufgelistet, welche für die Umsetzung genutzt wurden. Es wurde darauf geachtet auf bereits vorhandene und innerhalb der Firma breit genutzte Software- und Hardwarekomponenten zu setzten, damit meine Kollegen keine Probleme bei der Einarbeitung für eventuelle Anpassungen oder Wartungen haben. Neu akquirierte Software ist so gekennzeichnet und wurde kostenfrei erworben.

Hardware

- Laptop
- Büroarbeitsplatz mit Dockingstation
- HyperV-Server mit RDP-Anbindung

- SQL-Server

Software

- Windows 11 Pro - Betriebsystem
- Visual Studio 2022 Professional - Entwicklungsumgebung
- Jetbrains ReSharper 2024.1 - IDE-Erweiterung
- CDO-Gen - Interner Datenbankmapper
- SQL Server Management Studio - Datenbankverwaltung
- fullcalendar - JavaScript Bibliothek zur Visualisierung eines Kalenders (Neuakquise)
- Microsoft.Graph nuget-Paket - Programm-Schnittstelle zu Microsoft Exchange via GraphAPI (Neuakquise)
- Azure.Identity nuget-Paket - Authentifizierungsbibliothek für die GraphAPI (Neuakquise)
- cobra CRM PRO XL 2023 - CRM-System
- Git - Verteilte Versionsverwaltung
- TIME+PLUS - Arbeitszeitplanung
- SourceTree - Git-Helper
- Visual Studio Code - Entwicklungsumgebung LaTeX
- MiKTex - TeXRenderer

Personal

- Anwendungsentwickler - Umsetzung des Projektes
- Anwendungsentwickler - Review des Codes
- Geschäftsführer - Abnahme

2.4 Entwicklungsprozess

Zum Zeitpunkt des Projektantrags wurde in Absprache mit dem Entwicklungsleiter beschlossen das Wasserfallmodel als Entwicklungsprozess zu nutzen. Die Vorgaben waren somit vor Umsetzungsbegin klar gesetzt und die Entwicklung wurde linear durchgeführt.

3 Analysephase

3.1 Ist-Analyse

Bisher wurden Abwesenheiten über mehrstufige Prozesse gemeldet und erfasst. Für Urlaubsanträge wurde ein interner Exchange Nutzer (abwesenheit@ruthardt.eu) für den Zeitraum der Abwesenheit zu einer Besprechung eingeladen. Vorgesetzte wurden über diese Einladung informiert und konnten die Besprechung annehmen oder ablehnen um den Status der Urlaubsgenehmigung dem Mitarbeiter mitzuteilen. Arbeitsunfähigkeiten wurden den Vorgesetzten direkt über eine spezielle Adresse (krank@ruthardt.eu) per E-Mail mitgeteilt. Damit Kundenkontakte nicht durch die Abwesenheit verzögert wurden, musste der Arbeitnehmer manuell in seinen Outlook Einstellungen eine automatische Antwort-E-Mail aktivieren. Dies wurde, insbesondere im Krankenstand der Mitarbeiter, oft vergessen. Im Nachzug musste der Arbeitnehmer im cobra CRM System die Zeiterfassungsdatensätze der Abwesenheitstage mit den entsprechenden Abwesenheitsstunden versehen. Die Zeiterfassungsdatensätze wurden über die Windows Aufgabenplanung automatisch erzeugt und mit den zu erbringenden „Soll Stunden“ versehen. Dieser Task greift nur 14 Tage in die Zukunft. Eine Änderung der Datensätze durch den Mitarbeiter wird nach der Monatlichen Abrechnung blockiert, daher musste der Mitarbeiter sich zeitnah zu seiner Abwesenheit um die Bearbeitung kümmern.

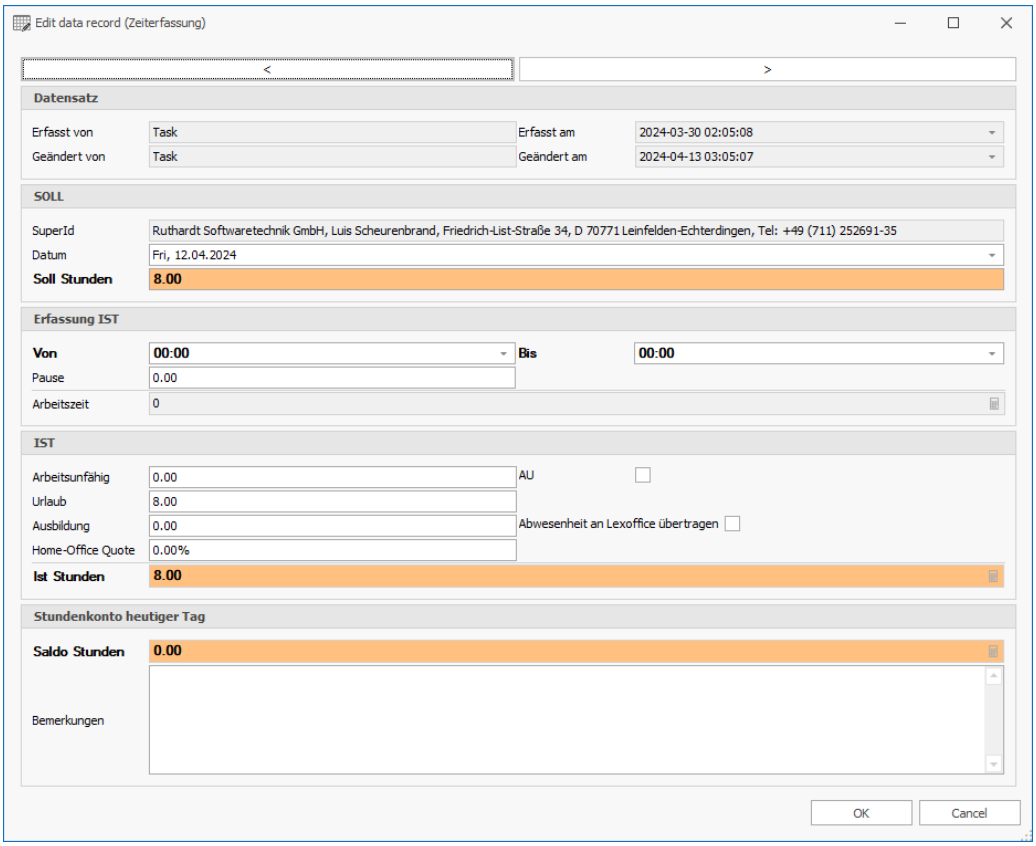
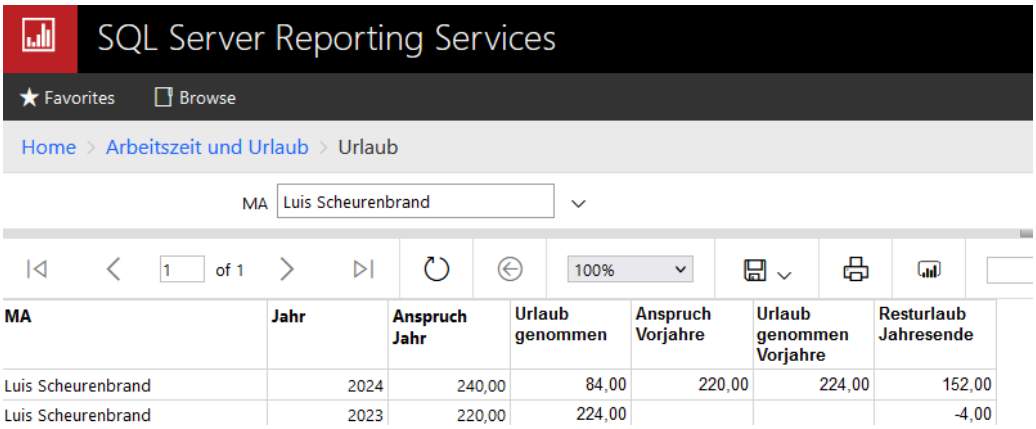


Abbildung 1: Urlaubsauswertung der SQL Server Reporting Services

Zur weiteren Übersicht konnten sich Mitarbeiter, im Auswertungsportal des SQL-Servers, eine Einsicht der bereits gebuchten Urlaubsstunden erhalten.



MA	Jahr	Anspruch Jahr	Urlaub genommen	Anspruch Vorjahre	Urlaub genommen Vorjahre	Resturlaub Jahresende
Luis Scheurenbrand	2024	240,00	84,00	220,00	224,00	152,00
Luis Scheurenbrand	2023	220,00	224,00			-4,00

Abbildung 2: Urlaubsauswertung der SQL Server Reporting Services

3.2 Wirtschaftlichkeitsanalyse

Da die TIME+PLUS Anwendung in erster Linie eine rein intern genutze Anwendung ist und das geplante Portal nicht für Kunden angedacht ist, ist eine Wirtschaftlichkeitsprüfung zum aktuellen Zeitpunkt erschwert möglich. Die Umsetzungskosten stehen gegenüber der Zeitersparniss aller Mitarbeiter und Vorgesetzen, welche dieses Portal zukünfig nutzen werden. Zusätzlich ist die Qualitätssteigerung, da automatische Abwesenheitsmeldungen in den Postfächern nicht mehr vergessen werden können. Falls die TIME+PLUS Anwendung zukünfig vertrieben wird, können die neuen Features als Verkaufsargument gelten.

3.2.1 Projektkosten

Die Kosten des Projekts bestehen hauptsächlich aus den Personalkosten. Für den Auszubildenden ist von etwa 1.500 € und für den Entwicklungsleiter und andere Anwendungsentwicklern von etwa 5.000 € Lohn & Versicherungsbeiträgen auszugehen.

$$8\ \text{h/Tag} \cdot 220\ \text{Tage/Jahr} = 1760\ \text{h/Jahr} \tag{1}$$

$$1500\,€/\text{Monat} \cdot 13\ \text{Monate/Jahr} = 19500\,€/\text{Jahr} \tag{2}$$

$$\frac{19500\,€/\text{Jahr}}{1760\ \text{h/Jahr}} \approx 11{,}08\,€/\text{h} \tag{3}$$

$$5000\,€/\text{Monat} \cdot 13\ \text{Monate/Jahr} = 65000\,€/\text{Jahr} \tag{4}$$

$$\frac{65000\,€/\text{Jahr}}{1760\ \text{h/Jahr}} \approx 36{,}93\,€/\text{h} \tag{5}$$

Für weitere Ressourcennutzung wird ein pauschaler Stundensatz von 15 € angenommen. Es ergeben sich also Stundenkosten für den Entwickler von etwa 26,08 € und für Kollegen von 51.93 €. Die Umsetzungszeit beträgt 80 Stunden. Davon sind 8 Stunden für Abnahme und Besprechungen angesetzt. Damit ergeben sich Gesamtkosten von 2381.84 €.

3.2.2 Amortisationsdauer

Wir gehen davon aus, dass jeder Mitarbeiter durch die Nutzung des Portals 20 Minuten pro Monat spart. Bei Vorgesetzen gehen wir von eine Stunde pro Monat aus. Wir gehen von zwei Vorgesetzen und 18 Mitarbeitern aus. Damit ist bei mit einer monatlichen Ersparniss von 5.6 Stunden zu rechnen. Da diese Arbeitszeit wieder abrechenbar genutzt werden kann, gehen wir von 40 € Opportunitätskosten pro Stunde die eingespart werden, aus.

$$2\dot{6}0min + 18\dot{2}0min = 5.6\text{h} \tag{6}$$

Dadurch ergibt sich eine monatliche Einsparung von

$$5.6\text{h} \cdot 40\,€/\text{h} = 224\,€ \tag{7}$$

Die Amortisationszeit beträgt also $\frac{2381.84\,€}{224\,€/\text{Monat}} \approx 10.6$ Monate.

4 Entwurfsphase

4.1 Zielplattform

Da das Portal als Erweiterung für die existierende TIME+PLUS Anwendung angedacht war, waren äußeren Rahmenbedingungen vorgegeben. Der Hauptteil der Entwickelung wird in der Programmiersprache `C#` realisiert. Dies Umfasst den Datenbankzugriff via linq2db und einen Webserver, welcher das ASP.NET Core Framework nutzt. Die Abwesenheitsnotiz im Exchange der Nutzer wird über die Microsoft-GraphAPI gesetzt. Microsoft bietet hierzu eine Client-Bibliothek an.

4.2 Architekturdesign

Die Architektur des Projekts ist aufgeteilt in 4 wesentliche Teilkomponenten: UI, Business Logic, Data Access und Model. Konkret werden diese Komponenten in .NET Teilprojekten definiert und verknüpft. In Abbildung 3 finden Sie eine besipielhafte Darstellung dieser Architektur ohne Fachbezug.

4.2.1 UI-Layer

Die UI-Komponente gliedert sogenannte „MVC-Controller“ anhand der Funktionen der Nutzeroberfläche. Hier werden die Endpunkte, welche der Browser anspricht definiert und Ansichten gestaltet. Der UI-Layer beinhaltet besonders Plattform- und Technologieabhängigen Code und ihre Abhängigkeiten (ASP.NET, Razor Views, Style-Sheets).

4.2.2 Business Logic-Layer

Der Business Logic-Layer beinhaltet die fachliche Logik zur Datenverarbeitung und -aufbereitung. Diese Komponente ist Technologieunabhängig gestaltet, und ihr Code somit wiederverwertbar. So soll gewährleistet werden, dass die Anwendung nicht zu sehr auf den Bestand von bestimmten Plattformen und Technologien beruht.

4.2.3 Data Access-Layer

Im Data Access befinden sich technologieabhängige Funktionen zum Abruf der Daten. Diese Komponente soll frei austauschbar sein, um den Umstieg auf andere Datenbanksysteme oder Mockbackends für automatisierte Test zu gewährleisten.

4.2.4 Model-Layer

Der Model-Layer hält Datenbankmappings und Entitätenklassen und soll frei von Abhängigkeiten sein, damit jeder andere Layer diesen referenzieren kann.

4.3 Entwurf der Benutzeroberfläche

Als Hauptansichtselement war ein Kalender angedacht, auf dem die Abwesenheiten gelistet sind, und neue eingetragen werden können. Die Gestalltung basiert stark auf der Programmbibliothek, die dafür genutzt wird und der Gestaltungsfreiheit, welche diese Bibliothek uns lässt.

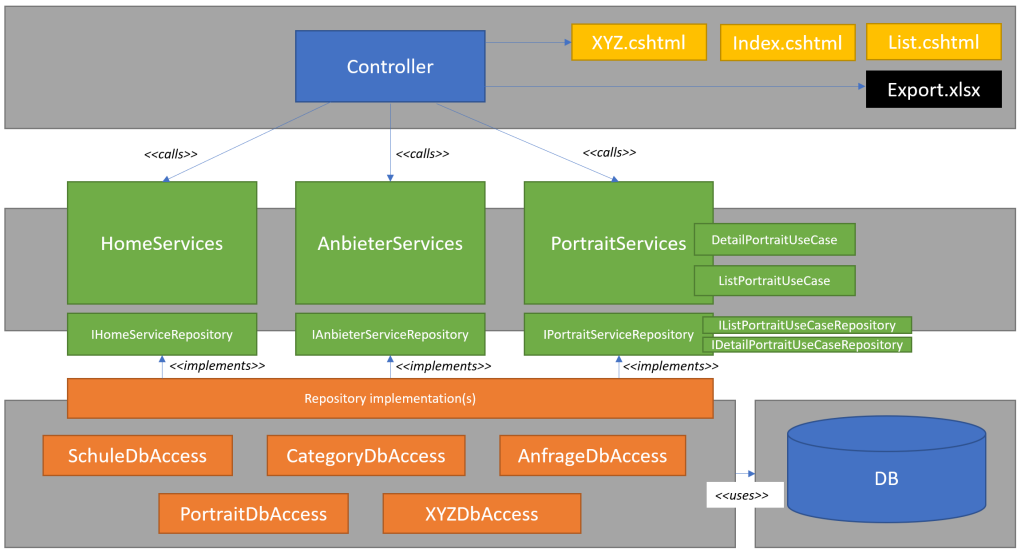


Abbildung 3: Firmeninterne Referenz-Architektur ohne Fachbezug

4.4 Auswahl der Kalenderansicht

Zur Visualisierung und Bearbeitung der Abwesenheiten soll eine Kalenderansicht geschaffen werden. Hierbei wurden mögliche Programmbibliotheken geprüft, getestet und gegenübergestellt. Bei der Betrachtung wurden folgende Kriterien berücksichtigt:

4.4.1 Vertrautheit mit der Technologie

Die Anwendungsentwickler, welche später Wartungen und Anpassungen durchführen, sollen mit der Technologie vertraut sein. Dieses Kriterium agierte ebenfalls als direkt-Ausschlusskriterium, wenn die Nutzung einer Bibliothekt eine vollständige Umstrukturierung der Anwendung verlangt hätte.

4.4.2 Dokumentation

Eine nutzbare Dokumentation der Funktionen, sowie Anleitungen und Beispielcode erleichtert den Entwicklern die Arbeit mit der Programmbibliothek ungemein. Beispielprogrammcode ermöglicht es dem Entwickler schneller zum Ziel zu gelangen.

4.4.3 Codeverständlichkeit

Der Entwickler soll sich auf die Umsetzung der Funktionalität konzentrieren können. Die Programmbibliothek sollte ihm dabei nicht im Weg stehen.

4.4.4 Nutzerfreundlichkeit

Die Bibliothekt soll für den Endnutzer einfach bedienbar sein und die Nutzung der Webseite nicht unnötig beeinträchtigen. Dahergehend ist die Größe der Bibliothek ebenfalls entscheidend, da Beispielsweise große JavaScript-Dateien zu Verzögerungen beim Laden der Ansicht führen können.

Eigenschaft	Gewichtung	DevExtreme	ToastUi Calendar	fullcalendar
Vertrautheit	5	4	3	4
Dokumentation	2	3	5	2
Codeverständlichkeit	3	0	4	5
Nutzerfreundlichkeit	4	2	2	5
Nutzwert:	**14**	**34**	**24**	**46**

Tabelle 2: Entscheidungsmatrix

4.5 Datenmodell

Das aktuelle Datenbankmodel hinter der TIME+PLUS Anwendung ist größtenteils ausreichend. Die Verarbeitungslogik soll Bezug auf die Zeiterfassungstabelle nehmen. Diese hält in Relation mit einem Mitarbeiter zu erbringenden und den tatsächlich erbrachten Stunden, Abwesenheiten jedlicher Art, so wie einen Vermerk über das Vorliegen einer Arbeitsunfähigkeitsbescheinigung. Lediglich ein Vermerk über den Genehmigungsstatus eines Urlaubsantrags muss ergänzt werden.

4.6 Data Access-Layer

Es müssen neue Methoden zur Ermittlung und Bearbeitung der Datenbankmodelle geschaffen werden.

4.7 Geschäftslogik

Da Zeiterfassungsdatensätze jeweils nur einen Tag darstellen, soll die Geschäftslogik in der Lage sein angrenzende Abwesenheiten zu gruppieren, damit der Nutzer diese sinnvoll dargestellt bekommt. Außerdem sollen eingereichte Abwesenheiten in Tage zerstückelt werden, um Zeiterfassungsdatensätze anzulegen und in diesen die Abwesenheiten zu vermerken. Urlaubsanträge sollen hier genehmigt oder abgelehnt werden können und als Folge soll der Mitarbeiter per E-Mail über die Entscheidung benachrichtigt werden. Damit Abwesenheitsnotizen automatisch gesetzt werden können, muss hier eine Anbindung an die Graph-API geschaffen werden, welche den Abwesenheitszeitraum bis zum nächsten Arbeitstag bestimmen kann.

Abbildung 4 zeigt den Prozessablauf beim Einreichen einer Abwesenheit als Flussdiagramm.

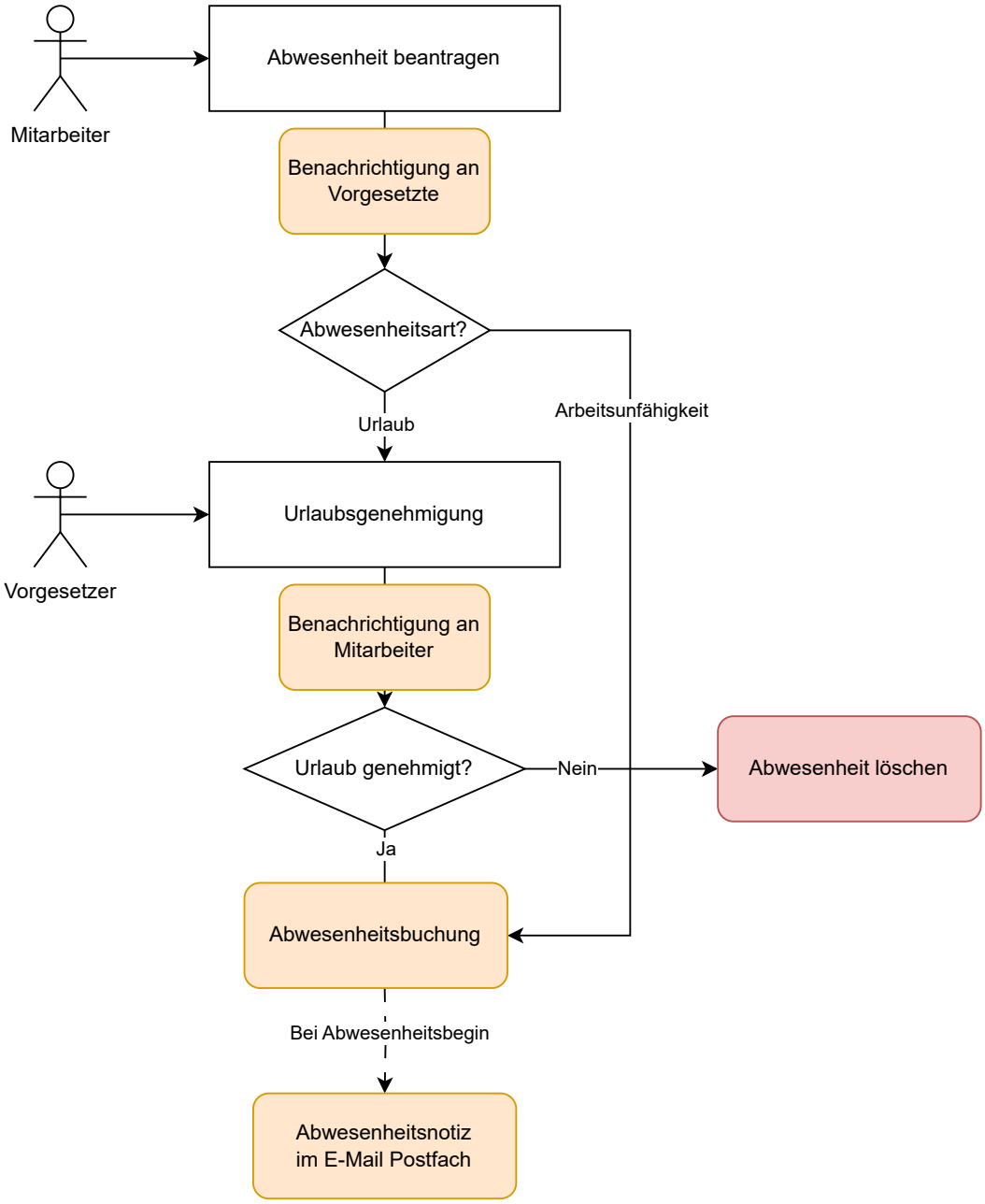


Abbildung 4: Prozess des Einreichens einer Abwesenheit

5 Implementierungsphase

Zur Umsetzung der Portalseite wurden alle Teilkomponenten der TIME+PLUS Anwendung angepasst.

5.1 Maßnahmen zur Qualitätssicherung

Zur Sicherstellung der Softwarequalität wurde während der Entwicklung der StyleCop-Analyzer verwendet, welcher mit einer firmenweiten Konfiguration, dem Entwickler Empfehlungen und Warnungen mitteilte. Damit gewährleistet werden konnte, dass komplexe Algorithmen korrekt funktionieren wurden Unit-Tests geschrieben und ausgeführt. Dies geschah direkt in der IDE des Entwicklers und beim Build-Vorgang in der Jenkins-CI.

5.2 Anpassung der Datenbankmodelle

Aus der Anforderung wurde klar, dass die existierenden Datenbankmodelle zu weiten Teilen hinreichend waren. Lediglich der Vermerk, ob ein Urlaubsantrag genehmigt wurde, existierte noch nicht. Hierzu wurde die Datenbankstruktur der Zeiterfassungstabelle über COBRA-CRM um eine Ja/Nein-Spalte 'Urlaub genehmigt' erweitert. Anschließend wurden über das internen CDO-Gen Anbindungen neue Datenbankmodelle erzeugt, damit von der Anwendung leichter auf das neue Spalte zugegriffen werden kann. Die Benutzeroberfläche im COBRA-CRM wurde angepasst um das neue Feld abzubilden.

5.3 Erweiterung des Data Access-Layers

Damit die Anwendung Zugriff auf die nötigen Zeiterfassungsdatensätze hat wurden neue Funktionen definiert und implementiert. Diese umfassten Speichern von neuen Datensätzen, sowie Abfragen und Editieren von existierenden Datensätzen. Dies geschah als Erweiterung eines existierenden 'Repository-Interfaces' und deren Implementation.

5.4 Erweiterung der Geschäftslogik im Business-Layers

Zur Abbildung der Geschäftslogik wurde eine neue Klasse 'AbwesenheitenLogic' erzeugt. Hier wurden Methoden geschrieben zur Einreichung von Abwesenheiten, zum Annehmen und Ablehnen von Urlaubsanträgen, zum Absagen einer Abwesenheit, zur Ermittlung des nächsten Arbeitstages eines Mitarbeiters und zur Ermittlung aller Abwesenheiten eines Mitarbeiters.

Zur Gruppierung der Abwesenheitstage werden diese nach Datum geordnet iteriert. Eine Abwesenheitsdatensatz wird mit Start und Ende des ersten Abwesenheitstages erzeugt. Es wird der nächste

Arbeitstag, unter Berücksichtigung der Arbeitszeiten und anstehenden Feiertagen, ermittelt. Ist der ermittelte Tag gleich dem nächsten Abwesenheitstages der selben Art, wird das Ende Abwesenheitsdatensatz auf das Ende dieses Abwesenheitstages gesetzt. Falls der ermittelte Tag kein Abwesenheitstag ist, wird der Abwesenheitsdatensatz zurück gegeben. Dies wird wiederholt, bis alle Abwesenheiten gruppiert sind. Die genaue Implementation finden sie in Anhang A.1: Klasse: AbwesenheitenLogic auf Seite i.

Zur Steuerung der Abwesenheitsnotizen im Exchange wurde eine Klasse 'GraphLogic' erzeugt. Diese bindet die nuget-Pakete Microsoft.Graph und Azure.Identity ein. Dazu wird initial im Konstruktor ein 'GraphServiceClient'-Objekt für die Ansteuerung der GraphAPI Schnittstelle konfiguriert. Eine öffentliche Methoden wurden angelegt, welche, über die oben erwähnte AbwesenheitenLogic-Klasse, die anstehenden Abwesenheiten ermittelt, und über den 'GraphServiceClient' die Abwesenheitsnotizen setzt. Eine weitere öffentliche Methode ermöglicht das setzen einer Abwesenheitsnotiz für akute Abwesenheit bei einer Arbeitsunfähigkeit.

5.5 Implementierung der Web-Endpunkte

Es wurde eine MVC-Controller Klasse AbwesenheitController erzeugt. Diese beinhaltet eine Seite zum Einreichen von Abwesenheiten für den angemeldeten Mitarbeiter und eine Seite zur Übersicht aller Abwesenheiten aller Mitarbeiter. Diese wurden mit Razor-Views dargestellt und mit CSS strukturiert. Die gewählte Kalender-Bibliothek fullcalendar wird mit JavaScript konfiguriert und Funktionalität an den Controller angebunden.

So hält der Controller weitere Endpunkte, welche JSON nutzen um über POST und GET mit dem Client zu kommunizieren. Zu Beginn ließt der Client die Anstehenden Abwesenheiten über den Endpunkt FilterAbwesenheitenForJahr. Der Controller ermittelt den angemeldeten Mitarbeiter und frägt den Business-Layer nach allen Abwesenheiten des Mitarbeiters für das Jahr und fügt diese in den Kalender ein. Siehe Anhang A.4: Abwesenheitsübersicht auf Seite vi

Ein Mitarbeiter kann Mittels Drag- & Drop-Verfahren neue Abwesenheiten in den Kalender ziehen und über den Button „Abwesenheiten abschicken“ die neuen Abwesenheiten prüfen und abschicken. Siehe Anhang A.5: Abwesenheitsantrag auf Seite vi

Über den Reiter „Abwesenheitsübersicht“ können Vorgesetze nun die Abwesenheiten in einer ähnlichen Ansicht verwalten. Ihnen ist ist hier möglich Abwesenheiten auch für andere Mitarbeiter einzureichen. Auch können sie, durch Anwählen von Urlaubsanträgen, diese genehmigen oder ablehnen. Siehe Anhang A.6: Abwesenheitsdetails auf Seite vii

6 Dokumentation

Die Dokumentation der Anwendung besteht im wesentlichen aus zwei Teilen. Der Entwicklerdokumentation und die Endnutzerdokumentation. Die Entwicklerdokumentation wurde in einer speziellen Kommentarsyntax für `C#` verfasst und wird den Entwicklern in der IDE gezeigt. Siehe Beispiel Anhang A.2: Entwicklerdokumentation auf Seite iv.

Die Endbenutzerdokumentation wurde für die intern genutzte Dokumentationsseite Confluence verfasst. Sie umfasst Benutzungsschritte für den Alltag. Siehe Anhang A.3: Endnutzerdokumentation auf Seite iv.

7 Fazit

7.1 Abnahme

Während der Entwicklungsphase wurden existierende Unit-Tests über Jenkins-CI und in der IDE ständig ausgeführt. Zur Prüfung der Korrektheit wurden für neue Algorithmen Unit-Tests verfasst und in die existierende Test-Infrastruktur integriert. Sie Anhang A.7: Unit-Tests auf Seite viii.

Die eigentliche Abnahme erfolte in Form der Inbetriebnahme mit dem Geschäftsführer. Dabei wurden alle Anwendungsfälle nacheinander durchgegangen und auf ihre Funktionsfähigkeit geprüft. Im Endeffekt konnte das System erfolgreich in Betrieb genommen werden. Es wird in einem der nächsten Meetings allen Mitarbeitern vorgestellt.

7.2 Ausblick

Bei der Abnahme durch den Geschäftsführer wurde beschlossen, dass die Formulierungen der Abwesenheitsnotiz zukünftig konfigurierbar sein soll. Hierzu sollen E-Mail Vorlagen im MSG-Format eingebunden werden.

Eidesstattliche Erklärung

Ich, Luis Scheurenbrand, versichere hiermit, dass ich meine **Dokumentation zur betrieblichen Projektarbeit** mit dem Thema

> *Portal zur Eintragung von Urlaub- und Krankheitsabwesenheiten – Weiterentwicklung von TIME+PLUS*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Stuttgart, den 04.06.2024

Luis Scheurenbrand

A Anhang

A.1 Klasse: AbwesenheitenLogic

Ermittlung der Datensätze aus Data Access-Layer werden sind ausgelassen.

```
1
2  private static DateTime? GetNextWorkday(
3      DateTime start,
4      ISet<DateTime> feiertage,
5      ISet<string> arbeitstage)
6  {
7      // maximal eine Woche in die Zukunft schauen
8      var end = start.AddDays(7);
9      for (var date = start; date < end; date = date.AddDays(1))
10     {
11         // Pruefen, ob der Wochentag fuer den Mitarbeiter gueltig ist.
12         var wochentag = new CultureInfo("de-DE")
13                             .DateTimeFormat.GetDayName(date.DayOfWeek);
14         if (!arbeitstage.Contains(wochentag))
15         {
16             continue;
17         }
18
19         // Pruefen, ob der Tag ein Feiertag ist.
20         if (feiertage.Contains(date))
21         {
22             continue;
23         }
24
25         return date;
26     }
27
28     return null;
29 }
30
31 private static decimal GetStunden(Zeiterfassung zA, AbwesenheitenArt art) =>
32     art switch
33     {
34         AbwesenheitenArt.Arbeitsunfaehigkeit => zA.Arbeitsunfaehig!.Value,
35         AbwesenheitenArt.Urlaub => zA.Urlaub!.Value,
36         AbwesenheitenArt.Ausbildung => zA.Ausbildung!.Value,
37         AbwesenheitenArt.Feiertag => 0m,
38         _ => throw new NotImplementedException(),
39     };
40
41 private static string GetColor(Zeiterfassung zA, AbwesenheitenArt art)
42 {
43     return art switch
44     {
```

```
45          AbwesenheitenArt.Arbeitsunfaehigkeit => zA.AU ? ArbeitsunfaehigMitAuColor :
                ArbeitsunfaehigOhneAuColor,
46          AbwesenheitenArt.Urlaub => zA.UrlaubGenehmigt ? UrlaubGenehmigtColor : UrlaubColor,
47          AbwesenheitenArt.Ausbildung => AusbildungColor,
48          AbwesenheitenArt.Feiertag => FeiertagColor,
49          _ => throw new NotImplementedException(),
50      };
51  }
52
53  private static bool IsBearbeitbar(Zeiterfassung zA) => zA.Datum!.Value > DateTime.Now;
54
55  private static IEnumerable<AbwesenheitenContainer> GroupAbwesenheiten(
56      IEnumerable<Zeiterfassung> abwesenheiten,
57      AbwesenheitenArt art,
58      IEnumerable<Feiertage> allFeiertage,
59      IDictionary<int, ISet<string>> arbeitstageByMitarbeiterId)
60  {
61      using var iterator = abwesenheiten.Where(zA => GetStunden(zA, art) > 0m).GetEnumerator();
62      if (!iterator.MoveNext())
63      {
64          yield break;
65      }
66
67      var feiertage = allFeiertage.Select(f => f.Datum!.Value).ToHashSet();
68
69      var current = iterator.Current!;
70      var abwesenheit = new AbwesenheitenContainer
71      {
72          MitarbeiterId = current.SuperId!.Value,
73          Start = current.Datum!.Value,
74          Ende = current.Datum!.Value.AddDays(1),
75          Stunden = GetStunden(current, art),
76          Art = art,
77          Bemerkungen = current.Bemerkungen,
78          Color = GetColor(current, art),
79          Bearbeitbar = IsBearbeitbar(current)
80      };
81
82      if (art == AbwesenheitenArt.Urlaub)
83      {
84          abwesenheit.Genehmigt = current.UrlaubGenehmigt;
85      }
86
87      if (art == AbwesenheitenArt.Arbeitsunfaehigkeit)
88      {
89          abwesenheit.Au = current.AU;
90      }
91
92      while (iterator.MoveNext())
93      {
```

```
94          current = iterator.Current!;
95          var canBeGrouped =
96              current.SuperId == abwesenheit.MitarbeiterId &&
97              GetStunden(current, art) == abwesenheit.Stunden &&
98              current.Bemerkungen == abwesenheit.Bemerkungen &&
99              IsBearbeitbar(current) == abwesenheit.Bearbeitbar;
100
101         if (art == AbwesenheitenArt.Urlaub)
102         {
103             canBeGrouped &= current.UrlaubGenehmigt == abwesenheit.Genehmigt;
104         }
105
106         if (art == AbwesenheitenArt.Arbeitsunfaehigkeit)
107         {
108             canBeGrouped &= abwesenheit.Au == current.AU;
109         }
110
111         // Naechster Arbeitstag ermitteln
112         var nextArbeitstagDate = GetNextWorkday(
113             abwesenheit.Ende,
114             feiertage,
115             arbeitstageByMitarbeiterId[abwesenheit.MitarbeiterId]);
116         if (canBeGrouped && nextArbeitstagDate != null && current.Datum == nextArbeitstagDate)
117         {
118             abwesenheit.Ende = current.Datum!.Value.AddDays(1);
119         }
120         else
121         {
122             yield return abwesenheit;
123             abwesenheit = new AbwesenheitenContainer
124             {
125                 MitarbeiterId = current.SuperId!.Value,
126                 Start = current.Datum!.Value,
127                 Ende = current.Datum!.Value.AddDays(1),
128                 Stunden = GetStunden(current, art),
129                 Art = art,
130                 Bemerkungen = current.Bemerkungen,
131                 Color = GetColor(current, art),
132                 Bearbeitbar = IsBearbeitbar(current)
133             };
134
135             if (art == AbwesenheitenArt.Urlaub)
136             {
137                 abwesenheit.Genehmigt = current.UrlaubGenehmigt;
138             }
139
140             if (art == AbwesenheitenArt.Arbeitsunfaehigkeit)
141             {
142                 abwesenheit.Au = current.AU;
143             }
```

```
144             }
145         }
146
147         yield return abwesenheit;
148     }
```

Listing 1: Klasse: AbwesenheitenLogic

A.2 Entwicklerdokumentation

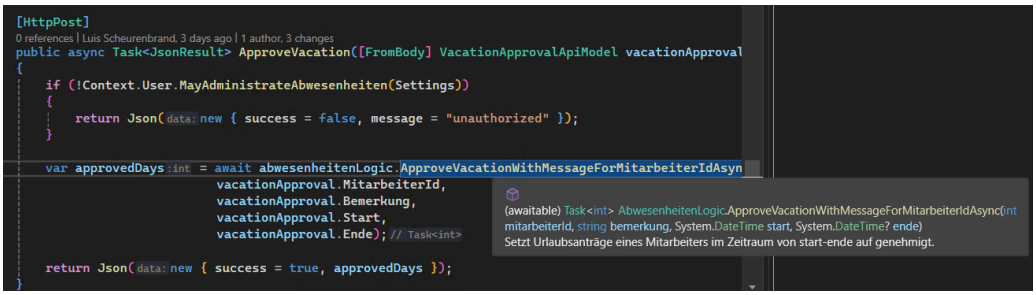


Abbildung 5: Entwicklerdokumentation

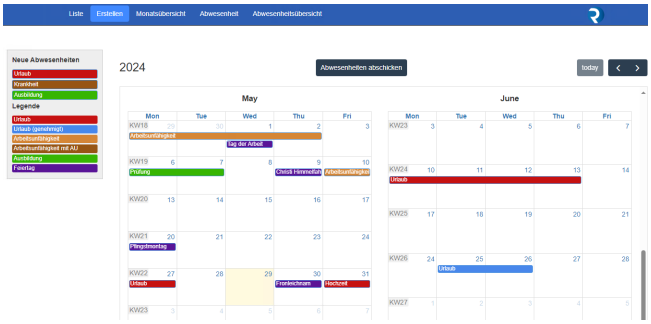
A.3 Endnutzerdokumentation

Abwesenheitsmeldungen über TIME+PLUS Portal

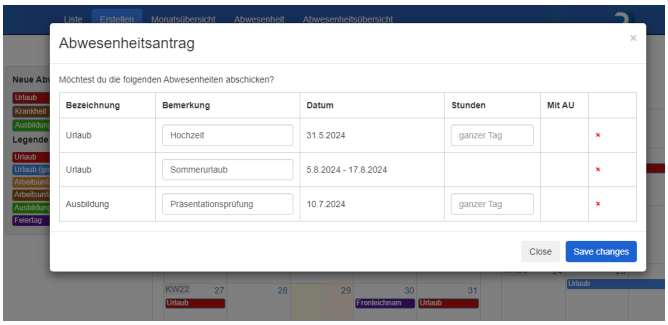
Krankeitsmeldungen und Urlaubsanträge können zukünftig über ein neu entwickeltes Portal über die Webseite https://ticket.ruthardt.eu/ eingereicht werden.

Falls ihr noch keinen Zugang habt, oder euer Kennwort vergessen habt, könnt ihr euer Passwort über den Link https://ticket.ruthardt.eu/Authorization/PasswortVergessen zurücksetzen.

Unter dem Reiter "Abwesenheiten" findet ihr einen Kalender mit euren bisherigen Abwesenheiten. Auf der linken Seite findet ihr eine Liste an neuen Abwesenheiten.



Ihr könnt per Drag & Drop eine neue Abwesenheit in den Kalender ziehen. Auf Mobilgeräten müsst ihr kurz z.B. Urlaub halten. Ihr könnt den Termin dann länger ziehen.
Wenn ihr eure Urlaube & Krankheiten im Kalender habt, könnt ihr über den Button "Abwesenheiten abschicken" in einer Übersicht die neuen Abwesenheiten betrachten.



Abwesenheiten, welche nur einen Tag gehen, können mit einem Stundenanteil bearbeitet werden. Falls ihr keine Stunden angebt, wird der volle Tag eingetragen.

Bei Krankeiten könnt ihr noch einen Haken setzen, ob eine AU (Arbeitsunfähigkeitsbescheinigung) vorliegt.

Anschließend könnt ihr mit "Save changes" die Abwesenheiten einreichen.

Eure Urlaube müssen dann von Frank oder Daniel genehmigt werden.

Anleitung für Vorgesetzte

Ihr findet im Portal einen Reiter "Abwesenheitenübersicht".

Hier findet ihr einen Kalender mit allen Abwesenheiten für alle Mitarbeiter.

Abbildung 6: Endnutzerdokumentation

A.4 Abwesenheitsübersicht

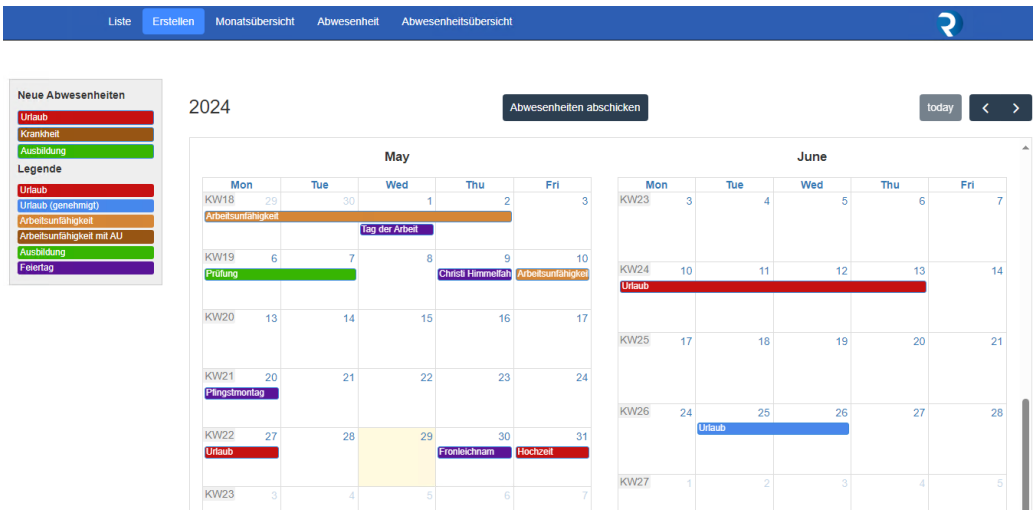


Abbildung 7: Abwesenheitsübersicht

A.5 Abwesenheitsantrag

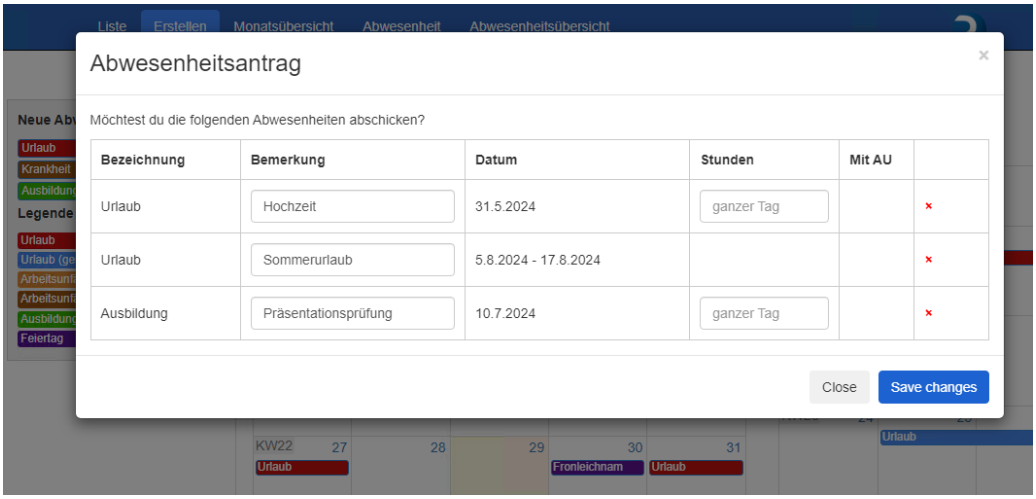


Abbildung 8: Abwesenheitsantrag

A.6 Abwesenheitsdetails

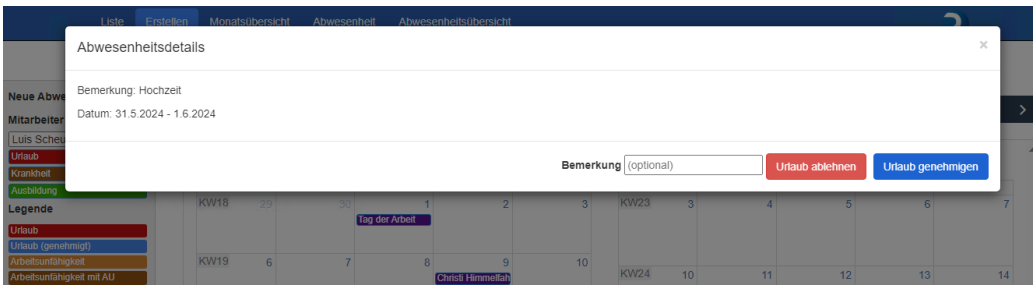


Abbildung 9: Abwesenheitsdetails für Vorgesetzen

A.7 Unit-Tests

```
1  namespace Ruthardt.TimePlus.UnitTest;
2
3  using System;
4  using System.Collections.Generic;
5
6  using Microsoft.VisualStudio.TestTools.UnitTesting;
7
8  using Ruthardt.CobraDomainModel;
9  using Ruthardt.TimePlus.BusinessLogic.Abwesenheiten;
10 using Ruthardt.TimePlus.Model.TimeTracker;
11
12 [TestClass]
13 public class AbwesenheitenTests
14 {
15   [TestMethod]
16   public void TestGetNextWorkday()
17   {
18     var arbeitstage = new HashSet<string>(["Montag", "Dienstag", "Donnerstag", "Freitag"]);
19     var feiertage = new HashSet<DateTime>(
20       [
21         new DateTime(2024, 05, 01),
22         new DateTime(2024, 05, 09),
23         new DateTime(2024, 05, 20),
24         new DateTime(2024, 05, 30),
25       ]);
26     // Mittwoch kein Arbeitstag, Donnerstag Feiertag
27     Assert.AreEqual(
28       new DateTime(2024, 05, 10),
29       AbwesenheitenLogic.GetNextWorkday(new DateTime(2024, 5, 08), feiertage, arbeitstage));
30     // Samstag & Sonntag keine Arbeitstage
31     Assert.AreEqual(
32       new DateTime(2024, 05, 13),
33       AbwesenheitenLogic.GetNextWorkday(new DateTime(2024, 5, 11), feiertage, arbeitstage));
34     // Samstag & Sonntag keine Arbeitstage, Montag Feiertag
35     Assert.AreEqual(
36       new DateTime(2024, 05, 21),
37       AbwesenheitenLogic.GetNextWorkday(new DateTime(2024, 5, 18), feiertage, arbeitstage));
38   }
39
40   [TestMethod]
41   public void GroupAbwesenheiten()
42   {
43     var arbeitstage = new HashSet<string>(["Montag", "Dienstag", "Donnerstag", "Freitag"]);
44     var feiertage = new HashSet<Feiertage>(
45     [
46       new Feiertage("Test") { Datum = new DateTime(2024, 05, 01) },
47       new Feiertage("Test") { Datum = new DateTime(2024, 05, 09) },
48       new Feiertage("Test") { Datum = new DateTime(2024, 05, 20) },
```

```
49        new Feiertage("Test") { Datum = new DateTime(2024, 05, 30) },
50      ]);
51      Zeiterfassung[] zA = [
52        new Zeiterfassung("Test")
53        {
54          SuperId = 1,
55          Datum = new DateTime(2024, 05, 02),
56          Urlaub = 8m,
57          UrlaubGenehmigt = true,
58        },
59        new Zeiterfassung("Test")
60        {
61          SuperId = 1,
62          Datum = new DateTime(2024, 05, 03),
63          Urlaub = 8m,
64          UrlaubGenehmigt = true,
65        },
66        new Zeiterfassung("Test")
67        {
68          SuperId = 1,
69          Datum = new DateTime(2024, 05, 06),
70          Urlaub = 8m,
71          UrlaubGenehmigt = true,
72        },
73        new Zeiterfassung("Test")
74        {
75          SuperId = 1,
76          Datum = new DateTime(2024, 05, 07),
77          Urlaub = 8m,
78          UrlaubGenehmigt = false,
79        },
80        new Zeiterfassung("Test")
81        {
82          SuperId = 1,
83          Datum = new DateTime(2024, 05, 10),
84          Urlaub = 8m,
85          UrlaubGenehmigt = false,
86        },
87      ];
88      var iter = AbwesenheitenLogic.GroupAbwesenheiten(
89        zA,
90        AbwesenheitenArt.Urlaub,
91        feiertage,
92        new Dictionary<int, ISet<string>>() { { 1, arbeitstage } }
93      ).GetEnumerator();
94      Assert.IsTrue(iter.MoveNext());
95      Assert.AreEqual(new DateTime(2024, 05, 02), iter.Current.Start);
96      Assert.AreEqual(new DateTime(2024, 05, 07), iter.Current.Ende);
97      Assert.IsTrue(iter.Current.Genehmigt);
98      Assert.IsTrue(iter.MoveNext());
```

```
99      Assert.AreEqual(new DateTime(2024, 05, 07), iter.Current.Start);
100     Assert.AreEqual(new DateTime(2024, 05, 11), iter.Current.Ende);
101     Assert.IsFalse(iter.Current.Genehmigt);
102   }
103 }
```

Listing 2: Klasse: AbwesenheitenTests