Abschlussprüfung Sommer 2018

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

"Single-Sign-On"-Authentifizierungsserver mit Beispielimplementierung eines Clients

Abgabetermin: Berlin, den 12.04.2018

Prüfungsbewerber:
Sibylle Blümke
Scharnweberstr. 6
10247 Berlin

wycomco.

Ausbildungsbetrieb:
wycomco GmbH
Fasanenstr. 35
10719 Berlin

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

AD	Active Directory
CSS	Cascading Style Sheets
CSRF	Cross-Site-Request-Forgery
GUI	Graphical User Interface
HTML	Hypertext Markup Language
JSON	Javascript Object Notation
MVC	Model-View-Controller
ORM	Object Relational Mapping
PHP	PHP: Hypertext Preprocessor
REST	Representational State Transfer
SAML	Security Assertion Markup Language
SSO	Single Sign On
TDD	Test Driven Development
UML	Unified Modeling Language
XML	Extensible Markup Language

1 Einleitung

Die folgende Projektdokumentation wurde im Rahmen des IHK Abschlussprojektes erstellt, welches während der Ausbildung zum Fachinformatiker Fachrichtung Anwendungsentwicklung durchgeführt wurde.

1.1 Projektumfeld

Ausbildungsbetrieb ist die *wycomco GmbH*, im Folgenden als *wycomco* bezeichnet. Wycomco ist ein Full-Service-Dienstleister im IT-Bereich mit Sitz in Berlin. Zu den Produkten des Unternehmens gehören außerdem individuell anpassbare Softwarelösungen für Anforderungen aller Art. Momentan beschäftigt das Unternehmen 12 Mitarbeiter.
Der Auftraggeber des Projektes ist die Entwicklungsabteilung von wycomco.

1.2 Projektziel

Wycomco implementiert immer mehr Webanwendungen zur internen Nutzung. Bisher müssen sich die Mitarbeiter an jeder Anwendung mit eigenem - teils verschiedenen - Benutzernamen und Passwort registrieren und anschließend anmelden. Rechte und Rollen werden manuell gesetzt, eine Anbindung an die Active Directory (AD) ist nicht vorhanden. Mit einem Single Sign On (SSO)-Server soll dieser Prozess automatisiert und vereinfacht werden. Die eigenständige Applikation, im folgenden *wy-connect* genannt, verwaltet die Benutzer an zentraler Stelle und vereinheitlicht die Nutzerdaten der verschiedenen Anwendungen. Die Anmeldung erfolgt ausschließlich am SSO über einen sicheren Kanal.

1.3 Projektbegründung

Im Unternehmen wird die Softwareentwicklung vorangetrieben und auch die Entwicklung im Eigenbedarf steigt stetig an. So existieren momentan drei Anwendungen die im Betrieb laufen oder in der Testphase sind. Alle drei Apps haben eine eigene Benutzerverwaltung mit eigenen Benutzerkennungen. Wird bei der ersten ein Nutzername zum Anmelden vergeben, nutzt die zweite die Firmenmailadresse oder eine ID. An allen muss in regelmäßigen Abständen das Passwort geändert werden. Damit gehört es zum Standard sich Passwörter aufzuschreiben, Passwörter generieren zu lassen oder vergessene Passwörter neu zu vergeben. Der Einfachheit halber neigt man dazu Trivialpasswörter zu nutzen, diese wiederholt einzusetzen und sie sich an unsicheren Stellen zu notieren[1]. Durch den wiederholten Anmeldeprozess an verschiedenen Diensten mit verschiedenen Sicherheitsvorkehrungen steigt zudem die Wahrscheinlichkeit, dass ein Passwort ausgespäht wird. Dazu ein Anwendungsdiagramm im Anhang A.4: Use-Case-Diagramm ohne SSO auf Seite iv.
Durch die Implementierung einer Single Sign-on-Lösung muss sich der Nutzer nur ein Master-Passwort merken und dieses einmalig bei der Anmeldung am wy-connect Dienst eingeben. Bei der Nutzung von SSO ist der größte Vorteil darin zu sehen, dass sich der Nutzer nicht nochmal registrieren muss, so dass das lästige Eintippen von Daten, Festlegen eines neuen Passwortes und Bestätigung der Registrierung entfällt.

[1] vgl. INTERSOFT CONSULTING SERVICES AG [2016]

Auch der administrative Aufwand verringert sich bei einer zentralen Nutzerverwaltung[2].
Dies hätte auf jeden Fall eine bessere Usability und eine nicht zu unterschätzende Zeitersparnis für Anwender im Vergleich zur derzeitigen Lösung zur Folge. Aufgrund der angeführten Gründe hat sich wycomco entschieden die Entwicklung von wy-connect in Auftrag zu geben.

1.4 Projektschnittstellen

Die Anwendung benötigt keinerlei Schnittstellen zu anderen Diensten. Der SSO-Server kommuniziert im Rahmen des Abschlussprojektes nur mit einem Testclient, der eigens implementiert wird. Integration mit anderen externen Systemen wie dem AD ist nicht Teil der Anforderung.

1.5 Projektabgrenzung

Die Anbindung an einen Client wird in diesem Projekt nur für einen Testclient durchgeführt, die Durchführung für alle bestehenden Anwendungen im Unternehmen gehört nicht zum Projekt.

2 Projektplanung

2.1 Projektphasen

Für die Umsetzung standen 70 Stunden zur Verfügung. Diese wurden vor Projektbeginn auf die verschiedenen Phasen des Entwicklungsprozesses aufgeteilt. Das Ergebnis der groben Zeitplanung lässt sich dem folgendem gestapelten Balkendiagramm entnehmen. Eine detaillierte Übersicht der Phasen befindet sich

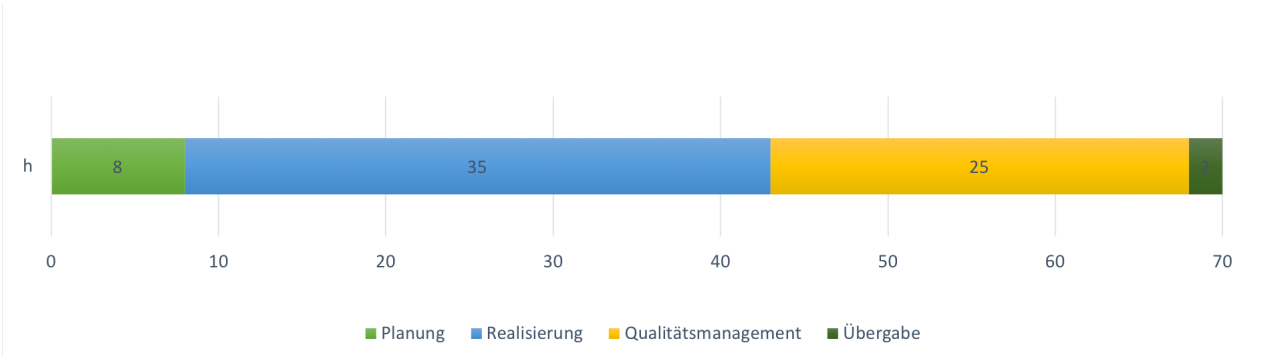

Abbildung 1: grobe Zeitplanung

im Anhang A.1: Detaillierte Zeitplanung auf Seite i.

[2] vgl. GmbH [2016]

2.2 Ressourcenplanung

Eine vollständige Auflistung aller während der Umsetzung des Projekts verwendeten Ressourcen befindet sich im Anhang A.2: Verwendete Ressourcen auf Seite i. Bei der Auswahl der verwendeten Software war es wichtig, dass hierdurch keine Zusatzkosten anfallen. Es sollte also Software verwendet werden, die entweder kostenfrei ist (z. B. Open Source) oder für die wycomco bereits Lizenzen besitzt.

2.3 Entwicklungsprozess

Die Vorgehensweise nach dem das Projekt entwickelt wird, nennt sich Entwicklungsprozess. In diesem Fall handelt es sich um das Wasserfallmodell. Bei diesem nicht iterativen, linearen Prozess werden die einzelnen Projektphasen schrittweise bearbeitet. Hierbei bilden die Phasen-Ergebnisse jeweils die bindende Vorgabe für die nächste Projektphase[3]. Das gesamte Projekt soll mithilfe von Test Driven Development (TDD), zu Deutsch *Testgetriebene Entwicklung*, umgesetzt werden. Das Kernprinzip von TDD besagt, dass das Testen der Programmkomponenten den kompletten Entwicklungsprozess leitet. Es handelt sich hierbei um eine Designstrategie in der das Testen vor der eigentlichen Implementierung stattfindet. Es soll keine Zeile Produktivcode geschrieben werden, die nicht durch einen Test vorher abgedeckt wird. Damit lässt sich die Qualität des Codes erhöhen und den späteren Wartungsaufwand im Nachhinein zu verringern[4]. Dieses Parallelentwicklung von Code und Tests erfolgt in sich wiederholenden Mikroiterationen, die nur einen kleinen Zeitraum in Anspruch nehmen sollte. Man kann TDD in drei Hauptteile aufspalten, die im englisch Red-Green-Refactor genannt werden. Die einzelnen Phasen lassen sich wie folgt beschreiben[5] In der ersten, der roten Phase wird ein Test geschrieben, der fehlschlägt. Dieser wird dann mit einem Minimum an Code in der grünen Phase soweit verändert, dass er erfolgreich durchläuft. In der letzten grauen Phase wird der Code refaktorisiert, dass einfacher, sauberer Quelltext entsteht.

Zur Verdeutlichung der drei Phasen die Abbildung 2[6]

3 Analysephase

3.1 Ist-Analyse

Derzeit gibt es bei der Anmeldung an den verschiedenen Diensten bei wycomco keine Optimierungen. Bei allen Anwendungen kann man sich nicht selbständig registrieren und sich damit ein Benutzerkonto anlegen. Der jeweilige Administrator oder Befugte muss den User per Hand per Email einladen, damit dieser die Registrierung durchführen kann und nach einem Aktivierungslink die Applikation vollständig nutzbar ist.

Dieser Prozess lässt wie folgt beschreiben:

1. Anlegen eines vorläufigen Benutzerkontos durch den Admin

[3] vgl. [Hübscher u. a. 2007, S. 263]
[4] vgl. intersoft consulting services AG [2016]
[5] vgl. Wikipedia [2018]
[6] vgl. EETech Media [2014]

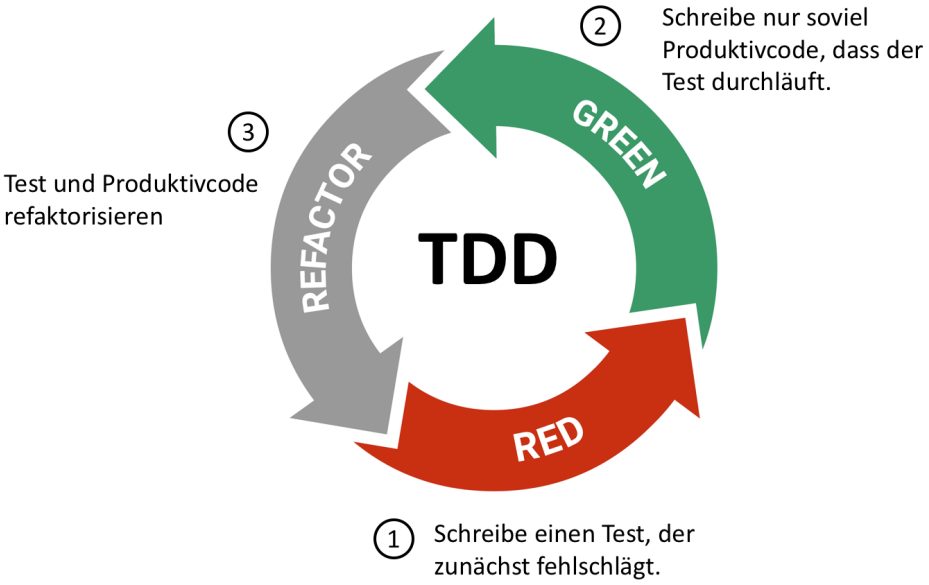

Abbildung 2: TDD Zirkel

2. Benutzer bekommt eine Mail und klickt auf den Registrierungslink
3. Benutzer füllt das Registrierungsformular aus
4. Benutzer bekommt eine Email mit einem Aktivierungslink
5. Benutzer klickt auf den Link und lässt sich damit aktivieren
6. Benutzer muss sich einloggen und kann damit die Anwendung vollständig nutzen

Wie schnell ersichtlich wird, ist der Aufwand immens, wenn dieser Vorgang auch nur dreimal wiederholt werden muss. Dieser Prozess ist sehr zeitaufwändig. Auch im täglichen Geschäft ist es erforderlich sich an jeder Anwendung einzeln anzumelden. Wie in Abschnitt 1.3 schon erwähnt, führt dies schnell zu Trivialpasswörtern und damit zu einem Sicherheitsrisiko.

3.2 Wirtschaftlichkeitsanalyse

Durch den momentanen Prozess entsteht ein hoher zeitlicher Mehraufwand, der durch die Umsetzung des Projektes verringert werden kann. Eine Vereinheitlichung der Nutzerzugänge durch ein Single-Sign-On-Verfahren erscheint als hilfreicher Ausweg. Damit ist es dringend nötig dieses Projekt umzusetzen. Ob sich das auch wirtschaftlich begründen lässt, wird in den nächsten Abschnitten erläutert.

3.2.1 „Make or Buy“-Entscheidung

Zu dieser Problematik gibt es eine Vielzahl von Anbietern auf dem Markt und es kommen fortlaufend Neue hinzu. Grundlegend für ein SSO-Verfahren ist dessen Integrierbarkeit, ein Großteil der genutzten Anwendungen sollte unterstützt werden und dabei sollten auch Vorgaben für komplexe Passwörter und verschlüsselte Anmeldeverfahren Standard sein. Würde ein Unbefugter Zugang erhalten, hätte er in der Regel Zugriff auf alle angebundenen Anwendungen. Auch durch ihre Nutzerfreundlichkeit sollte eine SSO Lösung ansprechen, für Standardanwender gleichermaßen wie für Administratoren[7].

Ein SSO-Verfahren, das tatsächlich alle eingesetzten Anwendungen einbinden kann und die oben genannten für wycomco erfüllt, ist kaum zu finden. Da der Großteil der entwickelten Anwendungen bei wycomco auf Laravel[8] basiert, lag es nahe die zur Verfügung gestellten Pakete zu nutzen. Dabei handelt es sich um Passport[9] und Socialite[10], die die Grundfunktionen eines SSO Servers und den angebundenen Clients bieten. Damit kann das Augenmerk auf die individuellen Anforderungen von wycomco gelegt werden und es wurde sich dazu entschieden, das Projekt in Eigenentwicklung durchzuführen.

! muss das wirklich rein? würde evtl den ganzen Abschnitt makeOrBuy löschen !

3.2.2 Projektkosten

Im Folgenden werden die Projektkosten, die während der Entwicklung anfallen, kalkuliert. Dafür müssen nicht nur die Personalkosten berücksichtigt werden, sondern auch die verwendeten Ressourcen, siehe unter A.2. Sämtliche Werte sind Beispiel-Angaben, da im Rahmen der IHK Projektangaben auf genaue Angaben der Personalkosten verzichtet wird.

Bei den Personalkosten wird zwischen dem Stundensatz eines Auszubildenden und eines Mitarbeiters unterschieden. Der eines Mitarbeiters wird mit 40 € bemessen, der eines Auszubildenden mit 10 €. Für die Nutzung der Ressourcen [11] wird ein Satz von 15 € angewendet. Aus diesen Werten ergeben sich die Projektkosten in Tabelle 1.

Vorgang	Zeit	Kosten pro Stunde	Kosten
Entwicklungskosten	70 h	10 € + 15 € = 25 €	1750,00 €
Fachgespräch	3 h	40 € + 15 € = 55 €	165 €
Code-Review	2 h	40 € + 15 € = 55 €	110,00 €
Abnahme	1 h	40 € + 15 € = 55 €	55 €
		Projektkosten gesamt	**2080,00 €**

Tabelle 1: Kostenaufstellung

[7] vgl. Schonschek [2015]
[8] vgl. Otwell [a]
[9] vgl. Otwell [b]
[10] vgl. Otwell [c]
[11] Hardware, Arbeitsplatz, etc.

3.2.3 Amortisationsdauer

Der Einsatz eines SSO-Servers hat eine deutliche Zeitersparnis zur Folge. Durchschnittlich loggt sich ein Mitarbeiter pro Tag mindestens einmal am Tag in einer Anwendung ein. Bei 3 Anwendungen wie momentan bei wycomco sind es 3 Vorgänge täglich. Dazu muss er alle sechs Monat das Passwort ändern. Durch die Vielzahl an Freelancern in der Firma, schätzt man die Anzahl neuer User pro Monat auf eins. Neue User werden von den Administratoren der Anwendung eingeladen und müssen danach selbstständig ihre Registrierung vervollständigen.

Dazu die Auflistungen in Tabelle 2 und Tabelle 3.

Vorgang	Zeit (alt) pro Vorgang x3	Zeit (neu) pro Vorgang	Einsparung
Admin legt neuen Benutzer an	6 min	2 min	4 min
Benutzer registriert sich	6 min	2 min	4 min
Benutzer loggt sich ein	1,5 min	0,5 min	1 min
Benutzer ändert Passwort	3 min	1 min	2 min
Gesamt			44 min

Tabelle 2: Zeitersparnis pro Vorgang

Vorgang	Anzahl / Monat	Anzahl MA / Monat	Einsparung / Monat
Admin legt neuen Benutzer an	1	1	4 min
Benutzer registriert sich	1	1	4 min
Benutzer loggt sich ein	30	15	450 min
Benutzer ändert Passwort	0,17	15	5 min
Gesamt			463 min

Tabelle 3: Zeitersparnis pro Monat

Für die Zeitersparnis pro Monat ergeben sich damit 463 Minuten.

Dies ergibt eine tägliche Ersparnis von

$$\frac{463 \text{ min/Monat}}{20 \text{ Tage/Monat}} = 23,15 \text{ min/Tag} \tag{1}$$

Bei einer Zeiteinsparung von 23,15 Minuten pro Tag für 252 Arbeitstagen[12] im Jahr ergibt sich eine Zeiteinsparung von

$$252\frac{Tage}{Jahr} \cdot 23,15\frac{min}{Tag} = 5833,8\frac{min}{Jahr} \approx 97,23\frac{h}{Jahr} \tag{2}$$

[12] vgl. WWW.SCHNELLE ONLINE.INFO

Dadurch ergibt sich eine jährliche Einsparung von

$$97,23\ \text{h} \cdot (40 + 15)\,€/\text{h} = 5347{,}65\,€ \tag{3}$$

Die Amortisationszeit beträgt also $\frac{2080{,}00\,€}{5347{,}65\,€/\text{Jahr}} \approx 0,4$ Jahre ≈ 5 Monate.

Der Server muss also mindestens 5 Monate das alte Vorgehen ersetzen, damit sich Anschaffungskosten und Kosteneinsparung ausgleichen. Da es vorgesehen ist die neue Anwendung längerfristig einzusetzen, kann die Umsetzung trotz der relativ langen Amortisationszeit auch unter wirtschaftlichen Gesichtspunkten als sinnvoll eingestuft werden. Eine grafische Darstellung der berechneten Werte findet sich unter A.9 .

3.3 Nutzwertanalyse

Neben den in 3.2.3 (Amortisationsdauer) aufgeführten wirtschaftlichen Vorteilen ergeben sich durch Realisierung des Projekts noch einige zusätzliche Vorteile.

Wie in der 1.3 (Projektbegründung) schon erläutert, bringt der Einsatz einer Single-Sign-On-Lösung eine höhere Sicherheit mit sich. In einer der Anwendungen von wycomco liegen vertrauliche Kundendaten. Sollte das Projekt dort nicht umgesetzt werden und daher Unbefugte Zugriff erlangen, kann man von Opportunitätskosten sprechen. Der Fremdzugriff bringt eine Vertragsstrafe mit sich, schätzungsweise bei größeren Kunden von 60000 . € Das sind Kosten die vermieden werden können mit der Umsetzung von wy-connect.

Obwohl das Projekt im Rahmen von wycomco entwickelt wurde, lässt es sich auch problemlos auf andere Kunden anwenden. Der tatsächliche Nutzen geht also über wycomco hinaus.

3.4 Qualitätsanforderungen

Die Qualitätsanforderungen an die Anwendung lassen sich Tabelle 4 entnehmen.

3.5 Anwendungsfälle

Es wird im Zuge der Analyse des Projektes ein Anwendungsfalldiagramm erstellt. Dies stellt Interaktionen von Benutzern mit dem System dar und zeigt somit das erwartete Verhalten der Anwendung. Das Anwendungsfalldiagramm ist im Anhang A.4 und A.5 dargestellt. Der vollständige Prozess ist in 3.1 (Ist-Analyse) beschrieben. Mit Hilfe von wyconnect kann dieser Prozess wie in den Diagrammen gut zu sehen ist, vereinfacht werden. Administratoren erstellen einmalig eine Einladung und der Benutzer registriert und loggt sich nur einmalig ein und kann sofort alle Anwendungen nutzen. Hierbei sei die Autorisierung des Clients vernachlässigt.

Qualitätsmerkmal	Definition
Abgabetermin einhalten	Der Abgabetermin vom 12.04.2018 ist einzuhalten.
Benutzerfreundlichkeit	Die Anwendung muss über eine GUI intuitiv bedienbar sein.
Flexibilität	Die Anbindung an verschiedene Laravel Anwendungen muss problemlos möglich sein.
Funktionalität	Der Login Prozess über den Server muss reibungslos von Statten gehen.
Zuverlässigkeit	Die Erreichbarkeit des Servers ist zuverlässig und das Deployment sowie die Tests weisen auf Ausnahmen hin.
Wartbarkeit	Um die Wartbarkeit der Anwendung durch die Mitarbeiter von wycomco zu gewährleisten, muss bei der Auswahl der Technologien darauf geachtet werden, dass diese in der Firma vertraut sind (z. B. Auswahl der Programmiersprache).

Tabelle 4: Qualitätsanforderungen

3.6 Lastenheft/Fachkonzept

Am Ende der Entwurfsphase wurde zusammen mit dem Projektleiter auf Basis des Anwendungsfalldiagramms das Lastenheft erstellt. Ein Auszug befindet sich im Anhang Anhang A.3: Lastenheft (Auszug) auf Seite iii

3.7 Zwischenstand

Tabelle 5 zeigt den Zwischenstand nach der Analysephase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Analyse des Ist-Zustands	3 h	2 h	-1 h
2. Soll-Analyse	2 h	2 h	
3. Wirtschaftlichkeitsanalyse	1 h	1 h	
4. Erstellen des Lastenhefts	2 h	2 h	

Tabelle 5: Zwischenstand nach der Analysephase

4 Entwurfsphase

4.1 Zielplattform

Wie in Abschnitt 1.2 (Projektziel) erwähnt, soll am Ende des Abschlussprojektes eine eigenständige Webanwendung vorliegen. Die für eventuelle Weiterentwicklungen eine größtmögliche Plattform-Unabhängigkeit gewährleistet und über einen zeitgemäßen Browser oder auch auf mobilen Endgeräten von jedem Nutzer verwendet werden kann. Eine reine Windows-Anwendung müsste beispielsweise jeweils erst lokal installiert,

und bei Änderungen zusätzlich aktualisiert werden. Die Daten, auf die zugegriffen werden soll, sind in einer bestehenden MySQL Datenbank gespeichert.

Als Programmiersprache wurde PHP: Hypertext Preprocessor (PHP)[13] gewählt. Dies bot sich an, da viele bestehende Anwendungen bereits in PHP geschrieben sind und sie sich damit leichter an wy-connect anbinden lassen und die Entwickler wycomcos mit dieser Sprache vertraut sind. Wie auch die bestehenden Webapplikationen wird bei der Implementation das Framework Laravel[14] genutzt.

4.2 Framework

Laravel ist ein freies PHP-Webframework, welches dem Model-View-Controller (MVC)-Muster folgt. Es ermöglicht neben dem im folgenden unter 4.5 erläuterten MVC-Architekturdesign, Representational State Transfer (REST)-Webdienste zu implementieren. Laravel wird mit dem Object Relational Mapping (ORM) Eloquent und einem gut bedienbaren Migrationssystem ausgeliefert. Damit werden Objekte einer objektorientierten Anwendung in eine relationale Datenbank überführt. Das Framework bringt von Hause aus ein Authentifizierungspaket, was auch von wyconnect genutzt wird und durch die Pakete Socialite und Passport ergänzt wird um den OAuth2 Mechanismus zu implementieren. Dazu mehr im Kapitel 4.8 (Geschäftslogik).

4.3 Wahl des SSO-Protokolls

Nach der Entscheidung einen SSO-Server einzusetzen, galt es ein geeignetes Protokoll für die Umsetzung zu finden. An erster Stelle steht dabei die Sicherheit, bei den wesentlichen Aspekten der Autorisierung und Authentifizierung. Autorisierung um bestimmten Clients den Zugriff auf die Ressourcen des Nutzers zu berechtigen und Authentifizierung um die Identität des Benutzer zu identifizieren. Eine mögliche Lösung bietet das OAuth2 Protokoll. [15].

Die Benutzerkontrolle stellt bei OAuth2 das grundlegende Prinzip dar. Im Fokus steht der Schutz von Benutzerdaten vor unbefugten Zugriff. Die Ressourcen von den Benutzern (als *Resource Owner* bezeichnet) verwaltet ein *Resource Server*, der nur zulässige und autorisierte Anfragen zulässt. Ein vertrauenswürdiger *Authorization Server* authentifiziert den Benutzer und holt dessen Autorisierung für den Zugriff ein. Als Autorisierungsnachweis stellt der *Auhorization Server* Access Token aus. Im vorliegenden Anwendungsfall sind *Resource Server* und *Authorization Server* ein und derselbe Server. Als *Client* bezeichnet man dabei jede Anwendung, die auf Ressourcen eines Benutzers in seinem Namen zugreifen möchte. Im Falle der Projektarbeit handelt es sich hierbei um *timy*, eine Zeiterfassungsanwendung, die die Autorin im Rahmen ihrer Ausbildung implementiert hat und die als erstes den Single-Sign-On-Dienst nutzen soll.

Ein anderer Ansatz wäre Kerberos, deren Infrastruktur schon im AD mit integriert ist. Allerdings ist damit die Authentifizierung auch an das AD gebunden. Sobald sich ein Benutzer außerhalb der Domäne an eine Anwendung anmelden möchte, ist dies nicht mehr möglich. Da wycomco immer mehr Webanwendungen

[13] vgl. PHP
[14] vgl. Otwell [a]
[15] vgl. Hardt

entwickelt und diese auch perspektivisch an Kunden gehen sollen, wäre eine Einschränkung an die AD nicht sinnvoll. SAML hingegen weist eine ähnliche Struktur wie die von OAuth2 auf. Während allerdings SAML längere Nachrichten über POST Parameter senden muss, greift OAuth2 auf GET zurück. Zudem sendet SAML XML Antworten an den Client zurück, der wesentlich komplizierter zu verarbeiten ist als das häufig genutzte JSON, welches auch bei wycomco standardmäßig genutzt wird. [16] Auch sendet OAuth2 keine Benutzerinformationen mit dem AccessToken mit, wie es in SAML der Fall ist, so erhält ein Angreifer nicht direkt den Zugang zu den Nutzerdaten.

Aufgrund den angeführten Merkmalen der verschiedenen Protokolle fällt die Wahl auf das OAuth2 Protokoll, welches innerhalb des SSO-Servers bei wycomco zukünftig zum Einsatz kommen wird. Für OAuth2 gibt es zudem zahlreiche Bibliotheken, die genutzt werden können um die Komplexität der Implementierung zu verringern.

4.4 Authorization Grant Types

OAuth2 bringt verschiedene Arten mit um den Nutzer zu autorisieren.

1. mit dem **Passwortansatz** gibt der Nutzer direkt am Client Nutzername und Passwort ein und sendet sie per POST Request an den Server. Das hat den Nachteil, dass der Client die Daten bearbeitet.
2. mit dem **Autorisationsansatz** lenkt der Client den Nutzer auf eine View vom Server. Dort kann der Zugriff gestattet oder abgelehnt werden. Nach Genehmigung bekommt der Client einen Authorization Code über eine Redirect URI, die der Client beim ersten Request angegeben hat. So kann sichergestellt werden, dass der richtige Client den Code bekommt. Nun kann der Client über einen Post Request mit dem Authorization Code ein Access Token beim Server beantragen.
3. mit dem **Implicitansatz** läuft der Prozess ähnlich dem Autorisationsansatz. Nur das senden eines Authorization Codes entfällt und der Client bekommt sofort den Access Code vom Server geschickt.
4. mit dem **Client Credentialsansatz** erhält der Client ein Access Token, um auf die eigenen Ressourcen zuzugreifen, und nicht im Auftrag eines Benutzers.

Im Projekt der Autorin fallen alle Grants bis auf den zweiten heraus. Beim Passwortansatz werden die Credentials des Nutzers von der Anwendung bearbeitet und verschickt werden, dies stellt ein Sicherheitsrisiko dar, was umgangen werden sollte. Der Client Credentialsansatz passt nicht zu den Anforderungen von wycomco an das Projekt, es soll eindeutig ein Nutzer authentifiziert werden, nur der Client sind nicht ausreichend. Nach Prüfung der Sicherheitsbestimmungen entschied sich die Autorin für den Autorisationsansatz. Der Nutzer kann den Zugriff gestatten und sendet seine Nutzerdaten nur über den Server, damit erhöht sich durch die Verwendet eines Authorizationscodes die Sicherheit und der Client ist nicht in die Authentifizierung verwickelt.

[16] vgl. Darfk

4.5 Architekturdesign

Beim Design wurde dem MVC Muster gefolgt, welches Laravel schon mit sich bringt. Die Anwendung wird in 3 Komponenten aufgespalten und sichert damit Flexibilität, Anpassbarkeit und Wiederverwendbarkeit. Bei einer späteren Implementierung als native Anwendung, kann das Model beibehalten werden und nur die View und der Controller müssten teilweise umgeschrieben werden.

Das **Model** enthält Daten zur Weiterverarbeitung. In vielen Fällen spiegelt ein Model eine Tabelle in der Datenbank wieder, so auch beim Eloquent ORM von Laravel. Nur in dieser Model Klasse werden die Daten bearbeitet oder erfasst. Eine Trennung vom Controller ist erwünscht, um die Ziele von MVC zu gewährleisten.

Die **View** ist das Benutzerinterface. Die Daten von den Models werden visualisiert und leitet z. B. Benutzeraktionen und Formulare weiter. Innerhalb der View sollte ein Schreibzugriff auf das Model oder den Controller vermieden werden. Somit bleiben alle Daten statisch und werden nicht verändert.

Der **Controller** empfängt Anfragen (Requests) von der View. In diesen Request sind bspw. die Login Daten eines Benutzers. Der Controller verarbeitet die Daten und sendet eine Anfrage an das User Model, welches dem Controller den richten User zurückgibt. Dieser loggt den Benutzer ein und die View zeigt den erfolgreichen Login Prozess an. *! Da war ein Kommentar....aber was stand dort??? :D !*

4.6 Entwurf der Benutzeroberfläche

Das Laravel Passport Paket bringt eine beispielhafte Oberfläche mit, in der alle relevanten Anforderungen Anwendung finden. Jedoch wurde nach Absprache mit dem Teamleiter beschlossen die Oberfläche selber zu implementieren und auf die gesonderten Anforderungen von wycomco anzupassen.

Dazu sind Screenshots und Mockups auf A.6 (Benutzeroberfläche Mockup) zu sehen. Timy stellt hier einen Client dar. Dabei handelt es sich um eine Zeiterfassungssoftware, die die Autorin im Rahmen ihrer Ausbildung implementiert hat und die auch über Single-Sign-On laufen soll. Auf diesem Client wurde eine Benutzerverwaltung implementiert, die auch auf die restlich angebundenen Clients übertragen werden kann.

Über die Hauptansicht kann der Administrator die autorisierten Clients verwalten, die per SSO auf Benutzerdaten zugreifen dürfen. Benutzer können lediglich ihr eigenes Profil verwalten.

4.7 Datenmodell

Im folgenden sollen die wichtigsten Komponenten des Datenmodells genannt und kurz erläutert werden. BeiA.8 (Entity Relationship Modell) werden die von der Autorin benötigten Entitäten dargestellt.

Entität	Beschreibung
`users`	Die Benutzertabelle mit ihren üblichen Attributen wie email, name etc.
`oauth_clients`	Die verbundenen Clients, die wyconnect akzeptiert
`oauth_auth_codes`	der Verweis auf Autorisierungscodes, die der Client vom Server erhält, wenn der Nutzer seine Autorisierung gibt. Die jedoch nicht direkt in der Datenbank zu finden sind.
`oauth_access_tokens`	Die eigentlichen Authentifizierungscodes. Sobald der Nutzer sich eingeloggt hat und die Anwendung autorisiert hat, erhält der Client einen Access Code und kann damit die Nutzerdaten per API erfragen. Wie auch bei den Autorisierungscodes sind diese nicht direkt in der Datenbank gespeichert.
`oauth_refresh_tokens`	Sobald das Access Token seine Gültigkeit verliert, kann mit diesem Refresh Token ein neues Access Token angefragt werden. Der Einfachheit halber, wird diese Funktion vernachlässigt

Tabelle 6: Entitäten

4.8 Geschäftslogik

Da wy-connect mittels PHP Laravel umgesetzt werden soll, lag es nahe bei der Implementierung auf das Passport Paket zurückzugreifen.[17] Passport bringt viele Interfaces mit, die dann nach den jeweiligen Anforderungen implementiert werden können. *! Ist das schon zuviel gesagt von Passport? !* In Kapitel 4.3 (Wahl des SSO-Protokolls) wurden bereits die verschiedenen Rollen erläutert, die im OAuth Prozess eine Rolle spielen. Nachfolgend der Workflow des Protokolls.

Zuerst muss, die neue App, die den Dienst nutzen möchte, registriert werden. Dabei werden Daten wie Anwendungsname und eine Weiterleitungsadresse - die *redirect URI*, auf die der Nutzer weitergeleitet wird, gespeichert.

redirect URI Der Service wird den Nutzer nur zur registrierten URL weiterleiten, wodurch einige Angriffe, wie z. B. Phishing verhindert werden können. Alle redirect URIs müssen TLS verschlüsselt sein, da der Dienst nur URIs akzeptiert die mit "https"beginnen. Das verhindert, dass Token während des Autorisierungsprozesses abgefangen werden können.

Client ID und Secret Nach der Registrierung der Anwendung erhält man eine *Client ID* und ein *Client Secret*. Die Client ID wird als öffentliche Information betrachtet und wird verwendet zum Erstellen von Login Seiten. Sie ist so etwas wie die öffentliche Kennung des Clients. Das Client Secret muss vertraulich behandelt werden und wird genutzt um vom Authorization Server ein Access Token zu erlangen. Das Secret ist nur der Anwendung und dem Autorisierungsserver bekannt und besteht aus einer zufälligen Zahlen-Buchstaben-Reihe.

[17] vgl. Otwell [b]

Der erste Schritt von OAuth 2 besteht darin, eine Autorisierung vom Benutzer zu erhalten. Dies wird dadurch erreicht, dass dem Benutzer eine vom Dienst bereitgestellte Schnittstelle angezeigt wird. Dazu wird der Nutzer über den Anmelden Button an folgende URL weitergeleitet:

```
https://wy-connect.wycomco.de/oauth/authorize?
client_id=1&
redirect_uri=https://wy-connect-client.test/login/wyconnect/callback&
response_type=code&
state=TMnRrCWI3HFx3crHYiqcKPepe2McHagD4NlGa5Dn
```

response_type=code - Gibt an, dass der Server einen Autorisierungscode erwartet
client_id - Die Client-ID, die beim Erstellen der Anwendung festgelegt wurde
redirect_uri - Gibt die URL an, zu dem der Benutzer nach Abschluss der Autorisierung zurückkehren soll
state - Eine zufällige Zeichenfolge, die vom Client generiert und später überprüft wird

Nachdem der Benutzer sich am Autorisierungsserver eingeloggt hat, sieht er die Autorisierungsaufforderung. Wenn der Nutzer auf Z̈ulassenk̈lickt, leitet der Dienst ihn mit einem Autorisierungscode zurück zum Client. Ein Beispiel für eine Antworte wäre:

```
https://wy-connect-client.test/login/wyconnect/callback?
code=def502009ae50a0d3dc123a13c16d3281a46e5301eec50483c3bb4e7 [...]&
state=TMnRrCWI3HFx3crHYiqcKPepe2McHagD4NlGa5Dn
```

code - Der Server gibt den Autorisierungscode zurück
state - Der Server gibt den gleichen Statuswert zurück, der übergeben wurde

Dabei ist wichtig, dass der state Wert dem vom vorherigen Request entspricht. Dieser Zustandswert wird zuerst verglichen, um sicherzustellen, dass er mit dem übereinstimmt, mit dem begonnen wurde. Sie können den Statuswert normalerweise in einem Cookie oder einer Sitzung speichern und vergleichen, wenn der Benutzer zurückkommt. Dies stellt sicher, dass Ihr Umleitungsendpunkt nicht dazu verleitet werden kann, willkürliche Autorisierungscodes auszutauschen.

Mittels des Autorisierungscodes kann nun der Client mittels folgednem Aufruf ein Access Token erlangen:

```
POST https://wy-connect.wycomco.de/token
grant_type=authorization_code&
code=def502009ae50a0d3dc123a13c16d3281a46e5301eec50483c3bb4e7 [...]&
redirect_uri=https://wy-connect-client.test/login/wyconnect/callback&
client_id=1&
client_secret=ZKKn5QypcIsO4iAh2dTdFAWJ47bHinbvMoLCYFP2
```

grant_type = authorization_code - Der Grant Type für diesen Ablauf ist authorization_code
code - Dies ist der Code, den der Nutzer im vorherigen Request erhalten hat
redirect_uri - Muss identisch mit der URL sein, die im ursprünglichen Link angegeben wurde

client_id - Die Client-ID, die beim Erstellen der Anwendung vergeben wurde
client_secret - Da diese Anfrage vom serverseitigen Code stammt, ist das Geheimnis enthalten

Der Server antwortet mit einem Zugriffstoken und einer Ablaufzeit. Mit diesem Access Token, kann timy dann die Nutzerdaten per API erfragen und den Nutzer einloggen.

Im Anhang A.10 (OAuth2 Flow) befindet sich ein Diagramm was den Ablauf des OAuth2 Authentifizierungsprozesses veranschaulicht.

4.9 Deployment

Der Server ist auf dem firmeneigenen Webserver gehostet mit der dazugehörigen Datenbank. Das komplette Repository liegt auf dem Gitlab Server und wird von da aus mittels dem GitLab Runner verarbeitet. Diese Jobs, ausgelöst durch einen Push auf den master Branch, werden innerhalb eines Docker Containers ausgeführt. Nachdem der Build erfolgreich war, werden die Tests durchlaufen und bei Erfolg wird der Build deployed und das System ist läuft auf dem Webserver.

4.10 Pflichtenheft

Ein Beispiel für das auf dem Lastenheft (siehe Kapitel **??**: **??**) aufbauende Pflichtenheft ist im Anhang A.12: Pflichtenheft (Auszug) auf Seite xiii zu finden.

4.11 Zwischenstand

Tabelle 7 zeigt den Zwischenstand nach der Entwurfsphase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Wahl des Authentifizierungsprotokolls	3 h	5 h	+2 h
2. Entwurf des SSO-Servers	6 h	6 h	
3. Entwurf der Clientanbindung	2 h	2 h	
4. Erstellen eines UML-Sequenzdiagramms und ERM der Anwendung	2 h	3 h	+1 h
5. Erstellen des Pflichtenhefts	4 h	4 h	

Tabelle 7: Zwischenstand nach der Entwurfsphase

5 Implementierungsphase

5.1 Aufsetzen des Grundgerüsts

Als erstes wurde eine neues Projekt auf dem GitLabServer erstellt und lokal geklont. Über das Terminal wurde auf Basis des MVC Musters ein neues Laravel Projekt mit Authentifizierungserweiterung angelegt. Für den Testclient wurde die fertige Instanz des von der Autorin entwickelten Zeiterfassungstool genutzt.

5.2 Middleware

Die Middleware bietet einen Filtermechanismus von HTTP-Anfragen, die in eine Anwendung eingehen. Die meist genutzte Middleware wäre hierbei die Überprüfung ob ein User authentifiziert ist. Ist dies nicht der Fall wird er automatisch zum Anmeldebildschirm weitergeleitet. Wenn der Benutzer jedoch authentifiziert ist, ermöglicht die Middleware, dass die Anforderung weiter in der Anwendung ausgeführt wird.

Im Projekt der Autorin werden die Middleware für Authentifizierung sowie der CSRF-Schutz (Schutz gegen Anfragenfälschung[18]) genutzt.

5.3 Implementierung der Benutzeroberfläche

Das eingesetzte CSS-Framework Bootstrap ermöglicht ohne Zusatzaufwand eine responsive Darstellung. Die standardisierten Oberflächengestaltungselemente fanden beim Erstellen der Views Anwendung.

Über die verschiedenen Routen, können Views oder Controllerfunktionen aufgerufen werden. Die Routen sind intuitiv benannt und werden so in der View gekennzeichnet.

Unter A.6 finden sich GUI-Ansichten der wichtigsten Views.

5.4 Implementierung des OAuth2 Flows

Mit dem Paketmanager composer wurde das Passport Paket eingebunden, welche den Grundstein des SSO-Servers legt.[19] und die erforderlichen Tabellen wurde mit dem laraveleigenen Befehlen migriert. [20]

Es wurde das ORM Modell und auch das Repository Design Pattern umgesetzt. Die Models haben eine zusätzliche Repository Klasse, die eine Entkapselung der Persistenzschicht mit sich bringt.

Im Anhang A.13: Routen innerhalb des SSO-Servers auf Seite xiv sind alle Routen aufgelistet, die der Server zur Verfügung stellt. Es ist gut zu erkennen, welche Route auf welchen Controller zugreift und wie die Zugriffsrechte über die Middlewares (siehe 5.2 (Middleware) geregelt sind. Alle Routen mit der Middleware auth können nur eingeloggte User benutzen, guest ist für alle Nutzer offen. *! kann den Kommentar nicht lesen !*

[18] vgl CSR
[19] Befehl zur Installation des Pakets: `composer require laravel/passport` vgl. COMPOSER
[20] Befehl zur Migration: `php artisan migrate` vgl. OTWELL [a]

Über ein Formular kann der Admin eines Clients den Namen und die redirect URL angeben und damit seinen Client in die Datenbank eintragen. Als response schickt der ClientController das secret. Wie üblich beim MVC Muster, leitet die View den Request mit den Clientdaten an den Controller per POST Request, der dann nach Validierung der Daten das Model beauftragt einen neuen Client anzulegen.

Mit dieser ID und der Redirect Adresse fragt der Client am Server per GET Request einen AuthorizationCode an. Sollte der Benutzer noch nicht eingeloggt sein, schützt die middleware die angeforderte Route und fordert den Nutzer auf sich einzuloggen und leitet in schließlich auf eine View, wo er der Autorisierung der Anwendung zustimmen kann. Die Antwort verarbeiten der *ApproveAuthorizationController* und der *DenyAuthorizationController*, die entweder eine Fehlermeldung zurückgeben, bei fehlerhaftem state Code oder über Passport eigene Funktionen per *completeAuthorizationRequest()* einen AuthCode generieren und ihn per HTTP dem Client als response zusenden.

Mit diesem Code, der ID und dem secret kann der Client letztendlich per HTTP POST Request eine Anfrage auf ein Access Token stellen. Der zuständige Controller validiert die Anfrage und generiert ein Token mit der Passport Funktion *respondToAccessTokenRequest*. Mit diesem Token kann per API und Bearer Autorisierung das Benutzerprofil abgefragt werden.

Die einzelnen Controller, Routen, Models und Views sind im Code Anhang zu finden und sind hier nicht verlinkt. Es wurde auf weitere Spezifizierungen der Sicherheit verzichtet, da das Passport Paket automatisiert alle wichtigen Standards aus dem Internet Engineering Task Force[21] umsetzt.

5.5 Implementierung der Anbindung an den Server

Wie auch beim SSO-Server wurde beim Client mittels composer das Socialite Paket eingebunden. Um sich mit dem Authentfizierungsserver identifizieren zu können, müssen in die Konfigurationsdatei *config/services.php* die Credentials des Clients eingetragen werden.

Die ID, sowie das SECRET sind in der *.env* Datei hinterlegt. Nachdem die Routen und Controller fertiggestellt wurden, muss der *WyconnectProvider* implementiert werden. Er setzt die verschiedenen Routen für den Abruf der Auth Codes und Access Tokens, generiert das *state* Feld und handhabt die übergeben Userdaten. Im LoginController muss dann nur noch der OAuth2 User aus dem Single-Sign-On in das Usermodel des Clients umgewandelt werden und anschließend eingeloggt werden. Als Letztes wurde der Quellcode auf den internen Gitlab Server geladen. In den angebunden Clients muss dann der Gitlab Server als Repository eingebunden werden und anschließend kann das Paket per composer verteilt werden. Nach Installation des wy-connect Providers müssen noch die Routen und der LoginController angepasst werden und der Single-Sign-On Mechanismus verlinkt werden. Siehe dazu die Quelltexte im Anhang.

5.6 Zwischenstand

Tabelle 8 zeigt den Zwischenstand nach der Implementierungsphase.

[21] vgl HARDT

Vorgang	Geplant	Tatsächlich	Differenz
1. Implementierung des Single-Sign-On Servers	18 h	16 h	-2 h
2. Implementierung der Anbindung an den Client	14 h	13 h	-1 h

Tabelle 8: Zwischenstand nach der Implementierungsphase

6 Abnahme- und Einführungsphase

Vor der Abnahme wurden sowohl Unittests, als auch manuelle Systemtests durchgeführt. Ein Screenshot der Testsuite findet sich im Anhang ANHANG!!! Alle Tests lieferten ein positives Ergebnis zurück und das Projekt konnte deployed werden. Abschließend wurde der Code von der Clientanwendung timy in den master Branch des Projektes gemerged und damit automatisiert deployed. Damit konnte produktiv der SSO-Server mit der ersten Client Anwendung laufen. Diese neue Funktionalität wurde vor dem Teamleiter präsentiert, der anschließend das Projekt abgenommen hat.

6.1 Zwischenstand

Tabelle 9 zeigt den Zwischenstand nach der Abnahme- und Einführungsphase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Abnahmetest der Fachabteilung	0,5 h	1 h	+ 0,5
1. Einführung/Benutzerschulung	0,5 h	0,5 h	

Tabelle 9: Zwischenstand nach der Abnahme- und Einführungsphase

7 Dokumentation

Die Anwendung muss aufgrund ihrer einfachen Bedienbarkeit für den Nutzer nicht dokumentiert sein. Die Programmführung ist intuitiv und selbsterklärend. Für die Entwickler wurden aussagekräftige Methoden und Klassennamen verwendet, sowie sich an die Framework-Spezifikationen gehalten und das MVC-Modell strikt umgesetzt. Für den Code selbst (Variablen-, Methodennamen, etc.) wurden selbsterklärende englische Begriffe verwendet, damit dieser im Clean Code-Prinzip (auch ohne Kommentare) lesbar ist. Mit der kostenlosen Software PHPDoc[22] wurde eine Entwicklerdokumentation automatisch generiert. Bei dieser Entwicklerdokumentation handelt es sich um eine detaillierte Beschreibung der Klassen, die in der Anwendung verwendet werden. Außerdem werden deren Attribute und Methoden sowie die Abhängigkeiten der Klassen untereinander erläutert. Diese Dokumentation soll dem Entwickler als Übersicht und Nachschlagewerk dienen. Im Anhang A.14: Screenshot der Entwicklerdokumentation auf Seite xv findet sich eine Abbildung dazu.

[22] Vgl. PHP

7.1 Zwischenstand

Tabelle 10 zeigt den Zwischenstand nach der Dokumentation.

Vorgang	Geplant	Tatsächlich	Differenz
1. Erstellen der Benutzerdokumentation	2 h	2 h	
2. Erstellen der Projektdokumentation	9,5 h	10 h	+0,5 h
3. Programmdokumentation	1 h	0,5 h	-0,5

Tabelle 10: Zwischenstand nach der Dokumentation

8 Fazit

8.1 Soll-/Ist-Vergleich

Rückblickend betrachtet, kann festgehalten werden, dass alle Anforderungen gemäß dem Pflichtenheft erfüllt wurden. In Tabelle 11 wird die Zeit, die tatsächlich für die einzelnen Phasen benötigt wurde, der zuvor eingeplanten Zeit gegenübergestellt. Es ist zu erkennen, dass nur geringfügig von der Zeitplanung abgewichen wurde. Die sich daraus ergebenen Differenzen konnten untereinander kompensiert werden, sodass das Projekt in dem von der IHK festgelegten Zeitrahmen von 70 Stunden umgesetzt werden konnte. Die zeitlichen Abweichungen kamen aufgrund von Mehraufwand bei der Recherche des OAuth2 Protokolls und gemindertem Aufwand bei der Implementierung aufgrund des verwendeten Frameworks.

Vorgang	Geplant	Tatsächlich	Differenz
Analysephase	8 h	7 h	-1 h
Entwurfsphase	17 h	20 h	+3 h
Implementierungsphase	32 h	29 h	-3 h
Übergabe	1 h	1,5 h	+0,5 h
Erstellen der Dokumentation	12 h	12,5 h	+0,5 h
Gesamt	70 h	70 h	

Tabelle 11: Soll-/Ist-Vergleich

8.2 Lessons Learned

Im Zuge der Projektdurchführung konnte die Autorin viele Erfahrungen auf dem Gebiet der Planung und Umsetzung sammeln. Besonders deutlich wurde wie wichtig eine gute Recherche ist, da ohne gründliches Wissen über das verwendete OAuth2 Protokoll keine Implementierung möglich wäre. Auch das verwendete Framework Laravel und seine Pakete erleichterten zwar die eigentliche Implementierung, aber ohne Verständnis und vorherige Recherche kann man auch dieses Werkzeug nicht bedienen.

8.3 Ausblick

Das Projekt findet jetzt Anwendung in den ersten Clientanwendungen und soll künftig noch ausgeweitet werden auf Dienste, die nicht mit Laravel geschrieben wurden und somit noch angepasst werden müssen. Auch ist es denkbar die Sicherheit zu erhöhen und eine Zwei-Faktor-Authentifizierung einzuführen oder eine Anbindung an die AD.

Literaturverzeichnis

CSR

Laravel CSRF Protection. : *Laravel CSRF Protection*, https://laravel.com/docs/5.6/csrf

PHP

PHP Manual. : *PHP Manual*, http://php.net/manual/de/index.php

php

PHPDoc. : *PHPDoc*, https://www.phpdoc.org/

intersoft consulting services AG 2016

AG intersoft consulting s.: Single Sign-on: Tipps beim Einsatz der Login-Technologie. (2016). https://www.datenschutzbeauftragter-info.de/single-sign-on-tipps-beim-einsatz-der-login-technologie/

composer

COMPOSER: *Composer*, https://getcomposer.org/

Darfk

DARFK: Single Sign-On Kerberos vs SAML vs OAuth2. http://dafrk-blog.com/de/sso-kerberos-saml-oauth-sap/

EETech Media 2014

EETECH MEDIA, LLC: How to Write Better Unit Tests For Embedded Software With TDD. (2014). https://www.allaboutcircuits.com/technical-articles/how-test-driven-development-can-help-you-write-better-unit-tests/

Git

GIT: *Git Documentation*, https://git-scm.com/documentation

GmbH 2016

GMBH, Univention: Kurz erklärt: Einmalige Anmeldung per Single Sign-on. (2016). https://www.univention.de/2016/12/einmalige-anmeldung-per-single-sign-on/

Hardt

HARDT, D.: *The OAuth 2.0 Authorization Framework*, https://tools.ietf.org/html/rfc6749

Hübscher u. a. 2007

HÜBSCHER, Heinrich ; PETERSEN, Hans-Joachim ; RATHGEBER, Carsten ; RICHTER, Klaus ; DR. SCHARF, Dirk: *IT-Handbuch*. 5. Westermann, 2007

improuv 2015

IMPROUV: TDD - so einfach und doch so schwer. (2015). https://improuv.com/blog/daniel-zappold/tdd-so-einfach-und-doch-so-schwer

www.schnelle online.info

ONLINE.INFO www.schnelle: *Arbeitstage 2018*. https://www.schnelle-online.info/Arbeitstage/Anzahl-Arbeitstage-2018.html

Oracle

ORACLE: *MySQL Documentation*, https://dev.mysql.com/doc/

Otwell a

OTWELL, Taylor: *Laravel Documentation*, http://www.laravel.com

Otwell b

OTWELL, Taylor: *Laravel Passport*, https://laravel.com/docs/master/passport

Otwell c

OTWELL, Taylor: *Laravel Socialite*, https://laravel.com/docs/5.5/socialite

Pages

PAGES, GitLab: *GitLab*, https://docs.gitlab.com

Ryte 2016

RYTE: Test Driven Development. (2016). https://de.ryte.com/wiki/Test_Driven_Development

Schonschek 2015

SCHONSCHEK, Oliver: Worauf es bei SSO Lösungen ankommt. (2015). https://www.computerwoche.de/a/worauf-es-bei-sso-loesungen-ankommt,2539134

Wikipedia 2018

WIKIPEDIA: Testgetriebene Entwicklung. (2018). https://de.wikipedia.org/wiki/Testgetriebene_Entwicklung

Eidesstattliche Erklärung

Ich, Sibylle Blümke, versichere hiermit, dass ich meine **Dokumentation zur betrieblichen Projektarbeit** mit dem Thema

"Single-Sign-On"-Authentifizierungsserver mit Beispielimplementierung eines Clients –

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

Berlin, den 12.04.2018

Sibylle Blümke

A Anhang

A.1 Detaillierte Zeitplanung

Analysephase			**8 h**
1. Analyse des Ist-Zustands		3 h	
1.1. Fachgespräch mit den Projektleitern	1 h		
1.2. Prozessanalyse	2 h		
2. Soll-Analyse		2 h	
3. Wirtschaftlichkeitsanalyse durchführen		1 h	
4. Erstellen des Lastenhefts mit den Projektleitern		2 h	
Entwurfsphase			**17 h**
1. Wahl des Authentifizierungsprotokolls		3 h	
1.1. Analyse OAuth2 Workflow	1 h		
1.2. Analyse Grant Types	2 h		
2. Entwurf des SSO-Servers		6 h	
3. Entwurf der Clientanbindung		2 h	
4. Erstellen eines UML-Sequenzdiagramms und ERM der Anwendung		2 h	
5. Erstellen des Pflichtenhefts		4 h	
Implementierungsphase			**32 h**
1. Implementierung des Single-Sign-On-Servers		18 h	
1.1. Implementierung des OAuth2 Protokolls	12 h		
1.2. Implementierung der Benutzerverwaltung	3 h		
1.3. Implementierung der Benutzeroberfläche	3 h		
2. Implementierung der Anbindung an den Client		14 h	
2.1. Implementierung des OAuth2 Handlings	10 h		
2.2. Implementierung Nutzerhandlings	4 h		
Übergabe			**1 h**
1. Abnahme durch die Projektleiter		0,5 h	
2. Einweisung in die Anwendung		0,5 h	
Erstellen der Dokumentation			**12 h**
1. Erstellen der Benutzerdokumentation		2 h	
2. Erstellen der Projektdokumentation		9,5 h	
3. Programmdokumentation		0,5 h	
3.1. Generierung durch PHPDoc	0,5 h		
Gesamt			**70 h**

A.2 Verwendete Ressourcen

A.2.1 Hardware

- Büroarbeitsplatz mit iMac

A.2.2 Software

- OS X 10.13 High Sierra – Betriebssystem

- LucidChart – Anwendung zum Erstellen von UML-Diagrammen
- Creately – Anwendung zum Erstellen von Online-UI-Mockups
- draw.io – Diagramm-Editor (Erweiterung für Google Drive)
- PHP – Programmiersprache
- PHPStorm – IDE mit Education Lizenz
- PHPUnit – Testframework für PHP
- Laravel 5.5 - PHP Framework
- Vue.js - Javascript Framework
- HTML, CSS Webtechnologien
- JetBrains PHP Storm – Entwicklungsumgebung PHP
- git – Verteilte Versionsverwaltung
- Gitlab – Selfhosted Repository Verwaltung
- texmaker – LaTeX Editor
- Sequel Pro – Verwaltungswerkzeug für Datenbanken

A.3 Lastenheft (Auszug)

Es folgt ein Auszug aus dem Lastenheft mit Fokus auf die Anforderungen:

Die Anwendung muss folgende Anforderungen erfüllen:

1. Single-Sign-On-Server
   - 1.1. Die Anwendung verfügt über eine eigene Datenbank für die Benutzerdaten.
   - 1.2. Anzeigen einer Übersichtsseite für autorisierte Clienten und Token mit allen relevanten Informationen zu diesen.
   - 1.3. Die Authentifizierung muss über einen sicheren Kanal erfolgen.
2. Login Prozess der Client Anwendung
   - 2.1. Die Client Anwendung verfügt über eine eigene Benutzerdatenbank.
   - 2.2. Die Authentifizierungsmöglichkeit über wy-connect muss einfach auf andere Clients übertragbar sein.
   - 2.3. Der Nutzer muss die Client Anwendung zum Zugriff autorisieren.
   - 2.4. Der Nutzer meldet sich nur am Server mit seinen Nutzerdaten an.
3. Sonstige Anforderungen
   - 3.1. Der Server sowie der Login Prozess am Client soll über eine GUI intuitiv bedient werden können.
   - 3.2. Der Server sowie der Client müssen ohne Installation von zusätzlicher Software über einen Webbrowser im Intranet erreichbar sein.
   - 3.3. Zur Versionskontrolle der Anwendungsentwicklung soll ein Git[23]-Repository verwendet werden.
   - 3.4. Die Anwendung soll in der Programmiersprache PHP[24] mittels des Frameworks Laravel[25] umgesetzt werden.
   - 3.5. Der Server soll auf dem firmeninternen Webserver gehostet werden.
   - 3.6. Bei Einsatz einer MySQL-Datenbank[26] soll der firmeninterne MySQL Server Verwendung finden.
   - 3.7. Der Einrichtungsprozess für neue Clienten muss dokumentiert werden.

[...]

[23] vgl. Git
[24] vgl. PHP
[25] vgl. Otwell [a]
[26] vgl. Oracle

A.4 Use-Case-Diagramm ohne SSO

Das folgende Diagramm beschreibt den Anmeldeprozess an verschiedenen Systemen ohne SSO.

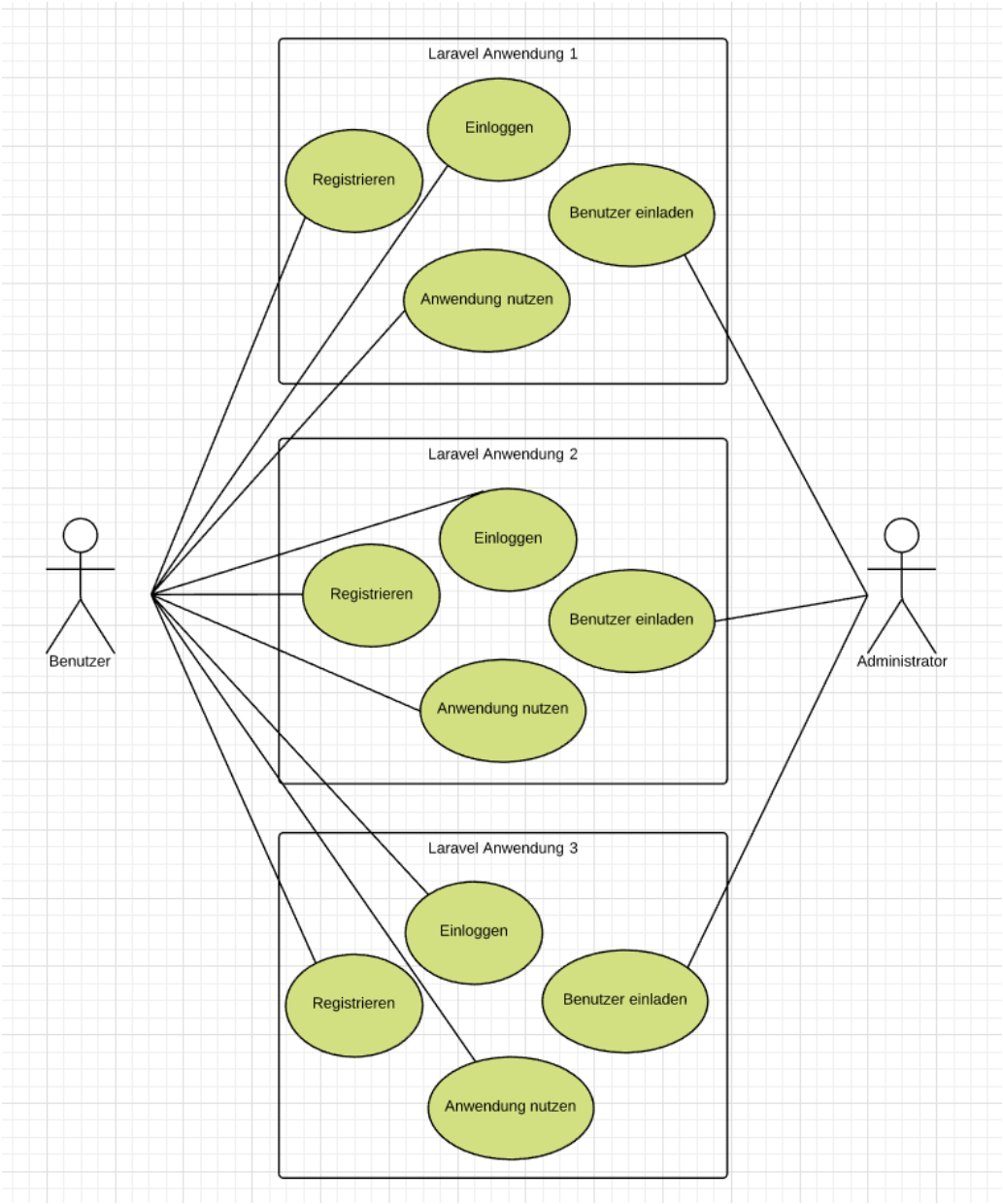

Abbildung 3: Use-Case-Diagramm ohne SSO

A.5 Use-Case-Diagramm mit SSO

Das folgende Diagramm beschreibt den Anmeldeprozess an verschiedenen Systemen mit SSO.

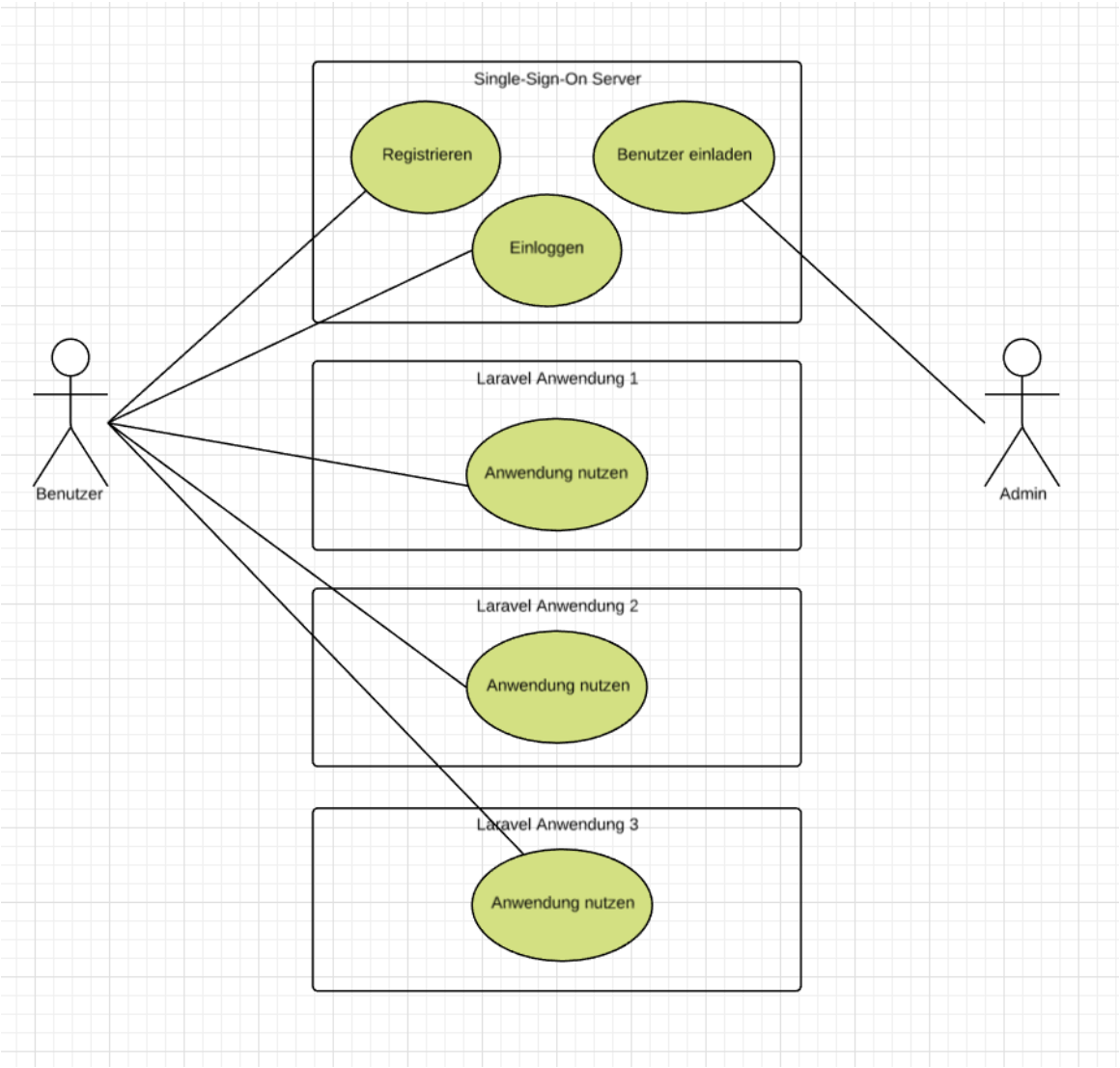

Abbildung 4: Use-Case-Diagramm mit SSO

A.6 Benutzeroberfläche Mockup

Das folgende Template beschreibt den Anmeldeprozess am SSO-Server.

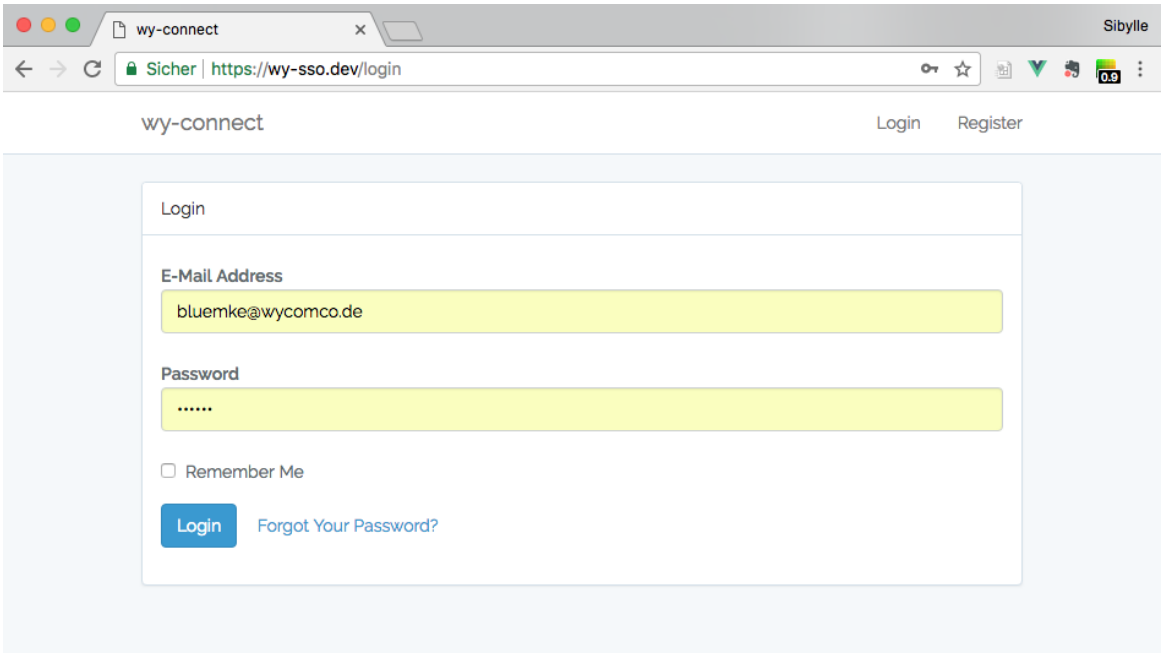

Abbildung 5: Login am SSO-Servers

A.7 Benutzeroberfläche Screenshots

Der folgende Screenshot beschreibt die Benutzeroberfläche an verschiedenen Systemen mit SSO.

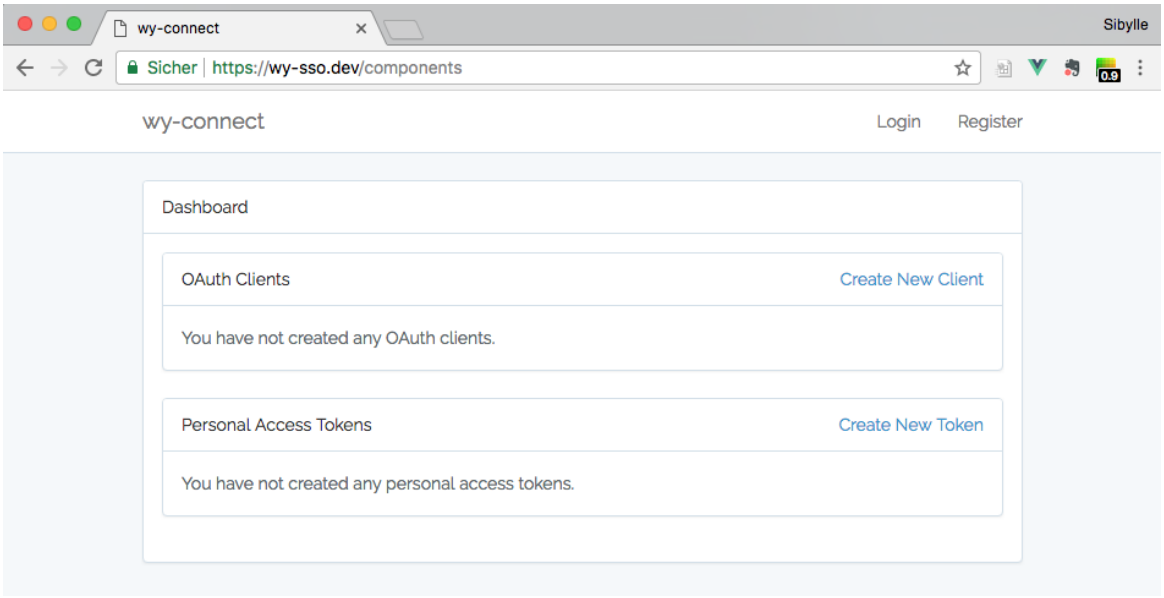

Abbildung 6: Benutzeroberfläche des SSO-Servers

Der folgende Screenshot beschreibt den Anmeldeprozess am Client mit SSO.

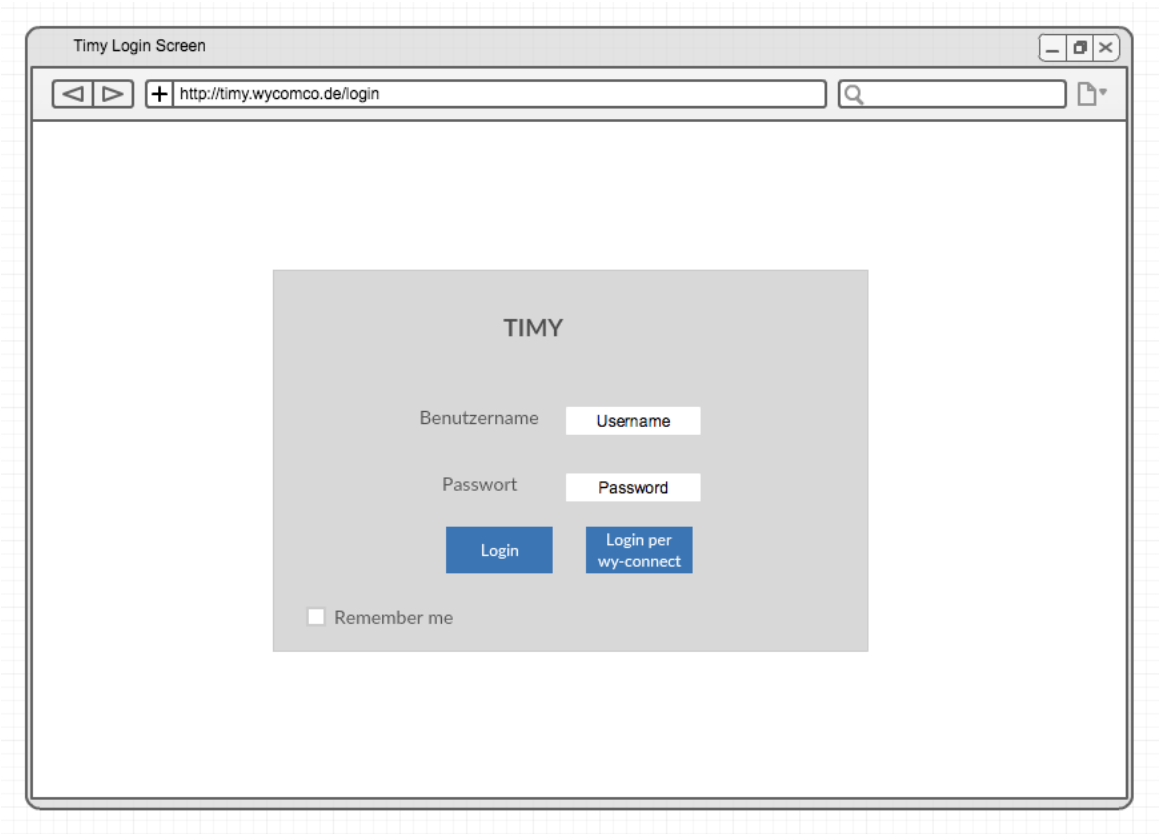

Abbildung 7: Login am Client Timy

A.8 Entity Relationship Modell

Das folgende Diagramm zeigt die Entitäten und Attribute der Datenbank und deren Beziehung zueinander.

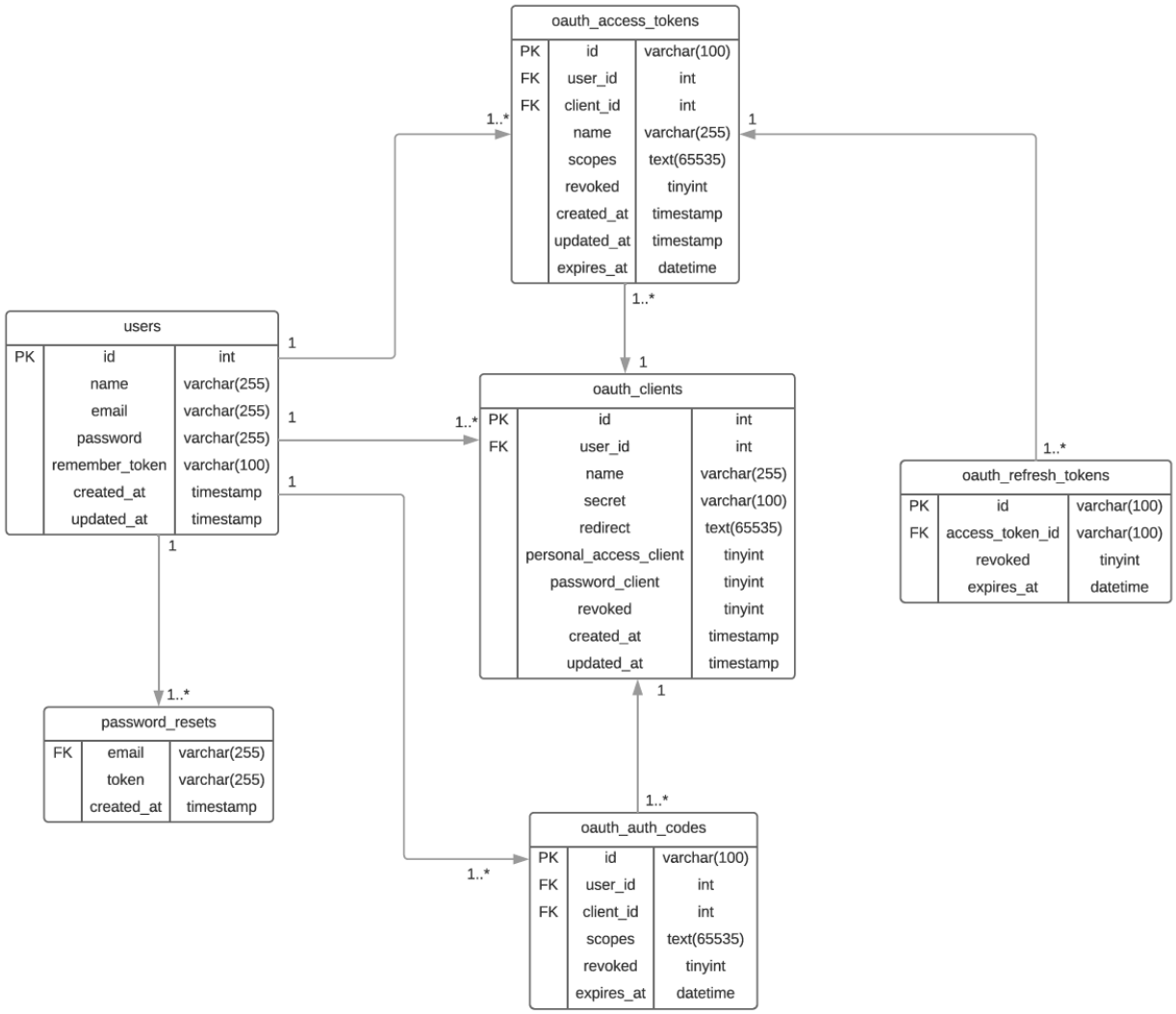

Abbildung 8: ERM

A.9 Amortisationsdiagramm

Das folgende Diagramm zeigt die Amortisation

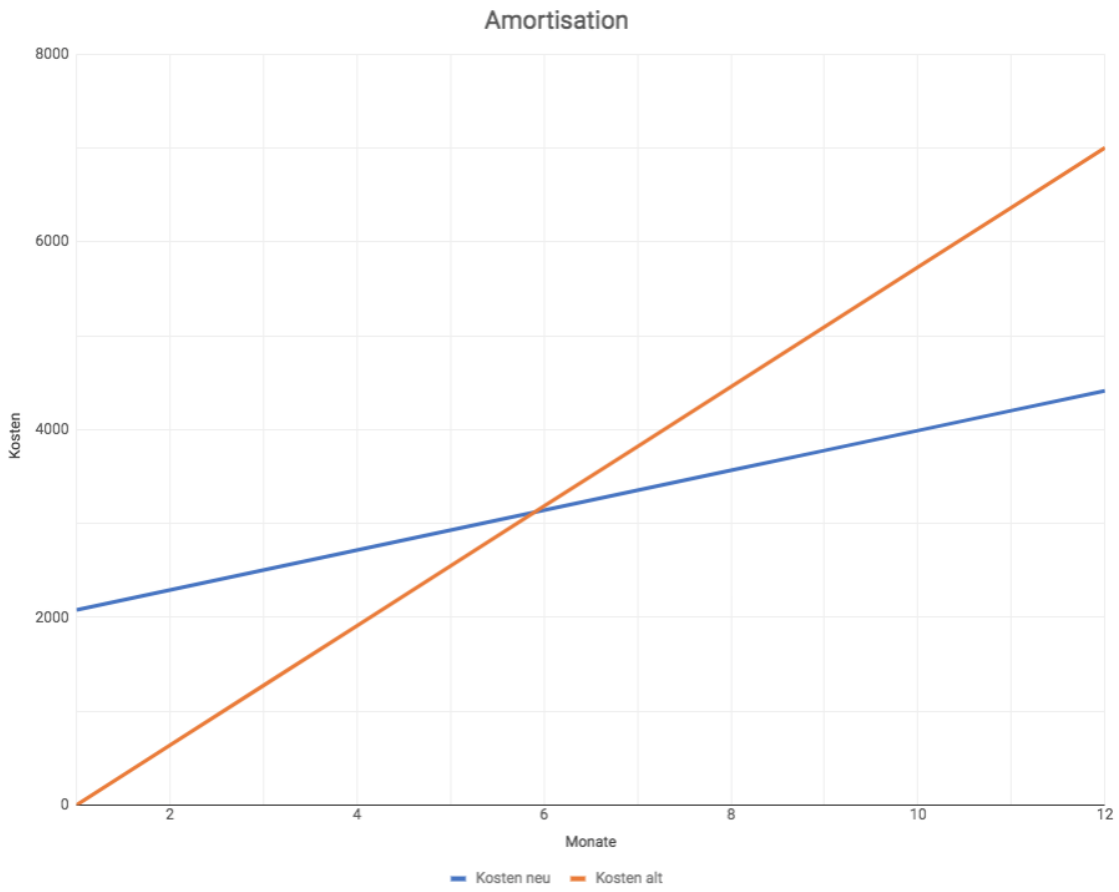

Abbildung 9: Amortisations-Diagramm

A.10 OAuth2 Flow

Das folgende Diagramm zeigt den Ablauf des OAuth2 Protokolls am Beispiel des Clients Timy.

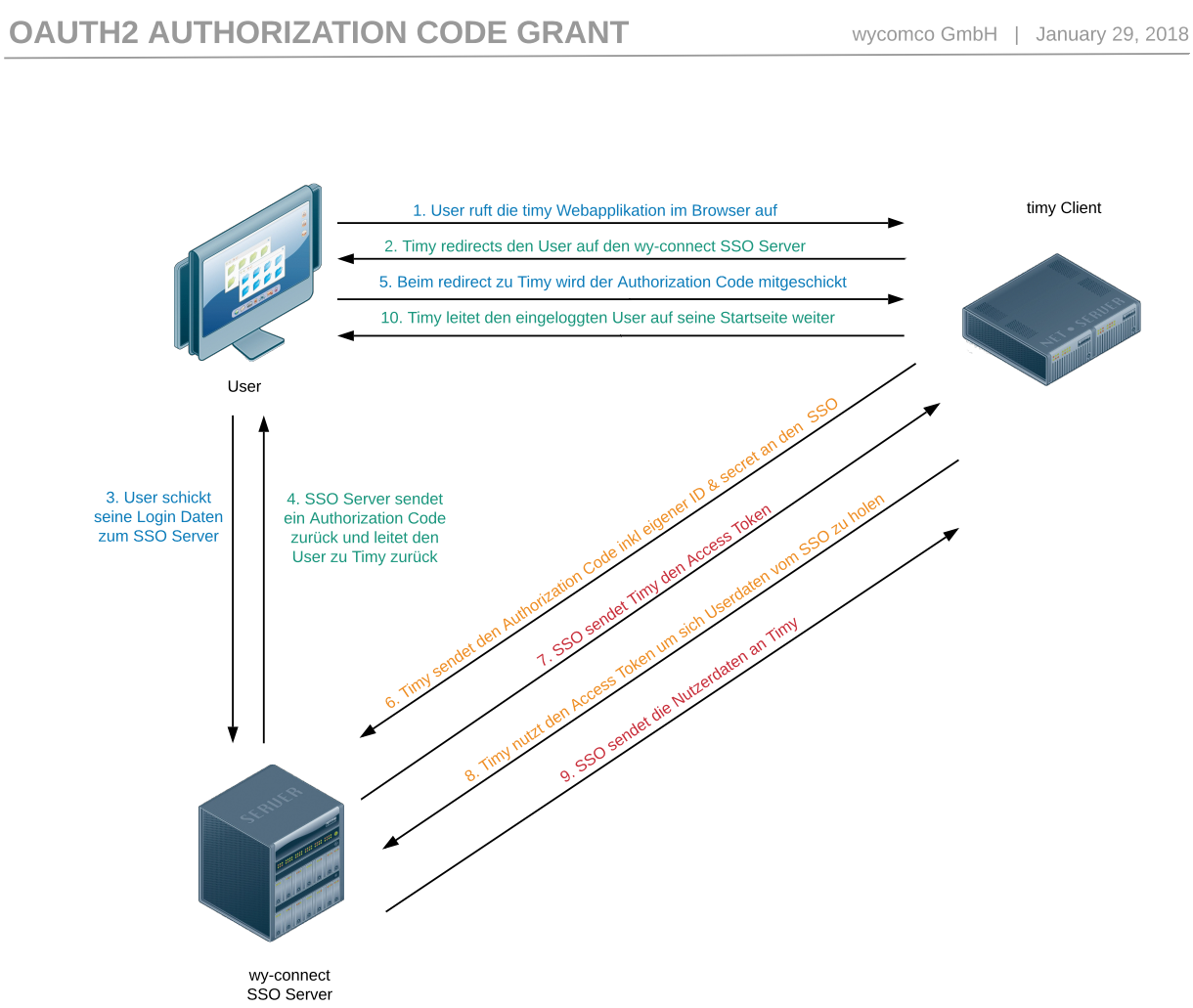

Abbildung 10: OAuth2 Flow

A.11 Sequenzdiagramm OAuth2

Das folgende Diagramm zeigt den Ablauf des OAuth2 Protokolls am Beispiel des Clients Timy.

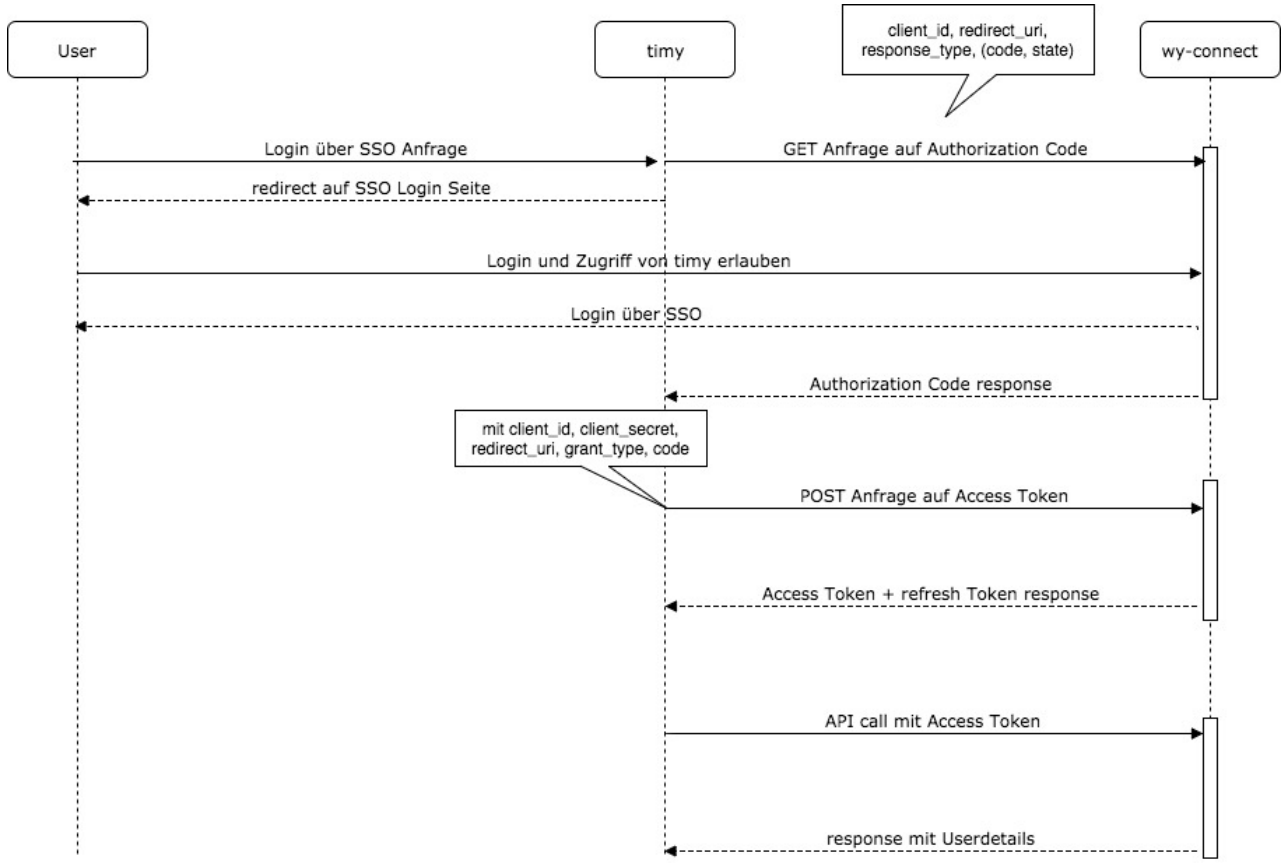

Abbildung 11: Sequenzdiagramm OAuth2

A.12 Pflichtenheft (Auszug)

Die geplante Umsetzung der im Lastenheft (Auszug siehe A.3) definierten Anforderungen wird in folgendem Auszug aus dem Pflichtenheft beschrieben:

Umsetzung der Anforderungen

1. Single-Sign-On-Server
   1.1. Für das speichern der Benutzerdaten wird die firmeninterne MySQL Datenbank genutzt.
   1.2. Innerhalb des **SSO-Servers!** gibt eine View für die Benutzerverwaltung:
      - Admins können Nutzer anlegen, löschen, bearbeiten
      - Benutzer können nur ihr eigenes Profil löschen und bearbeiten
   1.3. Als Authentifizierungsprotokoll wird OAuth2 [27]genutzt.
      - die Autorisierung wird hierbei über die API vorgenommen
      - die Authentifizierung erfolgt ausschließlich über den Server
      - zur Umsetzung wird das Laravel Paket Passport[28] genutzt
2. Login Prozess der Client Anwendung
   2.1. Für das speichern der Benutzerdaten wird die firmeninterne MySQL Datenbank genutzt.
   2.2. Die wy-connect Anbindung an andere Laravel Anwendungen wird mittels Installation eines bereitgestellten Pakets umgesetzt.
      - dieses Paket kann über einen einfachen Konsolenbefehl in den Clienten integriert werden
      - der wy-connect Provider liegt dabei auf der firmeneigenen Gitlab Instanz
      - zur Umsetzung wird das Laravel Paket Socialite[29] genutzt
   2.3. Nachdem Login Prozess am SSO wird der Benutzer gefragt ob der Test Client auf die Nutzerdaten zugreifen darf.
   2.4. Innerhalb des Login Prozesses am Clienten werden die Nutzerdatzen nur vom SSO-Server abgefragt.
3. Sonstige Anforderungen
   3.1. Über das Webinterface kann sich ein Nutzer anmelden und hat über eine übersichtlich gestaltete GUI Zugriff auf seine Einstellungen.
   3.2. Das Programm läuft als Webanwendung.
   3.3. Zur Versionskontrolle wird der firmeneigene GitLab-Server[30] genutzt.
   [...]

[27] vgl. Hardt
[28] vgl. Otwell [b]
[29] vgl. Otwell [c]
[30] vgl. Pages

A.13 Routen innerhalb des SSO-Servers

Die folgende Ausgabe zeigt alle vorhanden Routen, die der Server zur Verfügung stellt.

```
➜ wy-sso git:(master) php artisan route:list
+--------+----------+-----------------------------------------+------------------+-----------------------------------------------------------------------------------------+--------------+
| Domain | Method   | URI                                     | Name             | Action                                                                                  | Middleware   |
+--------+----------+-----------------------------------------+------------------+-----------------------------------------------------------------------------------------+--------------+
|        | GET|HEAD | /                                       |                  | Closure                                                                                 | web          |
|        | GET|HEAD | api/user                                |                  | Closure                                                                                 | api,auth:api |
|        | GET|HEAD | components                              |                  | Closure                                                                                 | web          |
|        | GET|HEAD | home                                    | home             | App\Http\Controllers\HomeController@index                                               | web,auth     |
|        | POST     | login                                   |                  | App\Http\Controllers\Auth\LoginController@login                                         | web,guest    |
|        | GET|HEAD | login                                   | login            | App\Http\Controllers\Auth\LoginController@showLoginForm                                 | web,guest    |
|        | POST     | logout                                  | logout           | App\Http\Controllers\Auth\LoginController@logout                                        | web          |
|        | POST     | oauth/authorize                         |                  | \Laravel\Passport\Http\Controllers\ApproveAuthorizationController@approve               | web,auth     |
|        | GET|HEAD | oauth/authorize                         |                  | \Laravel\Passport\Http\Controllers\AuthorizationController@authorize                    | web,auth     |
|        | DELETE   | oauth/authorize                         |                  | \Laravel\Passport\Http\Controllers\DenyAuthorizationController@deny                     | web,auth     |
|        | POST     | oauth/clients                           |                  | \Laravel\Passport\Http\Controllers\ClientController@store                               | web,auth     |
|        | GET|HEAD | oauth/clients                           |                  | \Laravel\Passport\Http\Controllers\ClientController@forUser                             | web,auth     |
|        | PUT      | oauth/clients/{client_id}               |                  | \Laravel\Passport\Http\Controllers\ClientController@update                              | web,auth     |
|        | DELETE   | oauth/clients/{client_id}               |                  | \Laravel\Passport\Http\Controllers\ClientController@destroy                             | web,auth     |
|        | GET|HEAD | oauth/personal-access-tokens            |                  | \Laravel\Passport\Http\Controllers\PersonalAccessTokenController@forUser                | web,auth     |
|        | POST     | oauth/personal-access-tokens            |                  | \Laravel\Passport\Http\Controllers\PersonalAccessTokenController@store                  | web,auth     |
|        | DELETE   | oauth/personal-access-tokens/{token_id} |                  | \Laravel\Passport\Http\Controllers\PersonalAccessTokenController@destroy                | web,auth     |
|        | GET|HEAD | oauth/scopes                            |                  | \Laravel\Passport\Http\Controllers\ScopeController@all                                  | web,auth     |
|        | POST     | oauth/token                             |                  | \Laravel\Passport\Http\Controllers\AccessTokenController@issueToken                     | throttle     |
|        | POST     | oauth/token/refresh                     |                  | \Laravel\Passport\Http\Controllers\TransientTokenController@refresh                     | web,auth     |
|        | GET|HEAD | oauth/tokens                            |                  | \Laravel\Passport\Http\Controllers\AuthorizedAccessTokenController@forUser              | web,auth     |
|        | DELETE   | oauth/tokens/{token_id}                 |                  | \Laravel\Passport\Http\Controllers\AuthorizedAccessTokenController@destroy              | web,auth     |
|        | POST     | password/email                          | password.email   | App\Http\Controllers\Auth\ForgotPasswordController@sendResetLinkEmail                   | web,guest    |
|        | GET|HEAD | password/reset                          | password.request | App\Http\Controllers\Auth\ForgotPasswordController@showLinkRequestForm                  | web,guest    |
|        | POST     | password/reset                          |                  | App\Http\Controllers\Auth\ResetPasswordController@reset                                 | web,guest    |
|        | GET|HEAD | password/reset/{token}                  | password.reset   | App\Http\Controllers\Auth\ResetPasswordController@showResetForm                         | web,guest    |
|        | GET|HEAD | register                                | register         | App\Http\Controllers\Auth\RegisterController@showRegistrationForm                       | web,guest    |
|        | POST     | register                                |                  | App\Http\Controllers\Auth\RegisterController@register                                   | web,guest    |
+--------+----------+-----------------------------------------+------------------+-----------------------------------------------------------------------------------------+--------------+
```

Abbildung 12: Routen des SSO-Servers

A.14 Screenshot der Entwicklerdokumentation

Der folgende Screenshot zeigt einen Ausschnitt einer Klasse innerhalb der Entwicklerdokumentation.

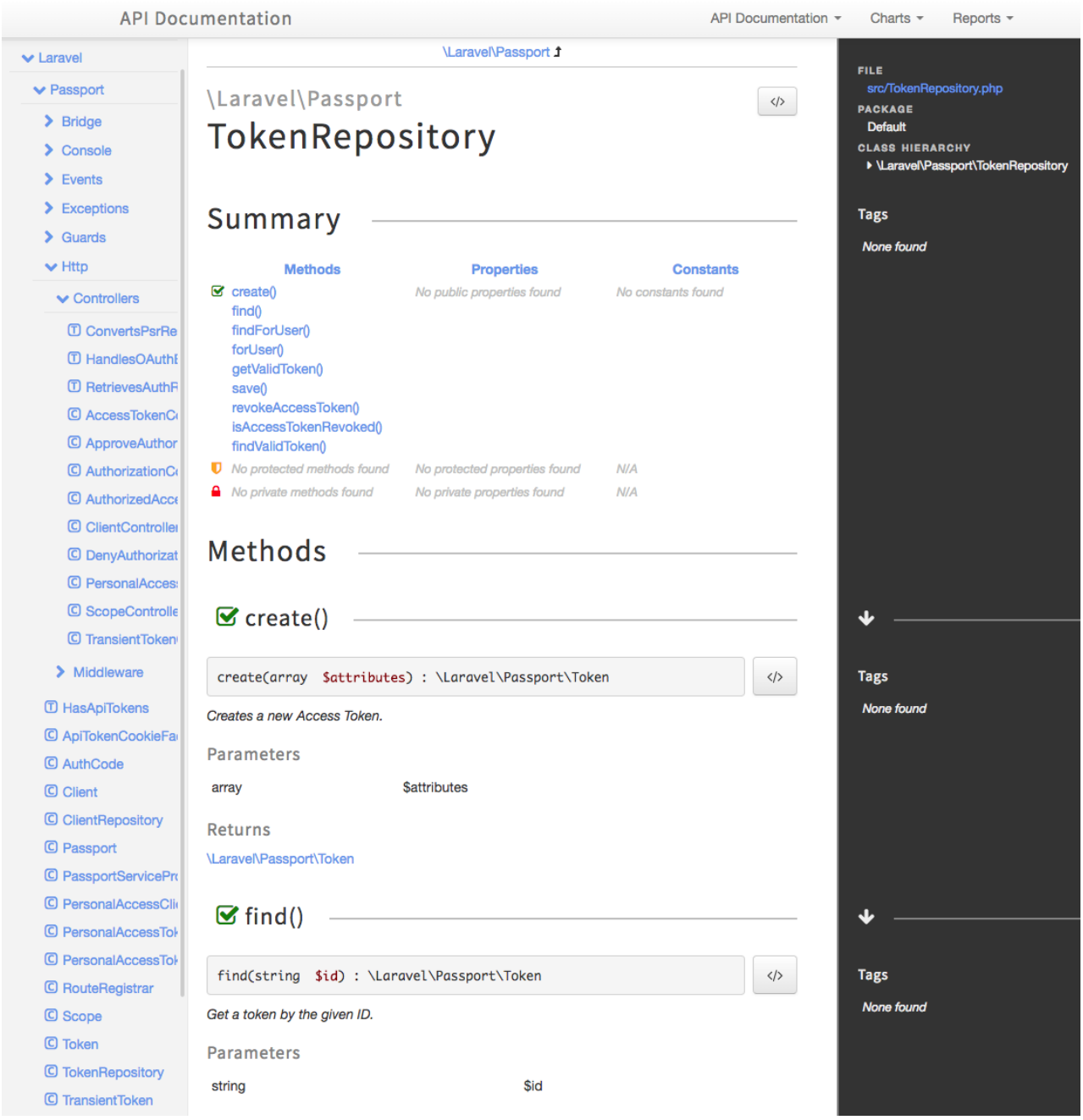

Abbildung 13: Benutzerdokumentation