

Vorwort

Die Folgende Dokumentation ist im Prüfungsrahmen der IHK Schwaben verfasst:

... "Die Ausführung der Projektarbeit wird mit praxisbezogenen Unterlagen dokumentiert. Durch die Projektarbeit und deren Dokumentation soll der Prüfling belegen, daß er Arbeitsabläufe und Teilaufgaben zielorientiert unter Beachtung wirtschaftlicher, technischer, organisatorischer und zeitlicher Vorgaben selbstständig planen und kundengerecht umsetzen sowie Dokumentationen kundengerecht anfertigen, zusammenstellen und modifizieren kann."

(Auszug der Praktischen Abschlussprüfung, s. [Literaturverzeichnis](#)).

Die Rahmenbedingungen der Dokumentation wurden von der Kammer wie folgt vorgegeben:

... "Umfang der Dokumentation ohne Anlagen höchstens 10 Seiten, Es gibt keine Vorgaben zu Formatierungen (bspw. Zeilenabstand, Schriftgröße, etc.), Nicht selbstständig erstellte Dokumentationsteile deutlich kennzeichnen, Kennzeichnung und Begründung von Änderungen gegenüber dem Projektantrag "

(Auszug der Projektverordnung, s. [Literaturverzeichnis](#)).

Der Dokumentationsumfang der genannten 10 Seiten erstreckt sich von Seite [9](#) bis [18](#).

Sprachgebrauch

Zur besseren Lesbarkeit wird in der vorliegenden Arbeit auf die gleichzeitige Verwendung männlicher und weiblicher Sprachformen verzichtet. Es wird das generische Maskulinum verwendet, wobei beide Geschlechter gleichermaßen gemeint sind.

Die in der Dokumentation gewählte männliche Form bezieht sich immer zugleich auf Männer und Frauen.

Inhaltsverzeichnis

Vorwort.....	1
Sprachgebrauch	1
Inhaltsverzeichnis	2
Tabellenverzeichnis	5
Glossar.....	6
Technisches Abkürzungsverzeichnis	8
1. Projektbeschreibung	9
1.1 Projektbezeichnung.....	9
1.2 Projektumfeld.....	9
1.3 Projektziel.....	9
1.4 Projektbegründung.....	9
1.5 Projektschnittstellen	9
1.6 Projektabgrenzung	10
2 Projektplanung	10
2.1 Projektphasen.....	10
2.2 Abweichungen vom Projektantrag.....	10
2.3 Ressourcenplanung.....	10
2.4 Entwicklungsprozess.....	10
3 Analysephase	11
3.1 Ist-Analyse	11
3.2 Wirtschaftlichkeitsanalyse	11
3.2.1 „Make or buy“ Entscheidung	11
3.2.2 Projektkosten.....	11
3.2.3 Amortisationsdauer	11
3.3 Anwendungsfälle	12
3.4 Lastenheft	12
4 Entwurfsphase.....	12
4.1 Zielplattform	12
4.2 Server	12
4.2.1 Spezifikationen.....	12

4.3	Architekturdesign	13
4.4	Benutzeroberfläche	13
4.5	Datenmodell.....	13
4.6	E-Mail	14
4.7	Geschäftslogik	14
5	<i>Implementierungsphase</i>	14
5.1	Implementierung des Datenmodells.....	14
5.1.1	Exkurs: Migration	14
5.1.2	Exkurs: Eloquent	14
5.2	Implementierung der Benutzeroberfläche	15
5.2.1	Exkurs: Blade	15
5.3	Implementierung der Routen.....	15
5.3.1	Exkurs: Routing	15
5.4	Implementierung der Controller	15
5.5	Implementierung der E-Mail Notifications	16
5.6	Testen der Anwendung.....	16
6	<i>Abnahme- und Deployment</i>	17
6.1	Code-Review.....	17
6.2	Deployment	17
7	<i>Dokumentation</i>	17
8	<i>Fazit</i>	18
8.1	Soll-/ Ist-Vergleich.....	18
8.2	Lessons Learned	18
8.3	Ausblick	18
	<i>Literaturverzeichnis</i>	19
	<i>Quellen</i>	19
	<i>Anhang</i>	20
A.1	Detaillierte Zeitplanung	20
A.2	Ressourcenplan.....	21
A.3	Ist-Analyse - Aktivitätsdiagramm.....	22
A.4	Kalkulatorische Aufstellung.....	23

A.5	Zeiteinsparung	23
A.6	Anwendungsfälle – Use-Case-Diagramm	24
A.7	Auszug Lastenheft.....	25
A.8	Mockup - Formular.....	26
A.9	Mockup - Dashboard	26
A.10	Entity Relationship Modell.....	27
A.11	Request Eloquent Model - Codeauszug	28
A.12	Request Migration - Codeauszug	29
A.13	Request - Klassendiagramm	30
A.14	Composer Spezifikation – Codeauszug	31
A.15	Composer Spezifikation – Erklärung	32
A.16	Routen – Codeauszug	33
A.17	Blade – Codeauszug.....	34
A.18	Screenshots der Applikation	35
A.18.1	Login-Maske	35
A.18.2	Dashboard.....	35
A.18.3	Erstellungs-Maske.....	36
A.18.4	Antrags Administration.....	37
A.18.5	Antrags Übersicht.....	38
A.19	Ist-Analyse – Word Vorlage	39
A.20	Antrags Administration – Codeauszug	40
A.21	E-Mail Notification – Sequenzdiagramm	42
A.22	Mailing - Codeauszug.....	43
A.23	Enlightn Test – CLI Auszug	44
A.24	Soll-/Ist-Vergleich	44
A.25	Entwicklerdokumentation – Auszug	45

Tabellenverzeichnis

Tabelle I – Grober Zeitplan.....	10
Tabelle II - Detaillierte Aufwandsschätzung	20
Tabelle III - Ressourcenplan	21
Tabelle IV - Kalkulatorische Aufstellung der Entwicklungskosten.....	23
Tabelle V - Zeiteinsparungskalkulation	23
Tabelle VI - Composer Spezifikation im Vergleich: Basis - Erweitert	32
Tabelle VII - Soll-/Ist Vergleich	44

Glossar

Alle im Glossar zu findenden Fachbegriffe sind in der nachfolgenden Dokumentation *kursiv-unterstrichen* formatiert.

Alle im Glossar aufgeführten Erklärungen beziehen sich zusammengefasst aus den genannten [Quellen](#).

ULLA – Projektname (Urlaubsverwaltung & Antragserstellung)

Mail2Many – Das von Atrivio entwickelte Newsletter-System mail2many ermöglicht die Kommunikation zwischen Herstellern, ihren Händlern und Kunden.

Geneva – Ein von Atrivio entwickeltes Content-Management-System.

CMS - Ein Content-Management-System (kurz CMS, deutsch Inhaltsverwaltungssystem) ist eine Software zur gemeinschaftlichen Erstellung, Bearbeitung, Organisation und Darstellung digitaler Inhalte (Content) zumeist zur Verwendung in Webseiten, aber auch in anderen Medienformen.

Compliance - Compliance ist die betriebswirtschaftliche und rechtswissenschaftliche Umschreibung für die Regeltreue von Unternehmen, also die Einhaltung von Gesetzen, Richtlinien und freiwilligen Kodizes.

Laravel - Laravel ist ein freies PHP-Webframework, das dem [MVC](#)-Muster folgt. Es wurde 2011 von Taylor Otwell initiiert.

Blade - Blade ist die standardmäßig mit Laravel mitgelieferte Template Engine. Mit Blade lassen sich Views in einer Laravel Anwendung implementieren. Template Engines wie Blade ermöglichen die Wiederverwendbarkeit von Views und erhöhen damit die Produktivität bei der Entwicklung sowie die Wartbarkeit einer Anwendung. View-Teile die sehr ähnlich oder identisch sind, können als einzelne Blöcke definiert und innerhalb anderer Views wiederverwendet werden. Beim richtigen Einsatz von Blade entsteht gut strukturierter und damit gut lesbarer Code, der neuen Entwicklern den Einstieg in bestehende Projekte deutlich erleichtern kann.

Reverse Engineering - Das Reverse Engineering von Code erlaubt es Programmierern, Entwicklungs- und Produktionsprozesse einer Software umzukehren und so einen Blick hinter die Kulissen eines Programms zu erhalten.

Composer - Composer ist ein anwendungsorientierter Paketmanager für die Skriptsprache PHP. Composer wird über die Kommandozeile ausgeführt und installiert Abhängigkeiten eines PHP-Programmes. Verfügbare PHP-Anwendungen können über die Plattform Packagist gesucht werden.

composer.json - Diese Datei beschreibt die Abhängigkeiten zu externen Paketen, Ressourcen und Metadaten des Projekts. Somit kann Composer technisch gesehen das Projekt systemunabhängig installieren.

Packagist - Packagist ist das Haupt-Repository von Composer. Es sammelt öffentliche PHP-Pakete, die mit Composer installiert werden können.

SCSS – SASS bzw. SCSS ist eine Stylesheet-Sprache, die als CSS-Präprozessor mit Variablen, Schleifen und vielen anderen Funktionen, die Cascading Style Sheets nicht mitbringen, die Erstellung von CSS vereinfacht und die Pflege großer Stylesheets erleichtert.

Node.js - Node.js ist eine plattformübergreifende Open-Source-JavaScript-Laufzeitumgebung, die JavaScript-Code außerhalb eines Webbrowsers ausführen kann.

Bootstrap - Bootstrap ist ein freies Frontend-CSS-Framework. Es enthält auf HTML und CSS basierende Gestaltungsvorlagen für Typografie, Formulare, Buttons, Tabellen, Grid-Systeme, Navigations- und andere Oberflächengestaltungselemente sowie zusätzliche, optionale JavaScript-Erweiterungen.

Markdown - Markdown ist eine vereinfachte Auszeichnungssprache ähnlich wie HTML oder XML.

Seeding - Das Seeding einer Datenbank ist ein Prozess, bei dem eine Datenbank bei der Installation mit einem anfänglichen Satz von Daten versorgt wird. Dies ist ein automatisierter Prozess, der bei der Ersteinrichtung einer Anwendung ausgeführt wird. Bei den Daten kann es sich um Dummy-Daten oder um notwendige Daten wie ein anfängliches Administratorkonto handeln.

Technisches Abkürzungsverzeichnis

Alle im technischen Abkürzungsverzeichnis zu findenden Fachbegriffen sind in der nachfolgenden Dokumentation unterstrichen formatiert.

Alle im Abkürzungsverzeichnis aufgeführten Erklärungen beziehen sich zusammengefasst aus den genannten Quellen.

PHP Hypertext Preprocessor

MVC Model View Controller

HTML Hypertext Markup Language

CSS Cascading Style Sheet

ERM Entity Relation Modell

ORM Object relation mapping

REPL Read Eval Print Loop (Shell)

SQL Structured Query Language

UML Unified Modeling Language

1. Projektbeschreibung

1.1 Projektbezeichnung

Firmeninternes Management Interface zur Urlaubsverwaltung & Antragserstellung als Webapplikation zum Zwecke der Digitalisierung. Kurzform ULLA (Akronymisiert aus Urlaubsverwaltung & Antragserstellung)

1.2 Projektumfeld

Auftraggeber ist die Atrivio GmbH (im Folgenden durch Atrivio abgekürzt). Das Projekt wurde in den Räumen der Atrivio in 87437 Kempten durchgeführt. Seit über 18 Jahren begleitet die Atrivio Unternehmen bei der Umsetzung kundenspezifischer, digitaler Lösungen. Dabei werden zum einen Dienstleistungen wie die Umsetzung Webbasierter [CMS](#) Lösungen, E-Commerce-Plattformen bis hin zur Cloud Infrastruktur angeboten, aber auch eigene Produkte wie [Mail2Many](#) oder [Genera](#) betriebseigen entwickelt und vertrieben.

1.3 Projektziel

Der gesamte Prozess eines Antrages von der Erstellung bis zur Archivierung soll zukünftig webbasiert – digitalisiert werden. Primär sollen die Abteilungsleiter durch den Digitalisierungsprozess von den bürokratischen und organisatorischen Mehraufwänden des bisherigen Prozesses entbunden werden. Außerdem werden durch die digitale Archivierung in Zukunft Druck- und Materialkosten reduziert. Durch das neue System wird eine strikte [Compliance](#) und Ordnung in den Ablauf der Antragserstellung eingebracht.

Durch die Umsetzung soll eine prozessorientierte, schlanke und zukünftig erweiterbare, intern gehostete Web-Anwendung mit Anbindung einer Datenbank und grafischer Benutzeroberfläche entstehen.

1.4 Projektbegründung

Derzeit steht allen Mitarbeitern eine Word Vorlage für die Antragserstellung zur Verfügung. Diese Vorlage ist zentral und für alle Mitarbeiter erreichbar gespeichert. Soll ein neuer Antrag eingereicht werden wird die Vorlage aufgerufen, gefüllt und ausgedruckt. Der unterschriebene Ausdruck wandert dann durch die Abteilung und wird schlussendlich in einem der Verwaltungsordner archiviert. Durch den bisherigen Prozess entstehen mehrere wirtschaftliche Makel. Beispielsweise wird die Prozessdauer durch die händische Weitergabe des Antrags unnötig in die Länge gezogen. Außerdem werden, wie in [Punkt 1.3](#) erwähnt, vermehrt Unternehmensressourcen wie Büromaterialien (Druckermittel, Papier, Ordner) oder Stellplätze (Regale) verbraucht.

Einen organisatorischen Makel stellt die fehlende Zentralisierung der Anträge dar. Momentan besteht keine Möglichkeit, Anträge Abteilungsübergreifend zu vergleichen u. o. abzustimmen.

Mit der Umsetzung von [ULLA](#) werden die organisatorischen und wirtschaftlichen Makel beseitigt und eine zentrale, zukunftssichere, kosten- und ressourcensparende Anlaufstelle geschaffen.

1.5 Projektschnittstellen

Der Endbenutzer kann die Webapplikation in jedem aktuellen Browser und auf einer Vielzahl von Geräten wie beispielsweise PC's, Notebooks über Tablets bis hin zu Smartphones im internen Firmennetz aufrufen. Dabei ist die Applikation im ersten Schritt zwar responsiv fähig, aber für die Ansicht auf Desktopendgeräten optimiert. Um an die Applikationsdaten zu gelangen, muss die Anwendung mit einer MySQL-Datenbank kommunizieren und Werte abfragen können. Diese befindet sich im internen Firmennetz und kann direkt angesprochen werden.

Weitere Schnittstellen sind im ersten Entwicklungsschritt nicht geplant.

1.6 Projektabgrenzung

ULLA ist eine von Grund auf eigenständig, im Rahmen der praktischen Abschlussprüfung der IHK Schwaben bzw. der Beauftragung durch Atrivio programmierte Applikation ohne Abgrenzung oder Abhängigkeiten von bereits bestehenden Applikationen oder Applikationen Drittanbietern.

2 Projektplanung

2.1 Projektphasen

Das Projekt wurde innerhalb den, von der IHK Schwaben, vorgegebenen 70 Stunden durchgeführt. Eine grobe Zeitplanung der Hauptphasen lässt sich Tabelle I entnehmen. Die im Projektantrag aufgeführte detaillierte Zeitplanung findet sich in [Anhang A.1](#).

Phase	Geplante Zeit
Analyse	3.5h
Entwurf	8.0h
Entwicklung und Implementierung	42.5h
Abnahme und Deployment	3.0h
Dokumentation	13.0h
Gesamt:	70.0h

Tabelle I – Grober Zeitplan

2.2 Abweichungen vom Projektantrag

In der Analysephase wurde zur Darstellung des Sachverhalts kein Use-Case- sondern ein Aktivitätsdiagrams verwendet. In der Entwurfsphase wurden die Handlungsschritte in einem Use-Case- anstatt eines Aktivitätsdiagrams dargestellt.

2.3 Ressourcenplanung

Die benötigten Ressourcen sowie die verwendete Hard- und Software können im [Anhang A.2](#) im Ressourcenplan eingesehen werden.

2.4 Entwicklungsprozess

Als letzter Schritt vor dem tatsächlichen Beginn des Projektes musste durch den Autor ein geeigneter Entwicklungsprozess gewählt werden. Durch diesen wird die Herangehensweise und Entwicklung bei der Umsetzung des Projektes definiert.

Grundsätzlich kann die Umsetzung im Wasserfallmodell erfolgen, da die einzelnen Projektabschnitte, durch die gegebenen Rahmenbedingungen klar definiert und konzeptioniert sind. Somit kann mit dem gewählten Umsetzungsmodell die Entwicklung präzise und zeiteffektiv durchgeführt werden. Im Bereich der Oberfläche und Prozessabläufe soll es jedoch durch regelmäßige Rücksprache mit dem Fachbereich einen agilen Prozess geben. Ebenso wird in Bezug auf die fachliche Logik und Prüfung der verarbeiteten Daten durch Plausibilitäten in iterativen Zyklen nach Erreichen von Projektmeilensteinen ein Anwendertest durch den Fachbereich schon in die Implementierungsphase integriert um den generellen Aufwand der Abnahmephase zu minimieren.

Auf Modultest (auch von englisch unit test als Unittest oder als Komponententest bekannt) wurde aus zeitlichen Gründen verzichtet, allerdings fanden in der Entwicklungsphase und Abnahme des Projektes Codereviews mit der Ausbildungsleitung statt.

3 Analysephase

3.1 Ist-Analyse

Wie bereits unter [1.4 \(Projektbegründung\)](#) beschrieben, stellt der aktuelle Ablauf der Beantragung einen wirtschaftlich und organisatorisch suboptimalen Prozess dar. Der genaue Ablauf des aktuellen Prozesses, die Akteure und Problemstellungen sollen in der folgenden Ist-Analyse genau dargestellt werden.

Am Prozess der Antragserstellung sind drei Akteure beteiligt. Der Prozess startet bisher mit dem Aufruf der Word Vorlage durch einen Mitarbeiter (Benutzergruppe: Mitarbeiter, Akteur 1). Auf das Öffnen der Vorlage folgt die Eingabe der Antragsparameter und abspeichern der Datei. Ein Ausdruck jener Datei wird anschließend unterschrieben dem Vorgesetzten (Benutzergruppe: Vorgesetzter, Akteur 2) der Abteilung vorgelegt. Entscheidet der Vorgesetzte den Antrag nicht freizugeben, nimmt der Prozess ein vorzeitiges Ende. Wird der Antrag freigegeben unterschreibt der Vorgesetzte den Antrag ebenfalls und informiert den Antragsteller bzw. Mitarbeiter mündlich über die Freigabe. Der Unterschriebene Antrag wird dann an die Verwaltung übergeben. Ein Verwaltungsmitarbeiter vermerkt die genehmigten Urlaubstage in der Personalverwaltung bzw. dem dafür vorgesehenen Personalverwaltungsprogramm und archiviert den Antrag. Außerdem wird dem Antragsteller bzw. Mitarbeiter eine Kopie zur schriftlichen Bestätigung und Absicherung vorgelegt.

Im [Anhang A.3](#) ist ein Aktivitätsdiagramm abgebildet, in dem dieser Prozess dargestellt wird. In [Anhang A.19](#) ist die bisherige Wordvorlage zu sehen.

3.2 Wirtschaftlichkeitsanalyse

3.2.1 „Make or buy“ Entscheidung

Im Vorfeld bestand bereits großes Interesse seitens Atrivio den derzeitig zweckmäßigen Workflow digital zu verbessern. Bei Lösungen von Drittanbietern ergaben sich dabei kostspielige Abo Modelle oder fehlende Schnittstellen u. o. Erweiterungsmöglichkeiten als Ausschlusskriterien. Als Software-Unternehmen bzw. IT-Dienstleister lag die strategische Entscheidung der Eigenentwicklung, um die Abhängigkeiten und zusätzliche Kosten von Drittanbietern bzw. Dienstleistern zu vermeiden, somit nahe.

3.2.2 Projektkosten

In der Kalkulation wurden alle anfallenden Entwicklungskosten von ULLA im Rahmen der Umsetzung von 70 Stunden erfasst. Betrachtet wurden dabei neben den personellen Kosten auch alle Software-, Hardware- und sonstige Kosten (Arbeitsplatz, Klo, ect.). Genaue personelle Kosten dürfen von der Personalabteilung nicht herausgegeben werden, daher wurden die vom Controlling verwendeten Kalkulatorischen Stundensätze verwendet. Die genaue Kalkulatorische Aufstellung kann in Anhang A.4 eingesehen werden.

3.2.3 Amortisationsdauer

Durch die deutliche Zeitersparnis und die zu erwartende Nutzungsdauer von mehreren Jahren ist darauf zu schließen, dass das Projekt auch unter wirtschaftlichen Gesichtspunkten sinnvoll ist.

Die folgende Amortisationsrechnung ist auf Basis der in [Anhang A.5](#) Zeiteinsparungsrechnung und in [Anhang A.4](#) entstandenen Entwicklungskosten zu verstehen.

$$\text{Amortisationsdauer} = \frac{\text{Entwicklungskosten}}{\text{Jährlicher Rückfluss}} = \frac{5.720,00\text{€}}{4.387,80\text{€}} = 1,3 \text{ Jahre}$$

3.3 Anwendungsfälle

Die Anwendungsfälle von ULLA wurden zur genaueren Darstellung in einem Anwendungsfalldiagramm zusammengefasst.

Das Anwendungsfalldiagramm (Use-Case-Diagramm) stellt das Zusammenspiel der Anwendungsfälle eines Systems untereinander und den Akteuren dar¹. Obwohl das Konzept der Anwendungsfälle vollkommen unabhängig von der objektorientierten Modellierung ist, besitzt es einen festen Platz in den meisten Vorgehensmodellen für die objektorientierte Softwareentwicklung bzw. der UML² und wurde somit als Grundlage für ein internes Anforderungsdokument nach dem Muster des Lastenhefts erstellt. Die Rollenverteilung des späteren Usermanagements wird durch die Vererbungshierarchie im Diagramm ebenfalls erkennbar. Das Anwendungsfalldiagramm ist im [Anhang A.6](#) zu finden.

3.4 Lastenheft

Am Ende der Analysephase wurde wie in der Konzeption und unter [Punkt 3.3](#) bereits angekündigt ein Lastenheft zum Projektanstoß erstellt. Das Lastenheft dient als zentrale Anlaufstelle, um alle zu berücksichtigen Anforderungen gegliedert zusammenzufassen. Ein Auszug aus besagtem Dokument befindet sich im [Anhang A.7](#).

4 Entwurfsphase

4.1 Zielplattform

Wie unter [Punkt 1.1](#) bereits beschrieben soll [ULLA](#) als intern gehostete Webapplikation (im Folgenden als WebApp abgekürzt) entwickelt werden. Das Deployment als WebApp bringt dabei mehrere Vorteile mit sich. Durch die Webanbindung stehen allen Usern permanent dieselbe Version und somit auch Funktionen und Konfigurationen zur Verfügung. WebApp's sind geräteübergreifend (man spricht auch von „Cross Device“) optimiert und somit nicht an Betriebssysteme oder Bildschirmgrößen gebunden. Durch die entfallenden Abhängigkeiten durch Betriebssystemversionen (Abwärtskompatibilität) und die damit erleichterte Entwicklung sowie Fehlersuche werden zudem Entwicklungskosten gespart. Alle Projektbezogenen Daten werden auf einer – für das Projekt dediziert erstellten – MySQL Datenbank gespeichert.

Als primäres Framework der Anwendung wird das Open-Source [PHP](#)-Framework [Laravel](#) eingesetzt. Die Auswahl ergibt sich zum einen durch die von [Atrivio](#) vorgeschriebenen Architekturregeln, zum anderen eignet sich die Framework Architektur durch die Bindung an das [MVC](#) Musters und [Blade](#) Templating für die vorgesehenen Anforderungen.

4.2 Server

Anders als die Quellcodeumwandlung einer compilerbezogenen Programmiersprache in Maschinencode ist bei WebApps ein Interpreter in Form eines Webservers gefragt. Von Ihm wird durch die Verwendung von [Laravel](#) PHP-Quellcode bei jedem Seitenaufruf eingelesen, analysiert und ausgeführt.

4.2.1 Spezifikationen

Wegen der höheren Effizienz, den niedrigeren Betriebskosten und der schnellen Bereitstellung wurde vom Autor eine Virtuelle Serverumgebung (auch kurz VM für Virtuelle Maschine) gewählt. Das Virtualisierte Gastsystem

¹ Vgl. [Literaturverzeichnis](#): *Objektorientierte Systemanalyse*, S. 59

² Vgl. [Literaturverzeichnis](#): *Objektorientierte Systemanalyse*, S. 58

baut dabei auf einem VMWare ESX Wirtsystem auf. Installiert wurde eine Ubuntu 18.04.5 Serverinstanz in Zusammenspiel mit einem Apache HTTP Server.

4.3 Architekturdesign

Als Architekturbasis bzw. Pattern soll das in [Punkt 4.1](#) bereits beschriebene [MVC](#) Muster dienen. Der größte Vorteil darin liegt in der klaren Trennung der Verarbeitung von Daten in dem jeweiligen Model (z.B. Erstellen, Suchen und Löschen von Anträgen), die Darstellung der Daten im View (typischerweise [HTML](#) Seiten, bzw. vor dem Rendering in Blade Templates) und der Anwendungslogik im Controller (z.B. Abfragen von Anträgen vom View und das Senden dieser ans Model). Müssen beispielsweise Datenbankstrukturen geändert werden, z.B. der Spaltenname der E-Mail-Adresse des Users wird umbenannt, so müsste diese Änderung nur in der Model-Klasse des Users geändert werden. Die Anwendungslogik (Controller) und Darstellungsschicht (View) bleibt davon unberührt.

4.4 Benutzeroberfläche

Wie in [Punkt 2.4](#) beschrieben soll es in Bezug auf die Oberfläche Rücksprache mit den Fachbereichen bzw. Projektakteuren geben. Um eine gemeinsame Diskussions- und Arbeitsgrundlage zu finden wurden erste Entwürfe angefertigt. Beispiele dieser Mockups bzw. Prototypen können in [Anhang A.8](#) und [Anhang A.9](#) angesehen werden. Wie im Lastenheft gewünscht, soll die Oberfläche ein einheitlich – minimalistisches Aussehen bekommen, wobei Farben, Schriftarten und Abstände auf die Firmen Corporate Identity abgestimmt werden. Diese Anforderungen wurden auch – bei den zuvor genannten Prototypen – berücksichtigt.

Die Anwendung wird im Intranet bzw. Internen Netz der Firma laufen. Da alle Mitarbeiter an Desktopgeräten arbeiten, besteht im ersten Schritt keine konkrete Anforderung an eine optimierte Responsivität der Applikation.

4.5 Datenmodell

Das Datenmodell wurde mit Hilfe von Draw.IO³ als Entity-Relation-Modell (kurz. [ERM](#)) entworfen. Um die Verständlichkeit zu wahren befindet sich das Diagramm ohne Attribute der Entitäten im [Anhang A.10](#). Grundsätzlich teilt sich die Datenbank in zwei Anwendungsbereiche. Der erste und umfangreichste Anwendungsteil beschäftigt sich mit der Verwaltung der [requests](#). Zusammengefasst mit allen Kardinalitäten ergeben sich sieben Tabellen die in der dritten Normalform (kurz. 3NF) das Administrative Tabellarium der Anträge stellen. Diese Entitätengruppe besteht aus der schon genannten [requests](#) Tabelle sowie, [request_periods](#), [request_types](#), [comments](#), [human_resources](#) und [user_stand_ins](#).

Jeder [request](#) ist dabei einem festen [request_type](#) zuzuordnen (Bspw. „Bezahlter Urlaub“, „Sonderurlaub“, ect.) dadurch entsteht eine 1:n-Beziehung. Beim Anlegen eines [requests](#) muss in der Tabelle [request_periods](#) unter anderem der Start- und End-Timestamp des Antrags gesetzt sein, dadurch besteht auch hier eine 1:n-Beziehung.

In [humane_resources](#) werden zunächst die [user_id](#) des Erstellers und des zugewiesenen Vorgesetzten eingetragen. Anschließend werden über die [user_stand_ins](#) beliebig viele, bzw. in der Erstellungsmaske eingetragene Vertretungen verknüpft. Die Verbindungstabelle beherbergt dabei die [user_id](#), [humane_resource_id](#) und den sog. [overhanding_tstamp](#) welcher das Übergabedatum des Antragstellers an die Vertretung(en) festlegt. Die [requests](#) Tabelle steht somit ebenfalls in einer 1:n-Beziehung zur [request_periods](#) Tabelle, wobei die [user_stand_ins](#) eine n:m-Beziehung zur [request_period](#) Entität pflegt. Mit der [comments](#) Tabelle können alle Status Änderungen der [requests](#) durch die Akteure

³ Kostenlose Diagramm WebApp aus dem Atlassian Ökosystem

kommentiert werden. Grundsätzlich stellt die Auslagerung eine 1:1 Beziehung dar. Daher besteht unter Gesichtspunkten der Normalisierung keine Notwendigkeit der Exkludierung. Jedoch sind mit steigender Auslastung der Datenbank, aufgrund der Maximalen Beschreibungsgröße von 255 Zeichen pro Datensatz in der **comments** Entität die größten Datensätze der Applikation zu erwarten. Somit macht die Auslagerung aus Performancetechnischer Sicht trotzdem Sinn.

Den zweiten Anwendungsteil bildet das **users**, **roles**, **role_users** Ensemble. Die Verbindungstabelle **role_users** legt dabei in einer m:n-Beziehung, die den Benutzern zugewiesenen Benutzerrollen fest.

4.6 E-Mail

Da die Applikation von den zukünftigen Nutzern nicht permanent benutzt, sondern Prozessorientiert nach Bedarf eingesetzt wird, ist es nötig ein Kommunikationsmedium zwischen Interakteuren und Applikation in der Absenz der Benutzer zu schaffen. E-Mail-Benachrichtigungen (eng. Notifications) kommen hier infrage, da Nutzer auf diese Weise unkompliziert über Zustandsänderungen Ihrer Anträge informiert werden können. Zur Veranschaulichung der E-Mail Notification Trigger zwischen dem Ersteller und Vorgesetzten wurde in [Anhang A.21](#) ein Sequenzdiagramm erstellt.

4.7 Geschäftslogik

Aufgrund der zeitlich eng geschachtelten Entwicklungsphasen war es nicht möglich, vor Beginn der Umsetzungsbzw. Implementierungsphase der Anwendung ein Klassendiagramm zur technischen Darstellung der Geschäftslogik auszuarbeiten. Um den Dokumentationsprozess dennoch einzuhalten, wurde nachträglich ein Klassendiagramm aus dem Programmcode nach der [Reverse Engineering](#) Technik generiert. Das generierte Klassendiagramm (Auszug) befindet sich in [Anhang A.13](#) und bildet das vom Request abhängige Klassenkonstrukt mit Controlleranbindung ab.

5 Implementierungsphase

Zum Beginn der Implementierungsphase muss mithilfe des Paketmanagers [composer](#) das [Laravel](#) Projekt initiiert werden. Die genauen Projektspezifikationen können in einem Auszug [composer.json](#) in [Anhang A.14](#) und der dazugehörigen Beschreibung in [Anhang A.15](#) angesehen werden.

5.1 Implementierung des Datenmodells

Das in [Punkt 4.5](#) entworfene Datenmodell wurde in der Implementierungsphase des Datenmodells mithilfe von Migrationen im Laravelkontext integriert.

5.1.1 Exkurs: Migration

Mit sog. Migrationen kann die - für das Projekt entwickelte Datenbankstruktur - festgehalten und beliebig oft wiederhergestellt werden. In den Migrationen werden die elementaren [SQL](#) Parameter (Datentyp, Größe, ect.) und Kardinalitäten (Foreign key's ect.) der Attribute festgelegt. Eine Beispielmigration der in [Punkt 4.5](#) erwähnten **request** Entität findet sich in [Anhang A.12](#).

5.1.2 Exkurs: Eloquent

Beim Thema Datenmodell (Nachtrag zu [Punkt 4.5](#)) muss im Projektumfeld das Thema Eloquent erwähnt und erläutert werden. Eloquent ist, wie die Migrationsarchitektur bereits in [Laravel](#) integriert und eines der wichtigsten Features des Frameworks. Eloquent [ORM](#) stellt Inhalte von relationalen Datenbanken als Klassen und Objekte in [PHP](#) bereit. Eines der größten Vorteile von Eloquent sind die Beziehungen (Relations) zwischen Datenbanktabellen. Wenn diese Beziehungen einmal in den Tabellen (Migrationen) und Models erstellt worden

sind, braucht sich der Entwickler hierum später im Quellcode der Anwendung nicht mehr zu kümmern. Die angelegten Beziehungen können dann per Methode aufgerufen und genutzt werden. Ein Auszug des `request` Modells befindet sich in [Anhang A.11](#).

5.2 Implementierung der Benutzeroberfläche

Grundsätzlich baut sich die Benutzeroberfläche der Applikation in `Blade` Templates auf.

5.2.1 Exkurs: Blade

Eine weitere elementare Funktion des [Laravel](#)-Frameworks ist die Blade-Templating-Engine. Klare Vorteile der Engine ist Ihre Freizügigkeit durch die direkte Verwendung bzw. Interpretation von [PHP](#)-Code und Verarbeiten von übergebenen Parametern. Außerdem können ges. Templates oder Teilabschnitte geschachtelt werden. Das `Blade`-Template des Dashboards kann in [Anlage A.17](#) als Beispiel begutachtet werden.

Die gerenderten `Blade`-Templates werden dann als reguläre [HTML](#) Dateien ausgegeben und vom Webserver interpretiert. Die Gestaltung der Oberfläche wird mit eingebundenen [CSS](#)-Dateien umgesetzt, welche wiederum [SCSS](#) basiert mithilfe von [node.js](#) im Verbund mit dem Frontend-Framework [Bootstrap](#) serverseitig gerendert worden sind. Im [Anhang A.14](#) befinden sich Screenshots der Applikation, die nach der Implementierungsphase entstanden sind.

5.3 Implementierung der Routen

Um die `Blade`-Templates bzw. Views der Applikation nun erreichbar bzw. für den User sichtbar zu machen werden Routen angelegt, die entweder direkt oder auf Umwegen über bspw. Controller auf die angesteuerten `Blade`-Templates zeigen.

5.3.1 Exkurs: Routing

Das [Laravel](#) Routing entscheidet, welche Aktion bei welcher URL ausgeführt wird. So kann im einfachsten Fall (wie im Vorwort erwähnt) eine View angezeigt werden. Es lassen sich aber auch durch bspw. GET und POST verschiedene Aktionen zuweisen. Auch Variablen können an Methoden übergeben werden. Eine vollständiger Codeauszug der im Projekt umgesetzten Routing-Handhabung befindet sich in [Anhang A.16](#).

5.4 Implementierung der Controller

Neben den in [Punkt 5.2](#) beschriebenen View- und in [Punkt 5.1](#) erläuterten Modelbestandteilen werden im [MVC](#) Pattern mithilfe von Controller-Klassen verbunden. Aktionen der Nutzer leitet die View zum Controller weiter, welcher die dahinterstehende Programmlogik ausführt. Die Logik informiert einzelne Views im Bedarfsfall über Änderungen am Model, um eine passende Reaktion auf diese zu ermöglichen. Durch die Komplexität der Projektgegebenen Controller-Strukturen und dem begrenzten Kerndokumentationsumfangs wird im Folgenden beispielhaft die Controller-Klasse der Antrags Administration (Handlungsschritt: Vorgesetzter entscheidet über den Antrag. Screenshot [Anhang A.18.4](#)) vorgestellt. Der Codeauszug hierzu befindet sich in [Anhang A.20](#).

Das Öffnen der Anfragen-Administration wird mit folgender Routen-Struktur ([Anhang A.16](#)) initiiert:

```
Route::get('/request/{id}/decision',
    [App\Http\Controllers\Request\RequestDecision::class, 'index']);
```

Dabei greift die Route zunächst auf die Index Methode der Klasse `RequestDecision` zu. Der Konstruktor der Klasse validiert beim Aufruf ob der ausführende User angemeldet ist. Die Index Methode implementiert nun zunächst ein Objekt der `requestDetail` Klasse. In der aufgerufenen Domain wird als Parameter die angefragte

Antrags `id` (`request/{id}/decision`) übergeben. Im `requestDetail` Objekt wird nun die Methode `decision` mit der übergebenen `id` aufgerufen. Die Funktion überprüft, ob der eingeloggte Benutzer berechtigt ist den Request einzusehen und zu administrieren. Nach erfolgreicher Autorisierung liefert die Methode eine Kollektion mit allen Detailinformationen des angefragten Antrags zurück⁴. Jene Kollektion wird aus der Index Methoden-Referenz an eine Variable übergeben und komprimiert an die View weitergeleitet. Nun kann der User im Frontend mit den Administrationsmaske interagieren und seine Entscheidung absenden (Jede Entscheidung kann einen Kommentar enthalten). Der nächste Aufruf der Klasse findet nach dem Absenden der Administrationsmaske statt. Folgende Post-Route reicht die Entscheidung ein:

```
Route::get('/request_decision_submit',
    [App\Http\Controllers\Request\RequestDecision::class, 'decisionSubmit']);
```

Dabei wird in der `RequestDecision` Klasse die Methode `decisionSubmit` aufgerufen, welche die Parameter des abgeschickten Post-Http-Headers erhält (Aufgrund der [Laravel](#)-Struktur müssen [HTML](#) Form Parameter nicht an die Route übergeben werden, sondern stehen der Methode nach dem Post mit dem `request` „helper“ direkt zur Verfügung). Die Methode überprüft zunächst welche der Radiobuttons gewählt wurden und führt so entweder die Methode `changeRequestToGranted` oder `changeRequestToRejected` aus. Die jeweilige Methode ruft zunächst das Model des `requests` auf und ändert die booleschen Werte auf den übergebenen Zustand. Enthält die Administrationsanfrage einen Kommentar, wird über die Methode `buildCommentRelation` eine neue Kommentar Modelinstanz aufgebaut, befüllt und in Relation mit dem zuvor aufgerufenen Request Model gebracht bzw. gespeichert. Sind alle Models erfolgreich aktualisiert, wird der Benutzer zurück auf die Übersichtsseite geleitet (Screenshot [Anhang A.18.5](#)).

5.5 Implementierung der E-Mail Notifications

In [Laravel](#) wird für jede von der Anwendung gesendete E-Mail-Typ eine " mailable " -Klasse erstellt. Die Klassen werden mit den Parametern Absender, Empfänger aufgerufen und generieren via Blade minimalistische [Markdown](#) E-Mail's. Einmal erstellt können die Klassen im Controller wie folgt aufgerufen werden (Handlungsschritt: Antrag wurde versendet, Antragsteller und Vorgesetzter werden Informiert):

```
Mail::to($creator->email)
    ->send(newCreationMailCreator($creator->name, $requestModel->id))5
```

Um das oben genannte Beispiel zu vervollständigen befindet sich ein Codeauszug der `CreationMailCreator` Klasse und das dazugehörige Template in [Anhang A.22](#).

5.6 Testen der Anwendung

Wie schon unter [2.4 \(Entwicklungsprozess\)](#) erwähnt, wurde das Frontend bzw. der für den User ersichtliche und nutzbare Funktionsumfang nach Erreichen von Projektmeilensteinen durch Anwendertest der Fachbereiche geprüft. Die Anwendertests in der Implementationsphase garantieren die Qualitätssicherung im Frontend sowie die intuitive Usability der Applikation. Zur Sicherung der Fach- und Codequalität wurden regelmäßig Codereviews abgehalten (In [Punkt 6.1](#) näher erläutert). Ein weiteres Testing-Instrument stellten regelmäßige

⁴ Sollte die Anfrage oder Autorisierung fehlschlagen enthält die Kollektion die entsprechenden Fehlermeldungen.

⁵ `$creator->name`: Name des Erstellers, `$requestModel->id`: Primärschlüssel des Antrags,
`$creator->email`: E-Mail Adresse des Erstellers.

Enlightn⁶-Codescans dar. Hier wurden Performance, Verlässlichkeit und Sicherheit im Entwicklungsprozess in regelmäßigen Intervallen überprüft und optimiert. Ein Auszug des Protokolls befindet sich in [Anhang A.23](#).

6 Abnahme- und Deployment

6.1 Code-Review

Bevor die Anwendung zum Test der Fachschaft vorgelegt wurde, fand ein Code-Review mit der Ausbildungsleitung statt. Dabei galt es zunächst aus technischer Sicht Funktionssicherheit, Fehlerfreiheit und Spezifikationen des Projekts zu analysieren und bewerten. Vor dem eigentlichen Review fand ein Vorgespräch mit der Einarbeitung in die Projektanforderung und -umgebung statt. Auf die Einarbeitung folgte das eigentliche Review nach der „Walkthrough“ Technik. Im Kern dieses Verfahrens geht es um das strukturierte Suchen, Finden und Korrigieren von Abweichungen gegenüber den gängigen Firmeninternen Qualitätskriterien in allen Kernmodulen des Projekts. Das Erfüllen der Qualitätskriterien stellte somit das zentrale Ziel des Code-Reviews dar.

6.2 Deployment

Das Deployment lief aufgrund des vorgegebenen Zeitrahmens manuell ab. Der Webserver und Datenbank ([Punkt 4.2.1](#)) wurden mithilfe der Hosting Control Plattform „ISPConfig“ aufgesetzt. Manuell wurde eine interne Domain vergeben und in der „vhost“ -Konfiguration des Webservers auf das [Laravel](#)-Rootverzeichnis gelegt. Anschließend wurde die Datenbank Basis Migration der Applikation angestoßen, um alle benötigten Entitäten und Relationen zu erstellen. Bei der Initiierung des Projektes mussten keine Daten migriert werden. Allerdings mussten mehrere Datenbank-[Seeds](#) ausgeführt werden, um die Grundfunktionalität zu gewährleisten (CLI-Code):

```
php artisan db:seed --class=ulladefaultroleseeder
php artisan db:seed --class=ulladefaultadminseed
php artisan db:seed --class=ulladefaultrequesttypeSeeder
```

Der **roleseed** integriert hierbei zunächst die festgelegten User Rollen⁷, **adminseed** setzt einen initialen Admin User, welcher elementar für die Einrichtung der Applikation nötig ist. Zuletzt pflegt der **typeseeder** die verschiedenen Antragstypen in die Datenbank ein. Anschließend ist die Applikation unter <https://ulla.atrивio.net/> voll funktionsfähig und somit bereit für die Benutzerschulung.

7 Dokumentation

Der Dokumentationsrahmen der Anwendung setzt sich neben der Projektdokumentation, in der alle Projektphasen des Umsetzungsrahmens beschrieben werden, auch aus der Entwicklungsdokumentation zusammen.

Die Entwicklungsdokumentation behandelt detailliert – technisch versiert die Punkte Setup und Funktion. Im Punkt „Setup“ wird dabei erklärt wie das Projekt erfolgreich aufgesetzt, bzw. lokal installiert werden kann. Der Punkt „Funktion“ enthält eine detaillierte Beschreibung aller implementierten Klassen. Dazu gehören deren Attribute, Methoden und Abhängigkeiten. Dieses Dokument soll bei der Weiterentwicklung oder Anpassung durch andere Entwickler als erste Anlaufstelle und Einarbeitungsmöglichkeit zentral im Git-Verzeichnis der

⁶ Die Enlightn OSS (Open Source Software, für Laravel optimiert) Version verfügt über 64 automatische Checks, die Anwendungscode, Webserver-Konfigurationen und Routen scannt, um Performance-Engpässe, mögliche Sicherheitslücken und Probleme mit der Code-Zuverlässigkeit zu identifizieren.

⁷ Mitarbeiter, Vorgesetzter, Verwaltung und Admin.

Applikation dienen. Daher wurde für die Entwicklungsdokumentation das Dateiformat [Markdown](#) genutzt. Ein Auszug des Punktes „Setup“ der Entwicklungsdokumentation findet sich in [Anhang A.25](#).

8 Fazit

8.1 Soll-/ Ist-Vergleich

Im folgenden Absatz soll zum Abschluss des Projekts ein Rückblick stattfinden, um den Projektabschluss, Projektziel und den zeitlichen Umsetzungsrahmen rückblickend differenziert zu bewerten. Das Projektziel wurde in [Punkt 1.3](#) aufgestellt und ergab sich aus der in [Punkt 1.4](#) beschriebenen Problemstellung bzw. Begründung. Die genannten Projektziele wurden erfolgreich umgesetzt. Der Auftraggeber bzw. Atrivio ([Punkt 1.2](#)) ist mit dem bisherigen Stand zufrieden und sieht bereits erste Erweiterungsmöglichkeiten vor (Genaue Ausführung in [Punkt 8.3](#)). Die vorgegebene Zeit von 70 Stunden wurde insgesamt eingehalten, allerdings ergaben sich Verschiebungen in der Zusammensetzung. Die Implementierungsphase schlug mit einem gesamten Mehraufwand von zwei Stunden am meisten zu Buche. Zurückzuführen ist diese Abweichung auf den unterschätzten Aufwand der Datenbank Relationsprogrammierung. Mit steigender Anzahl der Relationen und Kardinalitäten werden die Modelle der Relationen im backend linear komplexer und somit auch zeitintensiver zu programmieren. Kompensiert wurde diese Fehleinschätzung durch Einsparungen in der Analyse- u. Entwurfsphase. Beide Phasen wurden im Projektantrag überschätzt und konnten somit ohne Abstriche verkürzt umgesetzt werden. Aus Erfahrungswerten wurde zum Absichern von möglichen Eventualitäten in der Projektphase „Abnahme und Deployment“ im Punkt „Deployment der Applikation“ mehr Zeit als nötig geschätzt. Nachdem der Deployment Prozess problemlos ohne Zwischenfälle vollzogen wurde, konnte hier ebenfalls eine Stunde gespart werden. Die gewonnene Stunde wurde sinnvoll in die Dokumentation der Applikation investiert. Eine tabellarische Aufstellung des Soll-/ Ist-Vergleichs mit den zuvor genannten Abweichungen befindet sich in [Anhang A.24](#).

8.2 Lessons Learned

Mit der Umsetzung der Applikation bzw. des Abschlussprojektes gewann der Autor wertvolle Erfahrungen im gesamten Handling (dt. Handhabung) eines Projektes von der Planung und Entwurf bis hin zur Veröffentlichung. Vor allem im Bereich der Planung wurden – für die berufliche Zukunft – wichtige Erkenntnisse im Bereich Software Architektur, Pattern und [UML](#) gewonnen. Spannend war der Einsatz der verschiedenen Frameworks und der präzise Einsatz des [Laravel](#)-Ökosystems. Durch jenen versierten Einsatz lernte der Autor aber auch den Aufwand der Eloquent Integration, besonders in Bezug auf die Kardinalitäten der Relationsmodelle, nicht zu unterschätzen. In zukünftigen Projekten wird diese Erfahrung in der Zeitplanung berücksichtigt werden, um noch genauer die Kosten des Projektes einschätzen zu können.

8.3 Ausblick

Auch wenn das Projekt nach den gestellten Anforderungen bzw. Zielsetzungen umgesetzt wurde, sind zukünftige Erweiterungen nicht ausgeschlossen. So ist eine Anbindung an die firmeninternen Projektleitungs- und Verwaltungssoftwarelösungen wie Salesforce⁸ oder DATEV⁹ denkbar. Die [Laravel](#) Architektur macht die Schnittstellenintegration bzw. API Anbindung (RESTful, Soap bspw. möglich) dafür zeitnah realisierbar. Des Weiteren sollten die Anträge in diverse Dateiformate (bspw. PDF) exportierbar sein. Ebenfalls bietet das Dashboard noch Erweiterungspotenzial wie Statistiken, Shortcuts und Einstellungsmöglichkeiten.

⁸ <https://www.salesforce.com/>

⁹ <https://www.datev.de/>

Literaturverzeichnis

Prof. Dr. Gert Heinrich, Klaus Mairon (2008) *Objektorientierte Systemanalyse*,
Oldenbourg Wissenschaftsverlag GmbH, ISBN 978-3-486-58366-3

Dr. Dr. h. c. Niklaus Wirth *Algorithmen und Datenstrukturen*,
B.G. Teubner Stuttgart, ISBN 3-519-02250-8

Matt Stauffer (2017) *Laravel: Up and Running*,
O'Reilly Books USA

Dietmar Stoiber (2005) *Implementierung des Model-View-Controller Paradigmas*
Diplomarbeit Universität Linz

Ausbildungsverordnung der IHK Schwaben,

Link:

www.schwaben.ihk.de/produktmarken/meine-pruefung/ausbildungspruefungen/pruefungen-a-z/fachinformatiker-552906

geöffnet am 06.05.2021, 14:11 Uhr

Projektverordnung der IHK Schwaben,

Link:

www.schwaben.ihk.de/blueprint/servlet/resource/blob/2366802/3aac584a33671862b8b9027gf6324dd5/projektantrag-it-berufe-data.pdf

geöffnet am 06.05.2021, 16:20 Uhr

Quellen

Erklärungen Glossar u. Abkürzungsverzeichnis,

Link:

www.it-journal.de/

www.laravel.com/

www.php.net/

geöffnet am 11.05.2021, 16:50 Uhr

Anhang

A.1 Detaillierte Zeitplanung

Auszug aus dem Projektkonzept:

Analyse	3.5h
Durchführung einer Ist-Analyse	1.0h
Wirtschaftlichkeitsprüfung, (z.b Amortisationsrechnung & Kostenanalyse)	1.0h
Erstellung eines vereinfachten Lastenhefts	1.0h
Ermittlung des Use – Case (bzw. Use – Case Diagramm)	0.5h
Entwurf	8.0h
Erstellung eines vereinfachten Pflichtenhefts	3.0h
Aktivitätsdiagramm zu den einzelnen Projektsegmenten	1.0h
Frontend-Mockup	1.0h
Planung des Datenbankmodells	2.5h
ER-Diagramm o. DB Schema	0.5h
Entwicklung & Implementierung	42.5h
Implementierung des Datenbankmodells bzw. der Datenbanklogik	2.0h
Erstellung des Antrags-Interface	5.0h
Erstellung des Management-Interface	4.0h
Erstellung des Dashboards	4.0h
Erstellung des Verwaltungs-Interface	4.0h
Entwicklung der Zuweisungslogik	2.0h
Entwicklung der Mail-Notifications	3.0h
Entwicklung der Abnahme Logik	2.0h
Entwicklung der Ablehnungs Logik	3.0h
Einrichtung der Routen	1.5h
Frontend Erstellung nach Mockup	6.0h
Implementierung der Benutzer Authentifikation & Rechte	6.0h
Abnahme und Deployment	3.0h
Code-Review mit dem Ausbilder u. ggf Fachabteilung(en)	1.0h
Abnahme durch einen Ausgewählten Fachbereich	0.5h
Deployment der Applikation	1.5h
Dokumentation	13.0h
Erstellen der Projektdokumentation	12.0h
Erstellen der Entwicklungsdokumentation u.a. für zukünftige Erweiterungen. Die Projektdokumentation wird hier als Basis verwendet u. ggf. erweitert.	1.0h
Gesamt	70h

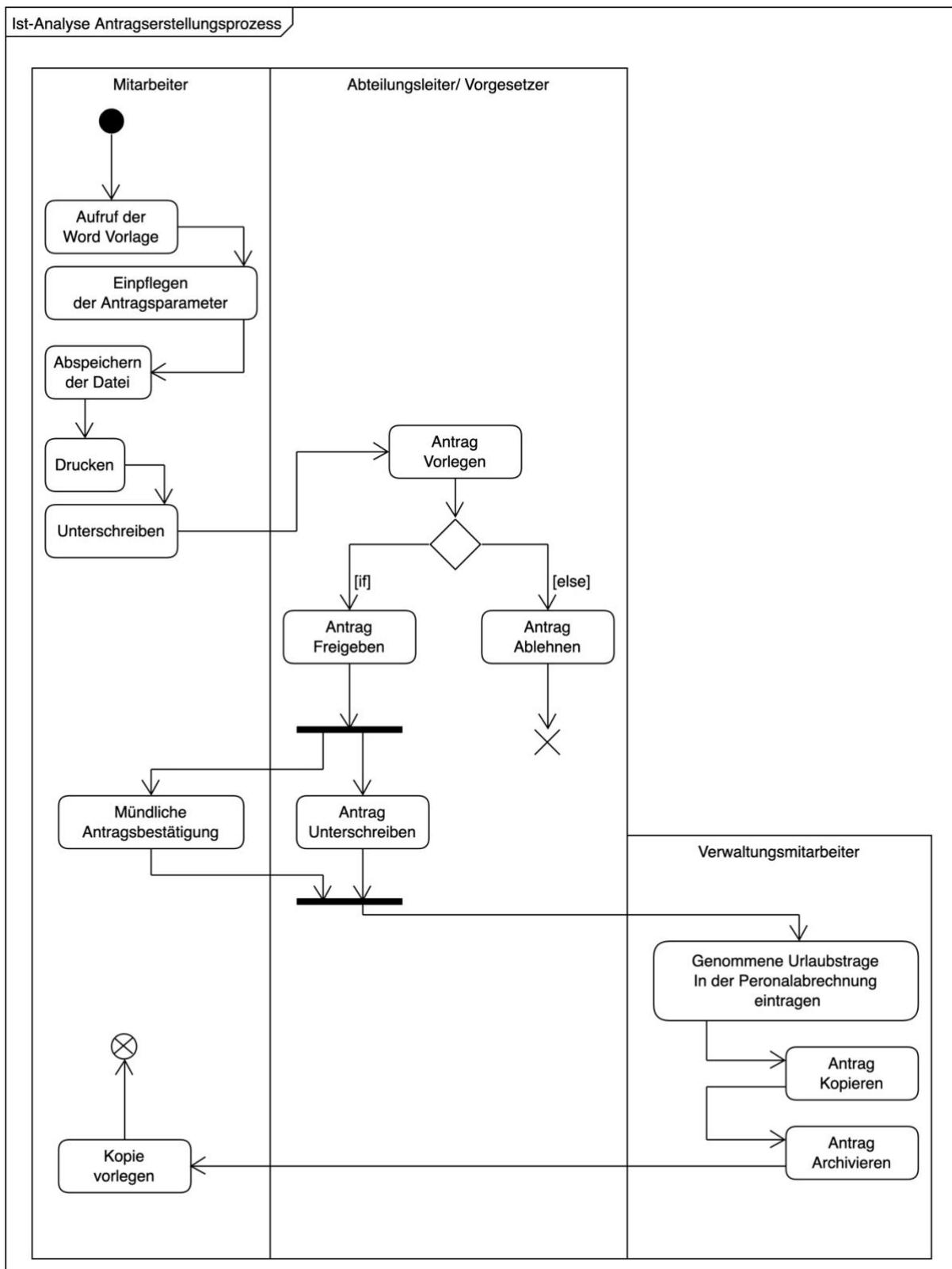
Tabelle II - Detaillierte Aufwandsschätzung

A.2 Ressourcenplan

Typ	Bezeichnung
Hardware	Tisch
	Stuhl
	Notebook
	Monitor
	Tastatur
	Maus
	Headset
Software	macOS Big Sur 11.2.3
	Ubuntu 20.04.02 LTS
	Ubuntu Server 18.02
	VMWare ESX
	Docker desktop 3.3.1
	Docker Container mysql/mysql-server (lokal)
	Docker Container dockage/mailcatcher (lokal)
	GIT (CLI)
	Jetbrains Datagrip
	Microsoft Visual Studio Code
	Microsoft Visual Studio Code UMLet Extension
	Microsoft Teams
	Composer (CLI)
	NPM (CLI)
	iTerm (im Verbund mit ZSH)
Personal	Leiter der Entwicklung Kundenentwicklung, Ausbildungsleiter
	Mitarbeiter der Verwaltung
	Mitarbeiter Testing bzw. Projektleitung
	Anwendungsentwickler, Prüfling

Tabelle III - Ressourcenplan

A.3 Ist-Analyse - Aktivitätsdiagramm



A.4 Kalkulatorische Aufstellung

Position	Kosten	Stunden	Gesamt
Herr Müller (Auszubildender Entwicklung, Prüfling)	65€	70h	4.550,00€
Herr Tamler (IT Consultant / Head of Development)	130€	8h	1.040,00€
Frau Klasen (Auszubildende Projektleitung, Testing)	65€	2h	130,00€
Gesamt	5.720,00€		

Tabelle IV - Kalkulatorische Aufstellung der Entwicklungskosten

A.5 Zeiteinsparung

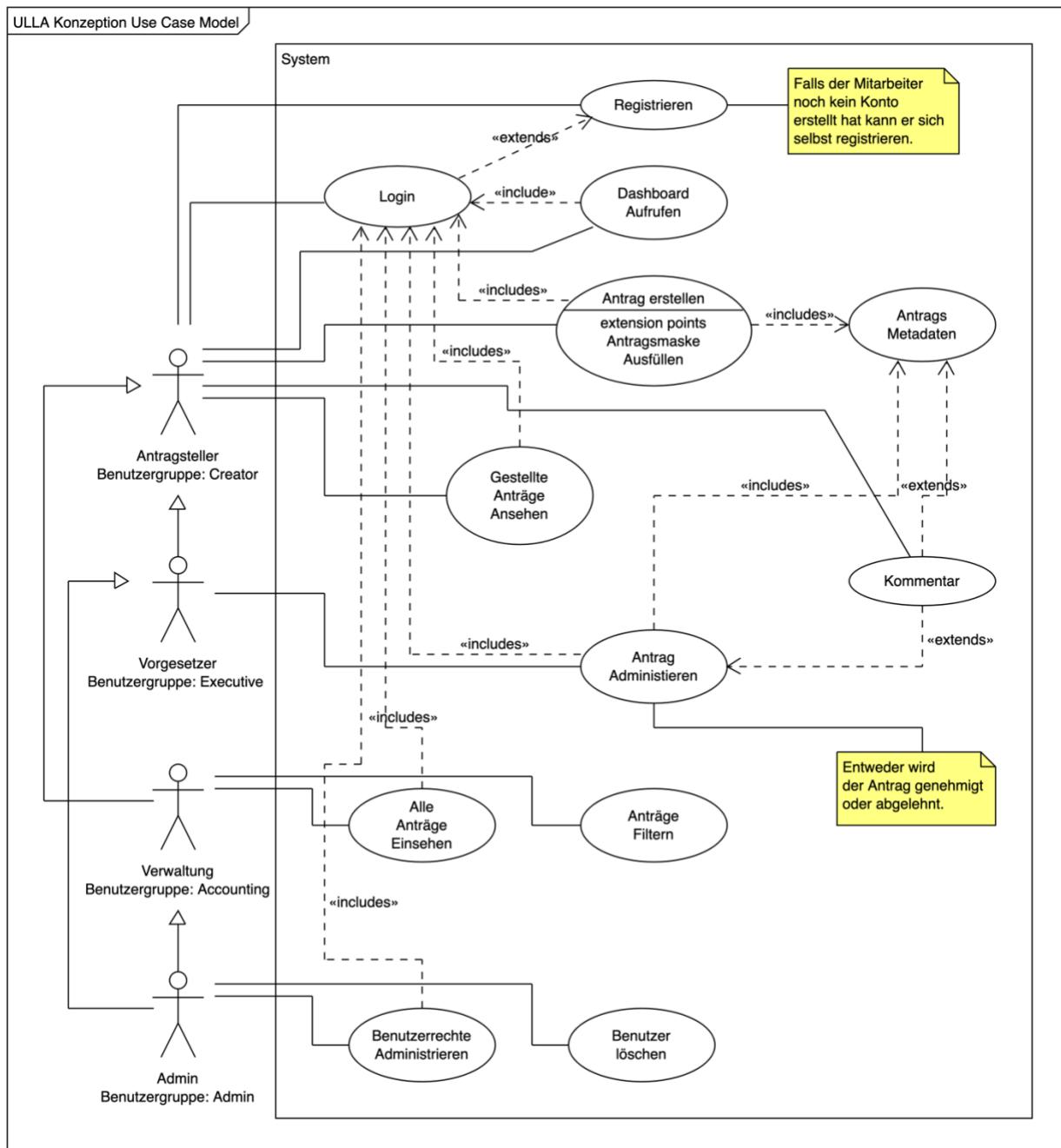
Stelle	Kosten in € pro Stunde	Ersparnis in Minuten	Gesamt
Mitarbeiter	97,5 *	15	24,38€
Vorgesetzter	97,5 *	10	16,25€
Verwaltung	97,5 *	20	32,50€
Ersparnis pro Antrag	73,13€		

Tabelle V - Zeiteinsparungskalkulation

* Durchschnittswerte aus den unterschiedlichen Besoldungsgruppen [(130 + 65) / 2].

Berechnung: (73,13€ * 3 Anträge pro Jahr) * 20 Mitarbeiter = 4387,80€ pro Jahr

A.6 Anwendungsfälle – Use-Case-Diagramm



A.7 Auszug Lastenheft


Lastenheft
Projekt: ULLA

5.1 Funktionale Anforderungen

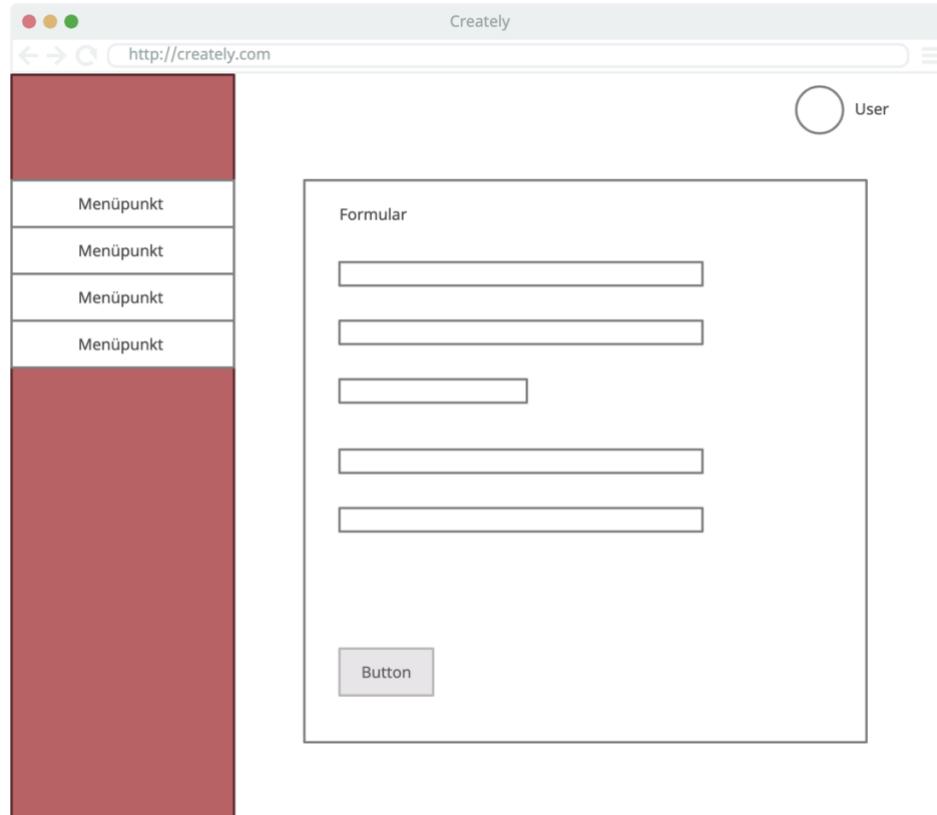
Die funktionalen Anforderungen beschreiben die Anforderungen an die zu entwickelnde Software. Der Anforderungskatalog ist aus Sicht des Anwenders verfasst.

Nr.	Beschreibung	Prio	Status
ULLA01	Neue Anträge müssen schnell und einfach erstellt werden können.	9	p. ¹
ULLA02	Die Antragsmaske muss auch zukünftig erweiterbar sein.	9	p.
ULLA03	Die Antragsmaske muss mindestens die Parameter der bisherigen Vorlage enthalten.	9	p.
ULLA04	Es muss eine zentrale, filterbare Anlaufstelle zur Einsicht aller Daten geben.	9	p.
ULLA05	User müssen Rollen nach Ihren Berechtigungen zugewiesen werden können.	9	p.
ULLA06	Die Rollenzuweisung muss zentral administrierbar sein.	8	p.
ULLA07	Anträge müssen den Vorgesetzten zugewiesen werden können.	9	p.
ULLA08	Vorgesetzte müssen die Anträge online zulassen oder ablehnen können.	9	p.
ULLA09	Vorgesetzte, Mitarbeiter und Verwaltungsmitarbeiter müssen über den Status Ihres Antrages per Mail erinnert werden.	6	p.
ULLA10	Ist ein Antrag bestätigt enthält die Bestätigungsmaile einen Vorgefertigten Kalendereintrag der schnell und einfach in den Outlookkalender des Mitarbeiters übernommen werden kann.	2	p.
ULLA10	Eine REST-Schnittstelle ermöglicht den einfachen Export von Anträgen.	2	p.
ULLA11	Alle Zustandsveränderungen der Anträge (Pending, Granted, Rejected) können mit Kommentaren versehen werden um Entscheidungen nachvollziehbar zu gestalten.	6	p.
ULLA12	Layout, Schriftart der Applikation soll an das Firmeninterne CI angepasst sein.	3	p.

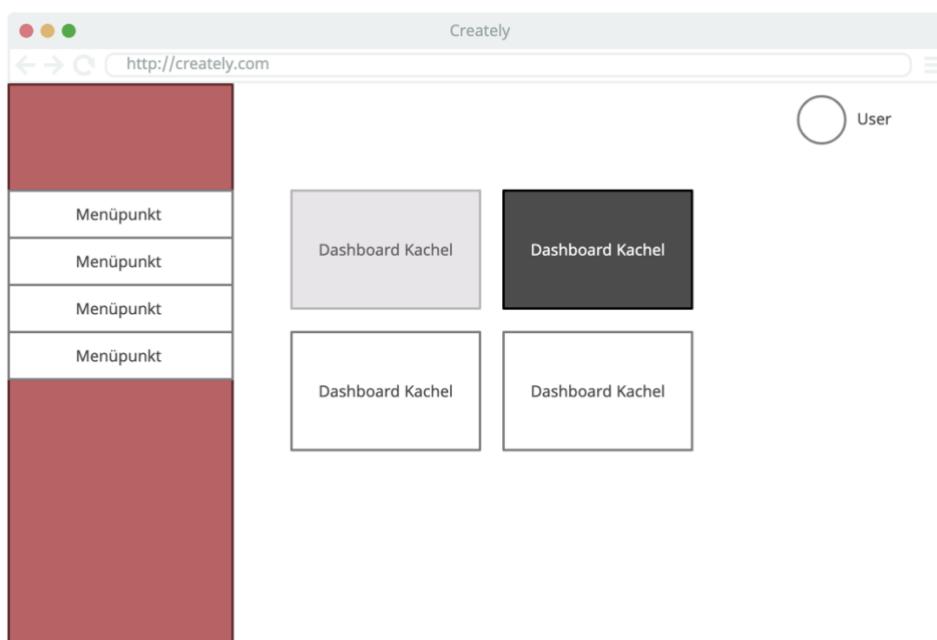
¹ Pending / Ausstehend

Müller, Maximilian
S. 6/8

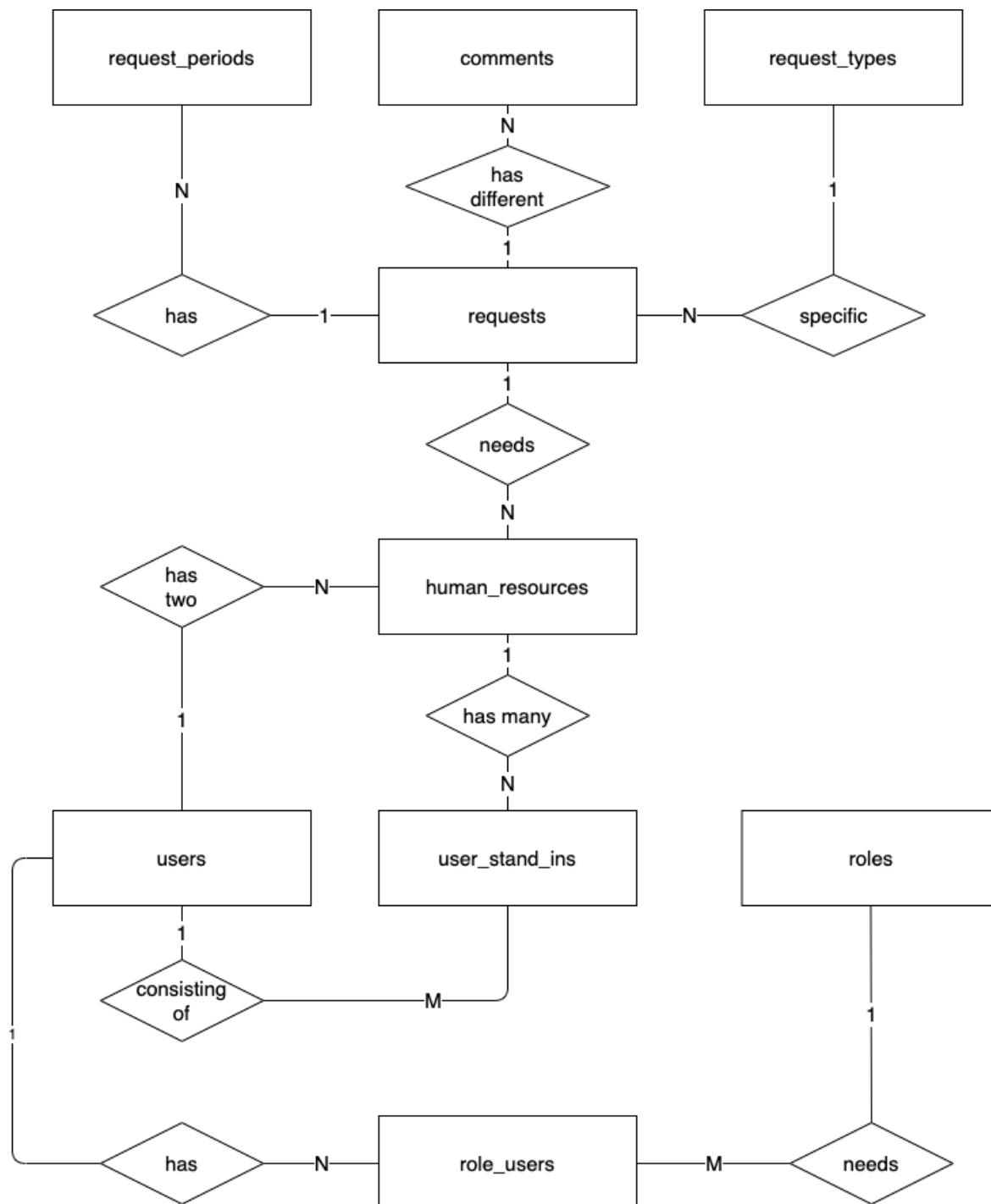
A.8 Mockup - Formular



A.9 Mockup - Dashboard



A.10 Entity Relationship Modell



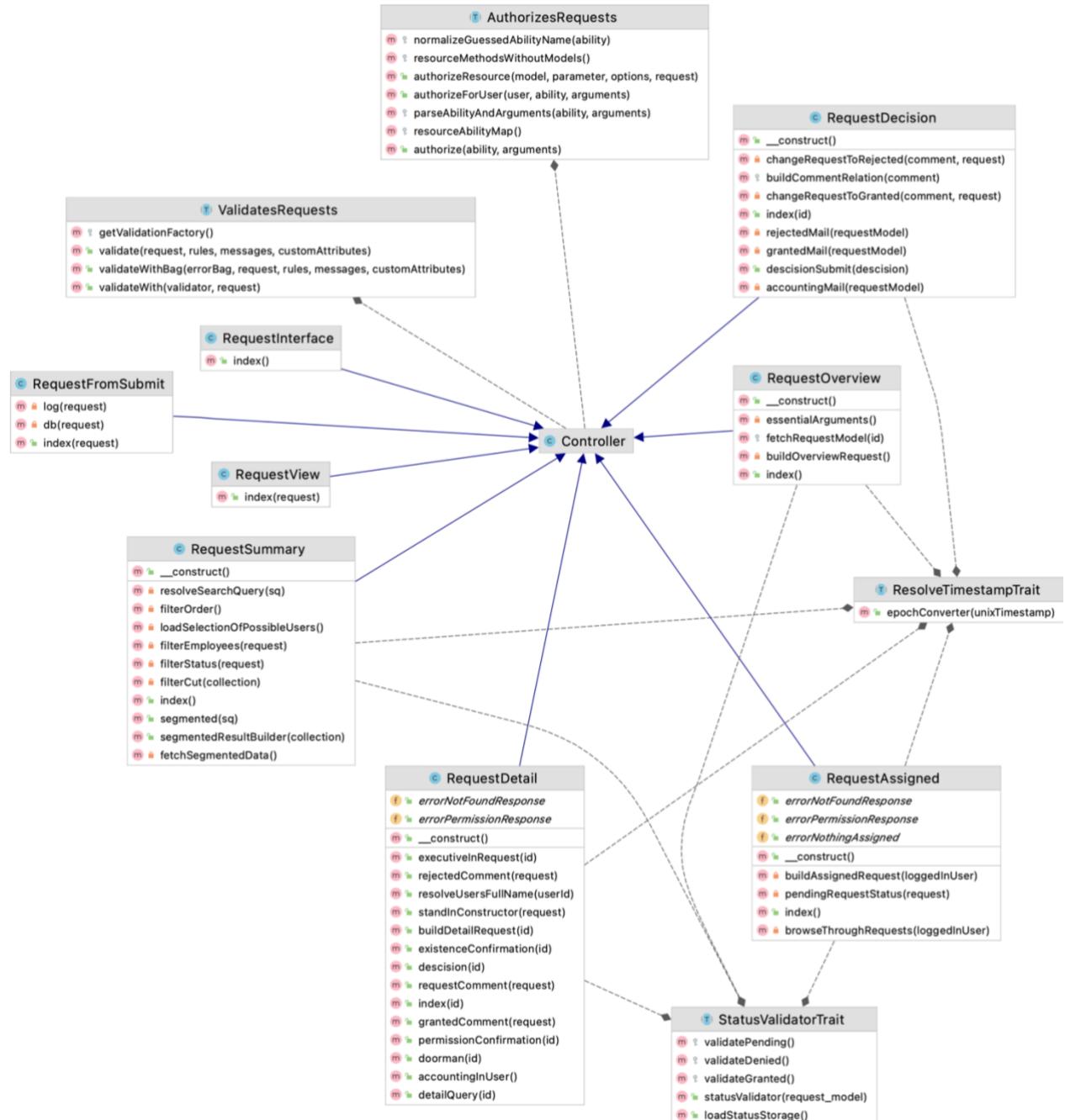
A.11 Request Eloquent Model - Codeauszug

```
● ● ●
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Request extends Model
9 {
10     use HasFactory;
11
12     public function request_type()
13     {
14         return $this->belongsTo(Request_type::class, 'request_type_id', 'id');
15     }
16
17     public function request_comment()
18     {
19         return $this->belongsTo(Comment::class, 'request_comment_id', 'id');
20     }
21
22     public function granted_comment()
23     {
24         return $this->belongsTo(Comment::class, 'granted_comment_id', 'id');
25     }
26
27     public function rejected_comment()
28     {
29         return $this->belongsTo(Comment::class, 'rejected_comment_id', 'id');
30     }
31
32     public function period()
33     {
34         return $this->hasOne(Period::class, 'id', 'id');
35     }
36
37     public function human_resource()
38     {
39         return $this->hasOne(Human_resource::class, 'id', 'id');
40     }
41
42     public function stand_in_users()
43     {
44         return $this->hasOne(User_stand_in::class, 'request_stand_in_id', 'id');
45     }
46 }
47
```

A.12 Request Migration - Codeauszug

```
● ● ●
1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 class CreateRequestsTable extends Migration
8 {
9     /**
10      * Run the migrations.
11      *
12      * @return void
13      */
14     public function up()
15     {
16         Schema::create('requests', function (Blueprint $table) {
17             $table->bigIncrements('id');
18             $table->unsignedBigInteger('request_type_id')->nullable();
19             $table->unsignedBigInteger('request_comment_id')->nullable();
20             $table->unsignedBigInteger('granted_comment_id')->nullable();
21             $table->unsignedBigInteger('rejected_comment_id')->nullable();
22             $table->boolean('granted')->nullable();
23             $table->boolean('rejected')->nullable();
24             $table->timestamps();
25
26             $table->foreign('request_type_id')->references('id')->on('request_types')
27                 ->onDelete('cascade');
28
29             $table->foreign('request_comment_id')->references('id')->on('comments')
30                 ->onDelete('cascade');
31
32             $table->foreign('granted_comment_id')->references('id')->on('comments')
33                 ->onDelete('cascade');
34
35             $table->foreign('rejected_comment_id')->references('id')->on('comments')
36                 ->onDelete('cascade');
37         });
38     }
39
40     /**
41      * Reverse the migrations.
42      *
43      * @return void
44      */
45     public function down()
46     {
47         Schema::dropIfExists('requests');
48     }
49 }
50
```

A.13 Request - Klassendiagramm



A.14 Composer Spezifikation – Codeauszug

```

1  {
2      "name": "laravel/laravel",
3      "type": "project",
4      "description": "The Laravel Framework.",
5      "keywords": [
6          "framework",
7          "laravel"
8      ],
9      "license": "MIT",
10     "require": {
11         "php": "^7.3|^8.0",
12         "fideloper/proxy": "^4.4",
13         "fruitcake/laravel-cors": "^2.0",
14         "guzzlehttp/guzzle": "^7.0.1",
15         "laravel/framework": "^8.12",
16         "laravel/tinker": "^2.5",
17         "laravel/ui": "^3.2",
18         "santigarcor/laratrust": "^6.3",
19         "spatie/calendar-links": "^1.4"
20     },
21     "require-dev": {
22         "facade/ignition": "^2.5",
23         "fakerphp/faker": "^1.9.1",
24         "mockery/mockery": "^1.4.2",
25         "nunomaduro/collision": "^5.0",
26         "phpunit/phpunit": "^9.3.3"
27     },
28     "config": {
29         "optimize-autoloader": true,
30         "preferred-install": "dist",
31         "sort-packages": true
32     },
33     "extra": {
34         "laravel": {
35             "dont-discover": []
36         }
37     },
38     "autoload": {
39         "psr-4": {
40             "App\\": "app/",
41             "Database\\Factories\\": "database/factories/",
42             "Database\\Seeders\\": "database/seeders/"
43         }
44     },

```

In [Anhang A.15](#)
 erläutert.

A.15 Composer Spezifikation – Erklärung

In der folgenden Erklärung werden die im Projekt verwendeten und in der [composer.json](#) aufgeführten [packagist](#) Pakete vorgestellt.

Package Name	Erklärung	Basis:	Erweitert ¹⁰ :
"php"	Definiert die im Projekt vorgesehenen PHP Version.	✓	
"fideloper/proxy"	Die Einstellung eines vertrauenswürdigen Proxys ermöglicht die korrekte URL-Generierung, Umleitung, Sitzungsbehandlung und Protokollierung in Laravel.	✓	
"fruitcake/laravel-cors"	Ermöglicht es, Cross-Origin Resource Sharing-Header mit der Laravel-Middleware-Konfiguration zu senden.	✓	
"guzzlehttp/guzzle"	Guzzle ist ein PHP-HTTP-Client, der das Senden von HTTP-Anfragen vereinfacht und die Integration mit Webdiensten trivial ermöglicht.	✓	
"laravel/framework"	Laravel repository.	✓	
"laravel/tinker"	Laravel Tinker ist eine leistungsfähige REPL für das Laravel-Framework.	✓	
"laravel/ui"	Einfaches Authentifizierungsgerüst, das auf dem CSS-Framework Bootstrap aufbaut.	✓	
"laravel/laratrust"	Dieses Paket bietet eine flexible Möglichkeit, rollenbasierte Berechtigungen zu Laravel hinzuzufügen.	✓	
"spatie/calendar-links"	Generieren von Links zum Hinzufügen zum Kalender für Google, iCal und andere Kalendersystemen.	✓	

Tabelle VI - Composer Spezifikation im Vergleich: Basis - Erweitert

¹⁰ Erweiterungen die vom Autor für das Projekt installiert wurden.

A.16 Routen – Codeauszug



```

1  <?php
2
3  use Illuminate\Support\Facades\Route;
4
5  Route::redirect('/', 'login');
6  Route::redirect('home', 'dashboard');
7  Route::post('change_user_role', [App\Http\Controllers\UserActions\updateUserRole::class, 'index']);
8  Route::post('delete_user', [App\Http\Controllers\UserActions\deleteUser::class, 'index']);
9  Route::post('request_form_data', [App\Http\Controllers\Request\RequestFromSubmit::class, 'index']);
10 Route::post('request_submit', [App\Http\Controllers\Request\RequestDecision::class, 'descisionSubmit']);
11 Route::get('/dashboard', [App\Http\Controllers\HomeController::class, 'index'])->name('home');
12 Route::get('/permission-board', [App\Http\Controllers\Tasks\TasksPermissionController::class, 'index'])->name('permission-board');
13 Route::get('/request-interface', [App\Http\Controllers\Request\RequestInterfaceController::class, 'index'])->name('request-interface');
14 Route::get('/request/view', [App\Http\Controllers\Request\RequestView::class, 'index'])->name('request-view');
15 Route::get('/request/summary', [App\Http\Controllers\Request\RequestSummary::class, 'index']);
16 Route::get('/request/overview', [App\Http\Controllers\Request\RequestOverview::class, 'index'])->name('request-overview');
17 Route::get('/request/detail', [App\Http\Controllers\Request\RequestDetail::class, 'index'])->name('request-detail');
18 Route::get('/request/decision', [App\Http\Controllers\Request\RequestDecision::class, 'index'])->name('request-decision');
19 Route::get('/request/assigned', [App\Http\Controllers\Request\RequestAssigned::class, 'index'])->name('request-assigned');
20 Route::get('/request/segmented/{sq}', [App\Http\Controllers\Request\RequestSummary::class, 'segmented']);
21 Route::get('/request/success', function () {
22     return view('request.success');
23 });
24 Route::get('/app', function () {
25     return view('layouts.app');
26 });
27 Auth::routes();
28

```

¹¹ Um eine bessere Lesbarkeit der Grafik zu gewährleisten wurde die Anlage um 90° gedreht.

A.17 Blade – Codeauszug

Mit `@extends('layouts.app')` werden Basiselemente (header, footer, ect.) in das Template eingebunden.
`@section('content')` legt den Contentabschnitt im Elterntemplate fest.

```
● ● ●
1 @extends('layouts.app')
2
3 @section('content')
4
5 <div class="row justify-content-left">
6     <div class="w-25 p-3 mt-2">
7         <div class="card rounded-0 custom-border-default">
8             <div class="card-body h4 mt-2">
9                 <div>{{ __('Herzlich willkommen') }}</div>
10                <div class="font-weight-bold">{{ Auth::user()->name }}</div>
11            </div>
12        </div>
13    </div>
14
15    <div class="w-25 p-3 mt-2">
16        <div class="card rounded-0 custom-border-default">
17            <div class="card-body h4 mt-2">
18                <div>{{ __('Benutzergruppe:') }}</div>
19                <div class="font-weight-bold">{{ Auth::user()->roles->first()->name }}</div>
20            </div>
21        </div>
22    </div>
23 </div>
24
25 @endsection
```

A.18 Screenshots der Applikation

(Alle Screenshots sind nach der Implementierungsphase entstanden.)

A.18.1 Login-Maske

Login Register

Login

E-Mail Address

Password

Remember Me

[Forgot Your Password?](#)

Login-Maske der Applikation

A.18.2 Dashboard

ULLA

Neu

Meine Anträge

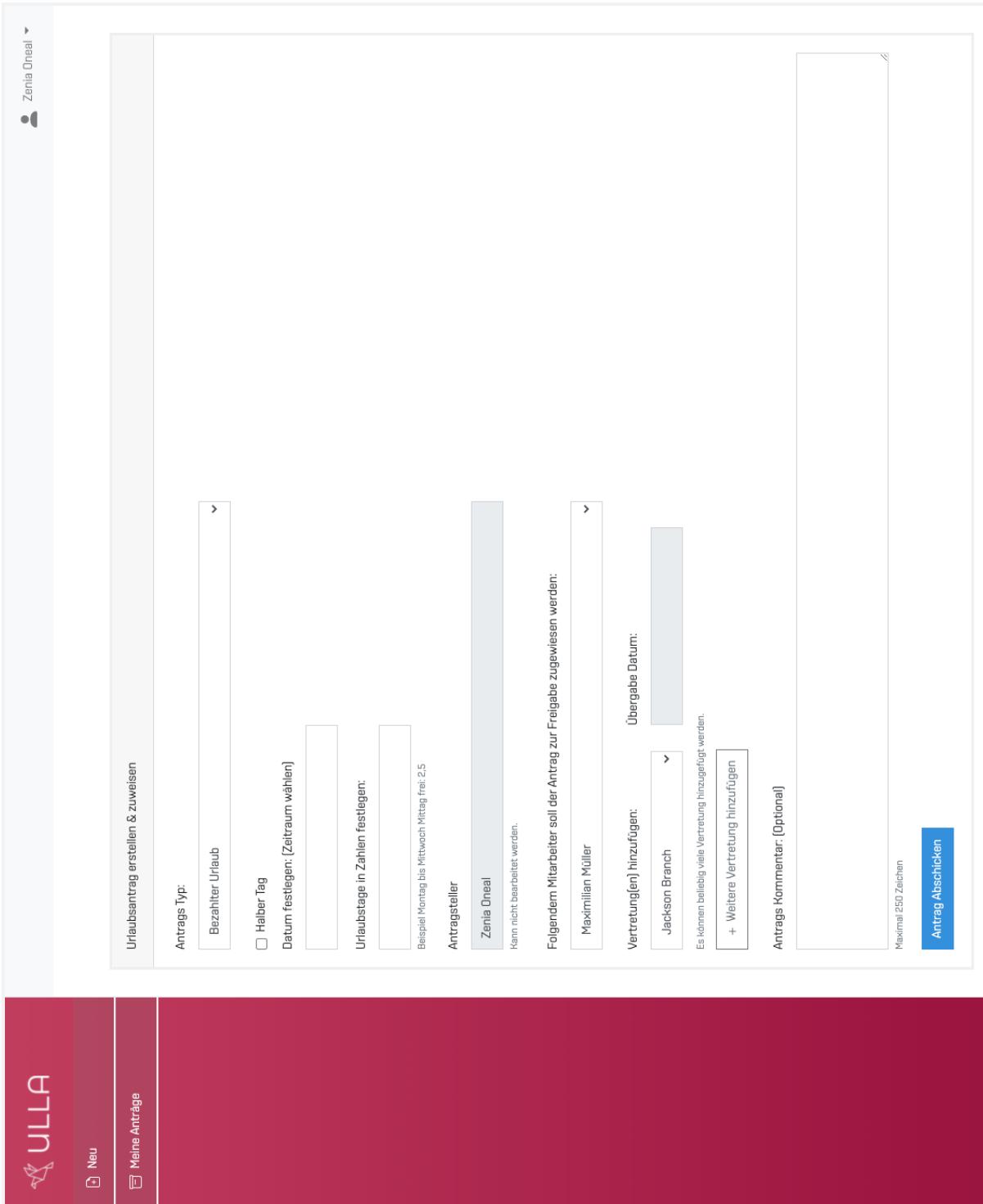
Herzlich willkommen
Zenia Oneal

Benutzergruppe:
staff

Zenia Oneal

Dashboard der Applikation aus der Sicht eines Mitarbeiters

A.18.3 Erstellungs-Maske



Zenia Onel ▾

Urlaubsantrag erstellen & zuweisen

Antrags Typ:

Bezahlter Urlaub

Halber Tag

Datum festlegen: [Zeitraum wählen]

Urlaubstage in Zahlen festlegen:

Beispiel Montag bis Mittwoch Mittag frei: 2,5

Antragsteller

Zenia Onel

Kann nicht bearbeitet werden.

Folgendem Mitarbeiter soll der Antrag zur Freigabe zugewiesen werden:

Maximilian Müller

Vertretung(en) hinzufügen:

Jackson Branch

Übergabe Datum:

Es können beliebig viele Vertretung hinzugefügt werden.

+ Weitere Vertretung hinzufügen

Antrags Kommentar: [Optional]

Maximal 250 Zeichen

Antrag Abschicken

12

Antrags Erstellungsmaske aus der Sicht eines Mitarbeiters

¹² Um eine bessere Lesbarkeit der Grafik zu gewährleisten wurde die Anlage um 90° gedreht.

A.18.4 Antrags Administration

The screenshot displays the ULLA web application's administrative interface for leave requests. On the left, a vertical sidebar has a red header with the ULLA logo and three menu items: 'Neu', 'Meine Anträge', and 'Mir zugewiesen'. The main content area shows a leave request for employee #10. The top section is titled 'Antrag #10' and contains a note: 'Auswahl treffen ob der unten aufgeführte Antrag genehmigt oder abgelehnt werden soll.' Below this are sections for 'Auswählen:' (with radio buttons for 'Genehmigen' and 'Ablehnen'), 'Kommentar: [Optional]' (a text area with a character limit of 250), and a 'Bestätigen' button. The middle section, titled 'Details:', lists general information: Ersteller: Zenia Oneal, Zugewiesen: Maximilian Müller. It also specifies the time period: Von: 24.04.2021, Bis: 30.04.2021, and Gesamt: 5 Tage. The bottom section, titled 'Kommentar(e)', shows a comment from Zenia Oneal: 'Sommerurlaub für dieses Jahr'.

Antrags Administrationsmaske Aus Der Sicht Eines Vorgesetzten

A.18.5 Antrags Übersicht

The screenshot shows the 'Antrags Übersicht' (Leave Application Overview) page. On the left is a red sidebar with navigation links: 'ULLA' (with a logo), 'Neu', 'Meine Anträge', and 'Mir zugewiesen'. The main area has three sections:

- Zugewiesen:** A message box states: 'Zugewiesene Anträge die noch nicht bearbeitet [freigegeben/abgelehnt] wurden werden unter ausstehend gelistet.' Below it says: 'Ist ein Antrag einmal bearbeitet kann dieser nicht mehr geändert werden.' A table lists three items:

ID	Datum:	Antragsteller/inn:	Status:	Aktion:
#8	21.04.2021 bis 21.04.2021	Maximilian Müller	∅	<button>Entscheiden</button>
#9	16.04.2021 bis 22.04.2021	admin	∅	<button>Entscheiden</button>
#10	24.04.2021 bis 30.04.2021	Zenia Oneal	∅	<button>Entscheiden</button>
- Ausstehend:** A table lists five items:

ID	Datum:	Antragsteller/inn:	Status:	Aktion:
#2	20.04.2021 bis 24.04.2021	Halee Allison	✓	<button>Einsehen</button>
#3	21.04.2021 bis 24.04.2021	admin	✓	<button>Einsehen</button>
#4	29.04.2021 bis 29.04.2021	admin	✓	<button>Einsehen</button>
#6	30.04.2021 bis 30.04.2021	admin	✓	<button>Einsehen</button>
#7	22.04.2021 bis 22.04.2021	admin	✓	<button>Einsehen</button>
- Bearbeitet:** A table lists five items:

ID	Datum:	Antragsteller/inn:	Status:	Aktion:
#2	20.04.2021 bis 24.04.2021	Halee Allison	✓	<button>Einsehen</button>
#3	21.04.2021 bis 24.04.2021	admin	✓	<button>Einsehen</button>
#4	29.04.2021 bis 29.04.2021	admin	✓	<button>Einsehen</button>
#6	30.04.2021 bis 30.04.2021	admin	✓	<button>Einsehen</button>
#7	22.04.2021 bis 22.04.2021	admin	✓	<button>Einsehen</button>

Antragsadministrations Übersichtsseite aus der Sicht eines Vorgesetzten

A.19 Ist-Analyse – Word Vorlage

ATRIVIO
media network

Urlaubsantrag

Antragsteller(in):
Name:

Ich beantrage Urlaub:

von	Uhrzeit (nur bei ½ Tagen)	bis	Uhrzeit (nur bei ½ Tagen)	Art	Anzahl Tage *
17.9.18		21.9.18		Urlaub	5

* Bei mehr als drei Tagen Urlaub ist ein Termin zur Urlaubsübergabe festzusetzen!

Übergabetermin am: _____ mit: _____
(mind. 2 Tage vor Antritt des Urlaubes)

Vertretung:
durch:

Kempten, _____ Datum _____ abgesprochen _____ Unterschrift Antragsteller

Urlaub genehmigt ja nein

Grund der Ablehnung: _____

Kempten, _____ Datum _____ Unterschrift Vorgesetzter

Buchungsvermerk: _____

Outlook:

A.20 Antrags Administration – Codeauszug

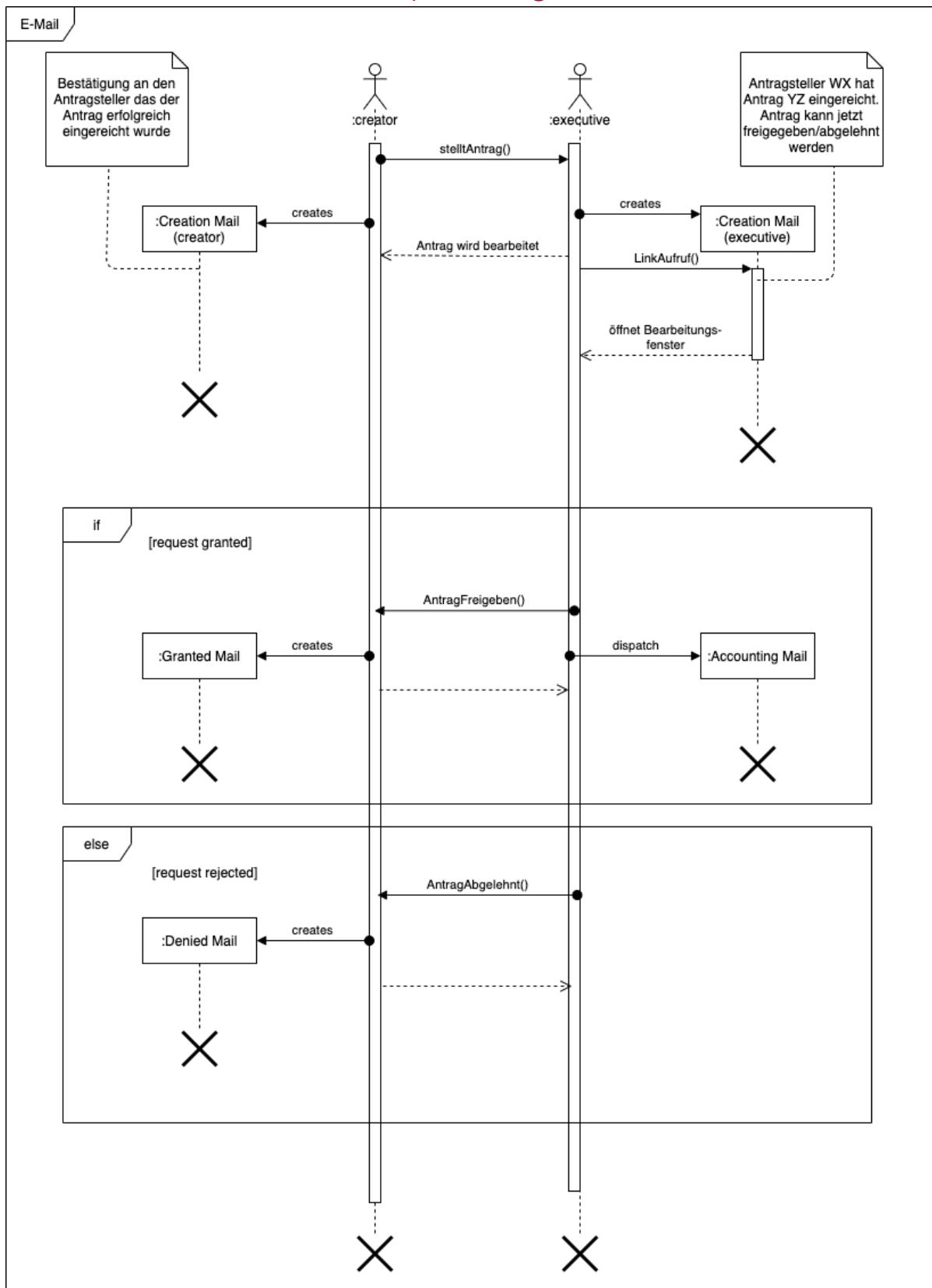
```

1  <?php
2
3  namespace App\Http\Controllers\Request;
4
5  use App\Http\Controllers\Controller;
6  use App\Http\Controllers\Request\RequestDetail;
7  use Illuminate\Http\Request;
8
9  use App\Models\Request as RequestModel;
10 use App\Models\Comment as CommentModel;
11
12 use App\Traits\ResolveTimestampTrait;
13
14 class RequestDecision extends Controller
15 {
16     use ResolveTimestampTrait;
17
18     public function __construct()
19     {
20         $this->middleware('auth');
21     }
22
23     public function index($id)
24     {
25         // Load same collection as detail views.
26         // Valid request->id and usergroup: executive has
27         // to be given.
28         $this->requestDetailClass = new RequestDetail;
29         $this->requestDecisionCollection = $this->requestDetailClass->decision($id);
30
31         $details = $this->requestDecisionCollection;
32
33         return view('request.decision', compact('details'));
34     }
35
36     public function descisionSubmit(request $descision)
37     {
38         if (!is_null($descision->radio) && $descision->radio == "granted")
39             $this->changeRequestToGranted($descision->comment, $descision->id);
40
41         if (!is_null($descision->radio) && $descision->radio == "rejected")
42             $this->changeRequestToRejected($descision->comment, $descision->id);
43
44         return Redirect('/request/assigned');
45     }
46 }
```

(Fortsetzung auf der nächsten Seite)

```
46
47     private function changeRequestToGranted($comment, $request)
48     {
49         $grantedRequestModel = RequestModel::findOrFail($request);
50         $grantedRequestModel->granted = 1;
51         $grantedRequestModel->rejected = 0;
52
53         if (!is_null($comment)) {
54             $commentRelation = $this->buildCommentRelation($comment);
55             $commentRelation->comment_granted()->save($grantedRequestModel);
56         } else {
57             $grantedRequestModel->save();
58         }
59     }
60
61     private function changeRequestToRejected($comment, $request)
62     {
63         $rejectRequestModel = RequestModel::findOrFail($request);
64         $rejectRequestModel->granted = 0;
65         $rejectRequestModel->rejected = 1;
66
67         if (!is_null($comment)) {
68             $commentRelation = $this->buildCommentRelation($comment);
69             $commentRelation->comment_rejected()->save($rejectRequestModel);
70         } else {
71             $rejectRequestModel->save();
72         }
73     }
74
75     protected function buildCommentRelation($comment)
76     {
77         $commentRelationModel = new CommentModel;
78         $commentRelationModel->comment = $comment;
79         $commentRelationModel->save();
80         return $commentRelationModel;
81     }
82 }
83 }
```

A.21 E-Mail Notification – Sequenzdiagramm



A.22 Mailing - Codeauszug

```

1  <?php
2
3  namespace App\Mail\Request;
4
5  use Illuminate\Bus\Queueable;
6  use Illuminate\Contracts\Queue\ShouldQueue;
7  use Illuminate\Mail\Mailable;
8  use Illuminate\Queue\SerializesModels;
9
10 class CreationMailCreator extends Mailable
11 {
12     use Queueable, SerializesModels;
13
14     public $recipient;
15     public $requestId;
16     public $detailLink;
17
18     /**
19      * Create a new message instance.
20      *
21      * @return void
22      */
23     public function __construct($name, $requestId)
24     {
25         $this->recipient = $name;
26         $this->requestId = $requestId;
27         $this->detailLink = 'http://ulla.atrивio.net/request/' . $requestId . '/detail';
28     }
29
30     /**
31      * Build the message.
32      *
33      * @return $this
34      */
35     public function build()
36     {
37         return $this->subject("Bald geht's in Urlaub! 🎉")
38             ->markdown('request-mails.creator.creation');
39     }
40 }
```

```

1  @component('mail::message')
2  # Hallo, {{$recipient}}!
3
4  Dein Urlaubsantrag mit der Id: #{{$requestId}} wurde eingereicht.
5
6  @component('mail::button', ['url' => $detailLink])
7  Antrag ansehen 📧
8  @endcomponent
9
10 Der von dir ausgewählte Vorgesetzte wurde informiert.
11 @endcomponent
```

A.23 Enlightn Test – CLI Auszug

Report Card

=====

Status	Performance	Reliability	Security	Total
Passed	12 (67%)	24 (92%)	14 (70%)	50 (78%)
Failed	3 (17%)	2 (8%)	4 (20%)	9 (14%)
Not Applicable	3 (17%)	0 (0%)	2 (10%)	5 (8%)
Error	0 (0%)	0 (0%)	0 (0%)	0 (0%)

A.24 Soll-/Ist-Vergleich

Phase	Soll (Geplant)	Ist (Tatsächlich)	+/- (Differenz)
Analyse	3.5h	2.5h	- 1.0h
Entwurf	8.0h	6.0h	- 1.0h
Entwicklung und Implementierung	42.5h	44.5h	+ 2.0h
Abnahme und Deployment	3.0h	2.0h	- 1.0h
Dokumentation	13.0h	14.0h	+ 1.0h
Gesamt	70h	70h	

Tabelle VII - Soll/Ist Vergleich

A.25 Entwicklerdokumentation – Auszug



Urlaubsverwaltungs- & Antragsstellungsapplikation (kurz: ULLA)

- [Setup](#)
- [Funktion](#)

Setup

Zunächst muss das Repository geklont werden.

Repository Origin:

```
git curl https://github.com/mxmueler/ULLA.git
cd laravel
```

Beispiel vhost Konfiguration (falls nötig):

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/ULLA/laravel/public

    <Directory "/var/www/ULLA/laravel/public">
        AllowOverride All
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

(Verzeichnisstruktur nur Beispielhaft).

(Fortsetzung auf der nächsten Seite)

Composer installieren: Composer kümmert sich darum, diese ganzen Abhängigkeiten aufzulösen und alle benötigten Bibliotheken automatisch in der richtigen Version in unser Projekt herunterzuladen. Sollte Composer bereits installiert sein kann dieser Schritt übersprungen werden.

```
composer install
```

Laratrust installieren: Laratrust ist ein Laravel (>=5.2)-Paket, mit dem sehr einfach alles, was mit Autorisierung (Rollen und Berechtigungen) zu tun hat, innerhalb einer Laravel Anwendung hangehabt werden kann. Elementar für die Applikation.

```
composer require santigarcor/laratrust
php artisan vendor:publish --tag="laratrust"

composer dump-autoload
php artisan migrate
```

Laravel UI installieren:

```
composer require laravel/ui
```

Frontend aktualisieren:

```
npm install
npm run dev
```

Seeds:

```
php artisan db:seed --class=ullaDefaultRoleSeeder
php artisan db:seed --class=ullaDefaultAdminSeed
php artisan db:seed --class=ullaDefaultRequestTypeSeeder
```

Der Admin Seed erstellt den Initialen Admin User.

Default Admin Zugangsdaten:

E-Mail	Passwort
admin@admin.com	password

Nach der einrichtung sollte der User gelöscht werden.

Mail Setup:

```
php artisan vendor:publish --tag=laravel-mail
composer require spatie/calendar-links
```

Wichtig! Beim dem Initialen Aufbau muss app/json Ordner nach app/storage/app kopiert werden