Graphical user interface

Description automatically generated

**ULLA**

Firmeninternes Management Interface zur Urlaubsverwaltung & Antragserstellung als Webapplikation zum Zwecke der Digitalisierung

*Datum: 23.04.2021*

Dokumentation zur Betrieblichen Projektarbeit

**Prüfungsbewerber:**

Maximilian Müller

**Prüf-Nr.:**

4087

Inhalt

[Vorwort 3](#_Toc70085519)

[Glossar 4](#_Toc70085520)

[Technisches Abkürzungsverzeichnis 6](#_Toc70085521)

[1 Projektbeschreibung 7](#_Toc70085522)

[1.1 Projektbezeichnung 7](#_Toc70085523)

[1.2 Projektumfeld 7](#_Toc70085524)

[1.3 Projektziel 7](#_Toc70085525)

[1.4 Projektbegründung 7](#_Toc70085526)

[1.5 Projektschnittstellen 8](#_Toc70085527)

[1.6 Projektabgrenzung 8](#_Toc70085528)

[2 Projektplanung 8](#_Toc70085529)

[2.1 Projektphasen 8](#_Toc70085530)

[2.2 Abweichungen vom Projektantrag 8](#_Toc70085531)

[2.3 Ressourcenplanung 8](#_Toc70085532)

[2.4 Entwicklungsprozess 8](#_Toc70085533)

[3 Analysephase 9](#_Toc70085534)

[3.1 Ist-Analyse 9](#_Toc70085535)

[3.2 Wirtschaftlichkeitsanalyse 9](#_Toc70085536)

[3.2.1 „Make or buy“ Entscheidung 9](#_Toc70085537)

[3.2.2 Projektkosten 9](#_Toc70085538)

[3.2.3 Amortisationsdauer 10](#_Toc70085539)

[3.3 Anwendungsfälle 10](#_Toc70085540)

[3.4 Lastenheft 10](#_Toc70085541)

[4 Entwurfsphase 10](#_Toc70085542)

[4.1 Zielplattform 10](#_Toc70085543)

[4.2 Server 11](#_Toc70085544)

[4.2.1 Spezifikationen 11](#_Toc70085545)

[4.3 Architekturdesign 11](#_Toc70085546)

[4.4 Benutzeroberfläche 11](#_Toc70085547)

[4.5 Datenmodell 12](#_Toc70085548)

[4.6 Geschäftslogik 12](#_Toc70085549)

[5 Implementierungsphase 13](#_Toc70085550)

[5.1 Implementierung des Datenmodells 13](#_Toc70085551)

[5.1.1 Exkurs: Migration 13](#_Toc70085552)

[5.1.2 Exkurs: Eloquent 13](#_Toc70085553)

[5.2 Implementierung der Benutzeroberfläche 13](#_Toc70085554)

[5.2.1 Exkurs: Blade 13](#_Toc70085555)

[5.3 Implementierung der Routen 14](#_Toc70085556)

[5.3.1 Exkurs: Routing 14](#_Toc70085557)

[Literaturverzeichnis 15](#_Toc70085558)

[A Anhang 16](#_Toc70085559)

[A.1 Detaillierte Zeitplanung 16](#_Toc70085560)

[A.2 Ressourcenplan 17](#_Toc70085561)

[A.3 Ist-Analyse - Aktivitätsdiagramm 18](#_Toc70085562)

[A.4 Kalkulatorische Aufstellung 18](#_Toc70085563)

[A.5 Zeiteinsparung 18](#_Toc70085564)

[A.6 Anwendungsfälle – Use Case Diagramm 19](#_Toc70085565)

[A.7 Auszug Lastenheft 20](#_Toc70085566)

[A.8 Mockup - Formular 21](#_Toc70085567)

[A.9 Mockup - Dashboard 21](#_Toc70085568)

[A.10 Entity Relationship Modell 22](#_Toc70085569)

[A.11 Request Eloquent Model - Codeauszug 23](#_Toc70085570)

[A.12 Request Migration - Codeauszug 24](#_Toc70085571)

[A.13 Request - Klassendiagramm 25](#_Toc70085572)

[A.14 Composer Spezifikation – Codeauszug 26](#_Toc70085573)

[A.15 Composer Spezifikation – Erklärung 27](#_Toc70085574)

[A.16 Routen – Codeauszug 28](#_Toc70085575)

[A.17 Blade – Codeauszug 28](#_Toc70085576)

[A.18 Screenshots der Applikation 29](#_Toc70085577)

# Vorwort

# Glossar

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

**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](#_Technisches_Abkürzungsverzeichnis)-Muster folgt. Es wurde 2011 von Taylor Otwell initiier.

**Blade** - Blade ist die standardmäßig mit Laravel mitgelieferte Template Engine. Mit Blade lassen sich Views in einer Laravel Anwendung implemetieren. 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.

# Technisches Abkürzungsverzeichnis

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

**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

# Projektbeschreibung

## Projektbezeichnung

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

## 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*](#_Glossar) Lösungen, E-Commerce-Plattformen bis hin zur Cloud Infrastruktur angeboten, aber auch eigene Produkte wie [*Mail2Many*](#_Glossar)oder [*Geneva*](#_Glossar)betriebseigen entwickelt und vertrieben.

## 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 Druckkosten- und material reduziert. Durch das neue System wird eine strikte [*Compliance*](#_Glossar) und Ordnung in den Ablauf der Antragserstellung eingebracht.

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

## 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, befüllt 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](#_Projektumfeld) erwähnt, vermehrt Unternehmensressourcen wie Büromaterialien (Druckermittel, Papier, Ordner) oder Stellplatze (Regale) verbraucht.

Ein Organisatorischen Makel stellt die fehlende Zentralisierung der Anträge da. Momentan besteht keine Möglichkeit Anträge Abteilungsübergreifend zu vergleichen u. o. abzustimmen.

Mit der Umsetzung von [*ULLA*](#_Glossar)werden die organisatorischen und wirtschaftlichen Makel beseitigt und eine zentrale, zukunftssichere, Kosten und Ressourcensparende Anlaufstelle geschaffen.

## 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 Firmennetzt 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.

## Projektabgrenzung

[*ULLA*](#_Glossar) 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 Drittanbieter.

# Projektplanung

## Projektphasen

Das Projekt wurde innerhalb den, von der IHK Schwaben, vorgegebenen 70 Stunden durchgeführt. Eine grobe sowie eine detaillierte Zeitplanung finden sich im [Anhang A.1](#_A.1__Detaillierte).

## Abweichungen vom Projektantrag

Zum Zeitpunkt der Abgabe gab es keine Abweichung vom Projektantrag.

## Ressourcenplanung

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

## 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 und 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.

# Analysephase

## Ist-Analyse

Wie bereits unter [1.4 (Projektbegründung)](#_Projektbegründung) beschrieben stellt der aktuelle Ablauf 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 eine/en Mitarbeiter:in (Benutzergruppe: Mitarbeiter:in, 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/die Antragsteller:in bzw. Mitarbeiter:in mündlich über die Freigabe. Der Unterschriebene Antrag wird dann an die Verwaltung übergeben. Ein/e Verwaltungsmitarbeiter:in vermerkt die genommen Urlaubstage in der Personalverwaltung und archiviert den Antrag, außerdem wird der/dem Antragsteller:in bzw. Mitarbeiter:in eine Kopie zur schriftlichen Bestätigung und Absicherung vorgelegt.

Im [Anhang A.3](#_A.3__Ist-Analyse) ist ein Aktivitätsdiagramm abgebildet, in dem dieser Prozess dargestellt wird.

## Wirtschaftlichkeitsanalyse

### „Make or buy“ Entscheidung

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

### Projektkosten

In der Kalkulation wurden alle anfallenden Entwicklungskosten von [ULLA](#_Glossar) im Rahmen des Umsetzungsrahmen von 70 Stunden aufgefasst. 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](#_A.4__Kalkulatorische) eingesehen werden.

### 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](#_A.5__Zeiteinsparung) Zeiteinsparungsrechnung und in [Anhang A.4](#_A.4__Kalkulatorische) entstandenen Entwicklungskosten zu verstehen.

## Anwendungsfälle

Die Anwendungsfälle von ULLA wurden zur genaueren Darstellung in einem Anwendungsfalldiagramm zusammengefast.   
Das Anwendungsfalldiagramm (Use Case Diagramm) stellt das Zusammenspiel der Anwendungsfälle eines Systems untereinander und mit den Akteuren dar[[1]](#footnote-1). 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 UML[[2]](#footnote-2) 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](#_A.6__Anwendungsfälle) zu finden.

## Lastenheft

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

# Entwurfsphase

## Zielplattform

Wie unter [Punkt 1.1](#_Projektbezeichnung) bereits beschrieben soll [*ULLA*](#_Glossar) 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 stehts 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](#_Technisches_Abkürzungsverzeichnis)-Framework [*Laravel*](#_Glossar) eingesetzt. Die Auswahl ergibt sich zum einen durch die von [Atrivio](#_Projektumfeld) vorgeschriebenen Architekturregeln, zum anderen eignet sich die Framework Architektur durch die Bindung an das [MVC](#_Technisches_Abkürzungsverzeichnis) Musters und [*Blade*](#_Glossar) Templating für die vorgesehen Anforderungen.

## Server

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

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

## Architekturdesign

Als Architekturbasis bzw. Pattern soll das in [Punkt 4.1](#_Zielplattform) bereits beschriebene [MVC](#_Technisches_Abkürzungsverzeichnis) 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](#_Technisches_Abkürzungsverzeichnis) 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). Müsst beispielsweise Datenbankstrukturen geändert werden, z.B. der Spaltenname der E-Mail-Adresse des Users wird umbenannt, so müsst diese Änderung nur in der Model-Klasse für den User geändert werden. Die Anwendungslogik (Controller) und Darstellungsschicht (View) bleibt davon unberührt.

## Benutzeroberfläche

Wie in [Punkt 2.4](#_Entwicklungsprozess) 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 Mockup‘s bzw. Prototypen können in [Anhang A.8](#_A.8__) und [Anhang A.9](#_A.9__Mockup) 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:innen an Desktopgeräten arbeiten besteht im ersten Schritt keine konkrete Anforderung an eine optimierte Responsivität der Applikation.

## Datenmodell

Das Datenmodell wurde mit Hilfe von Draw.IO[[3]](#footnote-3) als Entity-Relation-Modell (kurz. [ERM](#_Technisches_Abkürzungsverzeichnis)) entworfen. Um Übersichtlich- und Verständlichkeit zu wahren befindet sich das Diagramm ohne Attribute der Entitäten im [Anhang A.10](#_A.10_Entity_Relationship).  
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 Vorgesetzen 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\_tstmp 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 kommentiert werden.  
Grundsätzlich stellt die Auslagerung eine 1:1 Beziehung 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.

## Geschäftslogik

Aufgrund der Testgetriebenen Entwicklung war es nicht möglich, vor Beginn der Umsetzungs- bzw. Implementierungsphase der Anwendung ein Klassendiagram zur technischen Darstellung der Geschäftslogik zu erstellen. Um den Dokumentationsprozess dennoch einzuhalten wurde nachträglich ein Klassendiagram aus dem Programmcode nach der [Reverse Engineering](#_Glossar) Technik generiert. Das generierte Klassendiagram (Auszug) befindet sich in [Anhang A.13](#_A.13_Request_-) und bildet das Request abhängige Klassenkonstrukt mit Controlleranbindung dar.

# Implementierungsphase

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

## Implementierung des Datenmodells

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

### 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](#_Technisches_Abkürzungsverzeichnis) Parameter (Datentyp, Größe, ect.) und Kardinalitäten (Foreign key’s ect.) der Attribute festgelegt. Eine Beispielmigration der in [Punkt 4.5](#_Datenmodell) erwähnten request Entität findet sich in [Anhang A.12](#_A.12_Codeauszug_–).

### Exkurs: Eloquent

Beim Thema Datenmodell (Nachtrag zu [Punkt 4.5](#_Datenmodell)) muss im Projektumfeld das Thema Eloquent erwähnt und erläutert werden. Eloquent ist, wie die Migrationsarchitektur bereits in [*Laravel*](#_Glossar) integriert und eines der wichtigsten Features des Frameworks. Eloquent [ORM](#_Technisches_Abkürzungsverzeichnis) stellt Inhalte von relationalen Datenbanken als Klassen und Objekte in [PHP](#_Technisches_Abkürzungsverzeichnis) 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](#_A11._Codeauszug_–).

## Implementierung der Benutzeroberfläche

Grundsätzlich baut sich die Benutzeroberfläche der Applikation in [*Blade*](#_Glossar_1) Templates auf.

### Exkurs: Blade

Eine weiter elementare Funktion des [*Laravel*](#_Glossar_1)-Frameworks ist die Blade-Templating-Engine. Klare Vorteile der Engine ist Ihre Freizügigkeit durch die direkte Verwendung bzw. Interpretation von [PHP](#_Technisches_Abkürzungsverzeichnis)-Code und Verarbeiten von übergebenen Parametern. Außerdem können ges. Templates oder Teilabschnitte geschachtelt werden. Das [*Blade*](#_Glossar_1)-Template des Dashboards kann in [Anlage A.17](#_A.17_Blade_–) als Beispiel begutachtet werden.

Die gerenderten [*Blade*](#_Glossar_1)-Templates werden dann als reguläre [HTML](#_Technisches_Abkürzungsverzeichnis) Dateien ausgegeben und vom Webserver interpretiert. Die Gestaltung der Oberfläche wurde mit eingebundenen [CSS](#_Technisches_Abkürzungsverzeichnis)-Dateien umgesetzt welche wiederrum [*SCSS*](#_Glossar_1) Basiert mithilfe von [*node.js*](#_Glossar_1) im Verbund mit dem Frontend-Framework [*Bootstrap*](#_Glossar_1) Serverseitig gerendert worden. Im Anhang A.14 befinden sich Screenshots der Applikation, die nach der Implementierungsphase entstanden sind.

## Implementierung der Routen

Um die [*Blade*](#_Glossar_1)-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*](#_Glossar_1)-Templates zeigen.

### Exkurs: Routing

Das [*Laravel*](#_Glossar) Routing entscheidet, welche Aktion bei welcher URL ausgeführt wird. So kann im einfachsten Fall (wie im Vorwort erwähnt) eine View anzeigt 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](#_A.16_Routen_–).

# 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

# A 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 | 1.5h |
| 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 | 4.0h |
| Code-Review mit dem Ausbilder u. ggf Fachabteilung(en) | 1.0h |
| Abnahme durch einen Ausgewählten Fachbereich | 1.0h |
| Deployment der Applikation | 2.0h |
| 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** |

## 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:in der Verwaltung |
|  | Mitarbeiter:in Testing bzw. Projektleitung |
|  | Anwendungsentwickler, Prüfling |

## Diagram Description automatically generatedA.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€ |

## 

## A.5 Zeiteinsparung

|  |  |  |  |
| --- | --- | --- | --- |
| Stelle | Kosten in € pro Stunde | Ersparnis in Minuten | Gesamt |
| Mitarbeiter | 85 \* | 10 | 14,17€ |
| Vorgesetzer | 85 \* | 5 | 7.08€ |
| Verwaltung | 85 \* | 7 | 9.92€ |
| Ersparnis pro Antrag | | | 31,17€ |

\* Durchschnittswerte

Berechnung:

## A.6 Anwendungsfälle – Use Case Diagramm

Diagram

Description automatically generated

## A.7 Auszug Lastenheft



## A.8 Mockup - Formular

Graphical user interface

Description automatically generated

## A.9 Mockup - Dashboard

Diagram

Description automatically generated with low confidence

## A.10 Entity Relationship Modell

Diagram

Description automatically generated

## A.11 Request Eloquent Model - Codeauszug

A picture containing table

Description automatically generated

## A.12 Request Migration - Codeauszug

Text

Description automatically generated with low confidence

## A.13 Request - Klassendiagramm

Diagram, schematic

Description automatically generated

## A.14 Composer Spezifikation – Codeauszug

A picture containing text

Description automatically generated

In [Anhang A.15](#_A.15_Composer_Specifikation)  
erläutert.

## A.15 Composer Spezifikation – Erklärung

In der folgenden Erklärung werden die im Projekt verwendeten und in der [*composer.json*](#_Glossar) aufgerührten *[packagist](#_Glossar)* Packete vorgestellt.

|  |  |  |  |
| --- | --- | --- | --- |
| Package Name | Erklärung | Basis: | Erweitert[[4]](#footnote-4): |
| "php" | Definiert die im Projekt vorgesehen [PHP](#_Technisches_Abkürzungsverzeichnis) 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](#_Technisches_Abkürzungsverzeichnis) 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. |  | ✓ |

## A.16 Routen – Codeauszug

Text

Description automatically generated

## 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.

Text

Description automatically generated

## A.18 Screenshots der Applikation

(Alle Screenshots sind nach der Implementierungsphase entstanden.)

Graphical user interface, application

Description automatically generated  
*Loginmaske der Applikation*

Graphical user interface, application

Description automatically generated  
*Dashboard der Applikation aus der Sicht eines Mitarbeiters*

Graphical user interface, application

Description automatically generated  
*Antrags Erstellungsmaske aus der Sicht eines Mitarbeiters*

  
*Antrags Administrationsmaske aus der Sicht eines Vorgesetzen*  
Graphical user interface, table

Description automatically generated

*Antragsadministrations Übersichtsseite aus der Sicht eines Vorgesetzen*

1. Vgl. [Literaturverzeichnis](#_Literaturverzeichnis): *Objektorientierte Systemanalyse*, S. 59 [↑](#footnote-ref-1)
2. Vg. [Literaturverzeichnis](#_Literaturverzeichnis): *Objektorientierte Systemanalyse*, S. 58 [↑](#footnote-ref-2)
3. Kostenlose Diagramm WebApp aus dem Atlassian Ökosystem [↑](#footnote-ref-3)
4. Erweiterungen die vom Autor für das Projekt installiert wurden. [↑](#footnote-ref-4)