



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



Ausbildungsbetrieb:

wycomco GmbH
Fasanenstr. 35
10719 Berlin

Inhaltsverzeichnis

Abbildungsverzeichnis	IV
Tabellenverzeichnis	V
Listings	VI
Abkürzungsverzeichnis	VII
1 Einleitung	1
1.1 Projektumfeld	1
1.2 Projektziel	1
1.3 Projektbegründung	1
1.4 Projektschnittstellen	2
1.5 Projektabgrenzung	2
2 Projektplanung	2
2.1 Projektphasen	2
2.2 Ressourcenplanung	2
2.3 Entwicklungsprozess	3
3 Analysephase	4
3.1 Ist-Analyse	4
3.2 Wirtschaftlichkeitsanalyse	4
3.2.1 „Make or Buy“-Entscheidung	4
3.2.2 Projektkosten	5
3.2.3 Amortisationsdauer	5
3.3 Nutzwertanalyse	6
3.4 Qualitätsanforderungen	7
3.5 Anwendungsfälle	7
3.6 Lastenheft/Fachkonzept	7
3.7 Zwischenstand	8
4 Entwurfsphase	8
4.1 Zielplattform	8
4.2 Framework	8
4.3 Wahl des SSO-Protokolls	9
4.4 Authorization Grant Types	9
4.5 Architekturdesign	10
4.6 Entwurf der Benutzeroberfläche	11
4.7 Datenmodell	11
4.8 Geschäftslogik	11

Inhaltsverzeichnis

4.9	Pflichtenheft	13
4.10	Zwischenstand	13
5	Implementierungsphase	14
5.1	Aufsetzen des Grundgerüsts	14
5.2	Middleware	14
5.3	Implementierung der Benutzeroberfläche	14
5.4	Implementierung des OAuth2 Flows	14
5.5	Implementierung der Anbindung an den Server	15
5.6	Zwischenstand	16
6	Abnahme- und Einführungsphase	16
6.1	Zwischenstand	16
7	Dokumentation	16
7.1	Zwischenstand	17
8	Fazit	17
8.1	Soll-/Ist-Vergleich	17
8.2	Lessons Learned	17
8.3	Ausblick	18
	Literaturverzeichnis	19
	Eidesstattliche Erklärung	21
A	Anhang	i
A.1	Detaillierte Zeitplanung	i
A.2	Verwendete Ressourcen	i
A.2.1	Hardware	i
A.2.2	Software	i
A.3	Lastenheft (Auszug)	iii
A.4	Use-Case-Diagramm ohne SSO	iv
A.5	Use-Case-Diagramm mit SSO	v
A.6	Benutzeroberfläche Mockup	vi
A.7	Benutzeroberfläche Screenshots	vii
A.8	Entity Relationship Modell	x
A.9	Amortisationsdiagramm	xi
A.10	OAuth2 Flow	xii
A.11	Sequenzdiagramm OAuth2	xiii
A.12	Pflichtenheft (Auszug)	xiv
A.13	Routen innerhalb des SSO-Servers	xv
A.14	Ausschnitt des Klassendiagramms vom SSO-Server	xvi

Inhaltsverzeichnis

A.15	Ausschnitt des Klassendiagramms vom Client	xvii
A.16	Ausgabe der Unit- und Featuretests	xviii
A.17	Screenshot der Entwicklerdokumentation	xix

Abbildungsverzeichnis

1	grobe Zeitplanung	2
2	TDD Zirkel	3
3	Use-Case-Diagramm ohne SSO	iv
4	Use-Case-Diagramm mit SSO	v
5	Loginscreen Mockup Timy	vi
6	Loginseite des SSO-Servers	vii
7	Benutzeroberfläche des SSO-Servers	viii
8	Login am Client Timy	ix
9	ERM	x
10	Amortisations-Diagramm	xi
11	OAuth2 Flow	xii
12	Sequenzdiagramm OAuth2	xiii
13	Routen des SSO-Servers	xv
14	Ausschnitt aus dem UML Diagramm des SSO-Servers	xvi
15	Ausschnitt aus dem UML Diagramm des Clients	xvii
16	Konsolenausgabe der PHPUnit Tests	xviii
17	Benutzerdokumentation	xix

Tabellenverzeichnis

1	Kostenaufstellung	5
2	Zeitersparnis pro Vorgang	6
3	Zeitersparnis pro Monat	6
4	Qualitätsanforderungen	7
5	Zwischenstand nach der Analysephase	8
6	Entitäten	11
7	Zwischenstand nach der Entwurfsphase	13
8	Zwischenstand nach der Implementierungsphase	16
9	Zwischenstand nach der Abnahme- und Einführungsphase	16
10	Zwischenstand nach der Dokumentation	17
11	Soll-/Ist-Vergleich	17

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

Der 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.

Das Projekt wurde durch die Entwicklungsabteilung von wycomco in Auftrag gegeben.

1.2 Projektziel

Wycomco konzentriert sich zunehmend auf die Implementierung von Webanwendungen zur internen Nutzung. Bisher müssen sich die Mitarbeiter zur Nutzung jeweils mit eigenem - teils verschiedenen - Benutzernamen und Passwort registrieren und anschließend anmelden. 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

Das Unternehmen nutzt aktuell drei Anwendungen, die im Betrieb laufen oder sich in der Testphase befinden. Alle drei haben eine eigene Benutzerverwaltung mit eigenen, teils verschiedenen Benutzerkennungen. Die Nutzer müssen an Allen in regelmäßigen Abständen das Passwort ändern. In der Folge gehört es zum Standard, dass Nutzer Ihre Passwörter aufschreiben, diese generieren lassen oder vergessene Passwörter neu vergeben. In der Praxis führt dies zur Nutzung von Trivialpasswörtern, wiederholter Vergabe von Passwörtern oder zur mangelhaften Sicherung der sensiblen Daten.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](#).

Die Implementierung einer Single Sign-on-Lösung ermöglicht dem Nutzer ausschließlich ein Master-Passwort zu vergeben, welches nur einmalig bei der Anmeldung am wy-connect Dienst eingegeben werden muss. 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. Auch der administrative Aufwand verringert sich bei einer zentralen Nutzerverwaltung².

¹vgl. AG [2016]

²vgl. GMBH [2016]

2 Projektplanung

Dies verspricht eine bessere Usability und eine nicht zu unterschätzende Zeitersparnis für Anwender im Vergleich zur derzeitigen Lösung. 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 Active Directory (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

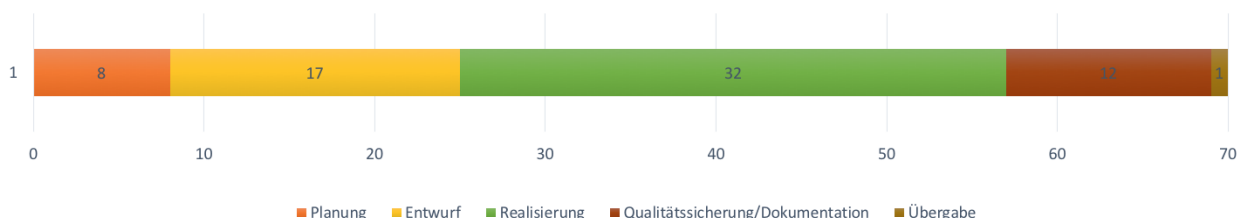


Abbildung 1: grobe Zeitplanung

im Anhang A.1: [Detaillierte Zeitplanung](#) auf Seite i.

2.2 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 Projektplanung

2.3 Entwicklungsprozess

Das Projekt wurde anhand des Wassfallmodells entwickelt. Bei diesem nicht iterativen, linearen Prozess werden die einzelnen Projektphasen schrittweise bearbeitet. Hierbei bilden die Phasen-Ergebnisse jeweils die bindende Vorgabe für die nächste Projektphase³. 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. Somit wurde jede geschriebene Zeile des Produktivcode zuvor getestet. Damit lässt sich die Qualität des Codes erhöhen und der Wartungsaufwand im Nachhinein verringern⁴. Diese Parallelentwicklung von Code und Tests erfolgt in sich wiederholenden Mikroiterationen, die nur einen kleinen Zeitraum in Anspruch nehmen sollten. Dabei kann TDD in drei Hauptteile aufspaltet werden. Im Englischen werden diese Red-Green-Refactor genannt. Die einzelnen Phasen lassen sich wie folgt beschreiben⁵. 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 refaktoriert, um einfachen, sauberen Quelltext herzustellen.

Zur Verdeutlichung der drei Phasen die Abbildung 2⁶

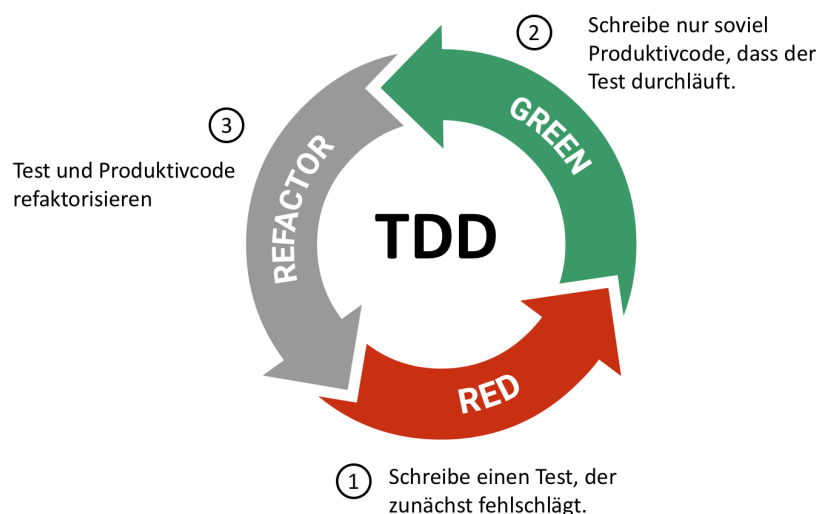


Abbildung 2: TDD Zirkel

³vgl. [HÜBSCHER U. A. 2007, S. 263]

⁴vgl. AG [2016]

⁵vgl. WIKIPEDIA [2018]

⁶vgl. EETECH MEDIA [2014]

3 Analysephase

3.1 Ist-Analyse

Derzeit gibt es bei der Anmeldung an den verschiedenen Diensten bei wycomco keine Optimierungen.

Dieser Prozess lässt wie folgt beschreiben:

1. Anlegen eines vorläufigen Benutzerkontos durch den Admin
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. Auch im täglichen Geschäft ist es erforderlich sich an jeder Anwendung einzeln anzumelden.

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. 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, die entsprechende Lösungen anbieten. Grundlegend für die Entscheidung für ein SSO-Verfahren ist dessen Anwendbarkeit auf die Ansprüche des Nutzers. Dabei ist zu beachten, dass ein Großteil der genutzten Anwendungen unterstützt werden sollten. Vorgaben für komplexe Passwörter und verschlüsselte Anmeldeverfahren sollten Standard sein, um Missbrauch zu verhindern. 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 Administratoren⁷.

Ein SSO-Verfahren, das tatsächlich alle eingesetzten Anwendungen einbinden kann und die oben genannten Voraussetzungen für wycomco erfüllt, konnte trotz ausführlicher Recherche nicht gefunden werden. Da der Großteil der entwickelten Anwendungen bei wycomco auf Laravel⁸ basiert, lag es nahe die zur Verfügung

⁷vgl. SCHONSCHKE [2015]

⁸vgl. OTWELL [a]

3 Analysephase

gestellten Pakete zu nutzen. Dabei handelt es sich um Passport⁹ und Socialite¹⁰, die die Grundfunktionen eines SSO Servers und den angebundenen Clients bieten. Damit kann das Augenmerk auf die individuellen Anforderungen von wycomco gelegt werden. Nach Abwägung der genannten Kriterien wurde sich dazu entschieden das Projekt in Eigenentwicklung durchzuführen.

3.2.2 Projektkosten

Für die Projektkosten 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 Ressourcen¹¹ 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

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 Monate das Passwort ändern. Durch die Vielzahl an Freelancern in der Firma, schätzt man die Anzahl neuer User pro Monat auf eins. Dazu die Auflistungen in Tabelle 2 und Tabelle 3. Für die Zeitersparnis pro Monat ergeben sich damit 463 Minuten. Dies ergibt eine tägliche Ersparnis von

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

⁹vgl. OTWELL [b]

¹⁰vgl. OTWELL [c]

¹¹Hardware, Arbeitsplatz, etc.

3 Analysephase

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

Bei einer Zeiteinsparung von 23,15 Minuten pro Tag an 252 Arbeitstagen¹² im Jahr ergibt sich eine Zeiteinsparung von

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

Dadurch ergibt sich eine jährliche Einsparung von

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

Die Amortisationszeit beträgt also $\frac{2080,00 \text{ €}}{5347,65 \text{ €/Jahr}} \approx 0,4 \text{ Jahre} \approx 5 \text{ Monate}$. 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 weitere. Wie in der [1.3 \(Projektbegründung\)](#) schon erläutert, bringt der Einsatz einer Single-Sign-On-Lösung eine erhöhte Sicherheit mit sich, da die Möglichkeit des unbefugten Zugriffs auf vertrauliche Daten stark minimiert werden würde. Wycomco verwart in einer ihrer Anwendungen beispielsweise sensible Kundeninformationen. Mit der Anwendung der entwickelten Single-Sign-On-Lösung, sinkt das Risiko des Datenverlustes. Mögliche Kosten durch Datenverlust, bei Nichtanwendung der entwickelten

¹²vgl. WWW.SCHNELLE.ONLINE.INFO

3 Analysephase

Lösung, werden als Opportunitätskosten bezeichnet. Möglicher Fremdzugriff bringt eine Vertragsstrafe mit sich. Bei größeren Kunden liegt diese bei schätzungsweise 60.000 . € Das sind Kosten die mit der Umsetzung von wy-connect vermieden werden können. Obwohl das Projekt im Rahmen von wycomco entwickelt wurde, eröffnet sich die Möglichkeit die entwickelte Applikation Kunden von wycomco zur Verfügung zu stellen. Der tatsächliche Nutzen geht also über wycomco hinaus.

3.4 Qualitätsanforderungen

Die Qualitätsanforderungen an die Anwendung lassen sich Tabelle 4 entnehmen.

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

3.5 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 erkennbar 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.

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 A.3: Lastenheft (Auszug) auf Seite iii

4 Entwurfsphase

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 Applikation entstanden sein, die für eventuelle Weiterentwicklungen eine größtmögliche Plattform-Unabhängigkeit gewährleistet und über einen zeitgemäßen Browser oder auf mobilen Endgeräten von jedem Nutzer verwendet werden kann. Die Daten, auf die zugegriffen werden soll, sind in einer bestehenden MySQL Datenbank gespeichert. Als Programmiersprache wurde PHP: Hypertext Preprocessor (PHP)¹³ gewählt, da dies bereits Grundlage zahlreicher Anwendungen ist, somit leichter an wy-connect angebunden werden kann und die Entwickler wycomcos mit dieser Sprache bereits vertraut sind. Wie auch die bestehenden Webapplikationen wird bei der Implementation das Framework Laravel¹⁴ 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 objekt-orientierten Anwendung in eine relationale Datenbank überführt. Das Framework bringt von Hause aus ein Authentifizierungspaket mit, welches 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).

¹³vgl. PHP

¹⁴vgl. OTWELL [a]

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 unbefugtem Zugriff. Die Ressourcen von den Benutzern (als *Resource Owner* bezeichnet) verwaltet ein *Resource Server*, der nur autorisierte Anfragen zulässt. Ein vertrauenswürdiger *Authorization Server* authentifiziert den Benutzer und holt dessen Autorisierung für den Zugriff ein. Als Autorisierungsnachweis stellt der *Authorization 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 entwickelt und diese auch perspektivisch Kunden zur Verfügung gestellt werden 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 OAuth2 sendet keine Benutzerinformationen mit dem AccessToken mit, wie es in SAML der Fall ist, so erhält ein Angreifer keinen direkten Zugang zu den Nutzerdaten¹⁷.

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.

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 Nutzernamen und Passwort ein und sendet sie per POST Request an den SSO-Server. Das hat den Nachteil, dass der Client die Daten bearbeitet.

¹⁵vgl. HARDT

¹⁶vgl. DARFK

¹⁷vgl. WOLFF [2015]

4 Entwurfsphase

2. mit dem **Autorisationsansatz** lenkt der Client den Nutzer auf eine View vom **SSO**-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 **SSO**-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, dies stellt ein zu umgehendes Sicherheitsrisiko dar. Der Client Credentialsansatz passt nicht zu den Anforderungen von wycomco an das Projekt, es soll eindeutig ein Nutzer authentifiziert werden, nur der Client ist nicht ausreichend. Nach Prüfung der Sicherheitsbestimmungen entschied sich die Autorin für den Autorisationsansatz. Der Nutzer kann den Zugriff gestatten und sendet seine Nutzerdaten ausschließlich über den **SSO**-Server, damit erhöht sich durch die Verwendung eines Authorizationscodes die Sicherheit und der Client ist nicht in die Authentifizierung eingebunden.

4.5 Architekturdesign

Beim Design wurde dem **MVC** Muster gefolgt, welches Laravel mit sich bringt. Die Anwendung wird in 3 Komponenten aufgespaltet 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. 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 richtigen User zurückgibt. Dieser loggt den Benutzer ein und die View zeigt den erfolgreichen Login Prozess an.

4 Entwurfsphase

4.6 Entwurf der Benutzeroberfläche

Das Laravel Passport Paket bringt eine beispielhafte Oberfläche mit, in der alle relevanten Anforderungen Anwendung finden. Mockups sind dazu auf [A.6 \(Benutzeroberfläche Mockup\)](#) zu sehen. Timy stellt hier einen Client dar. Es handelt sich um eine Zeiterfassungssoftware, die die Autorin im Rahmen ihrer Ausbildung implementiert hat und die auch über Single-Sign-On laufen soll. 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. Bei [A.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.

Tabelle 6: Entitäten

4.8 Geschäftslogik

Da wy-connect mit Hilfe von Laravel umgesetzt werden soll, lag es nahe bei der Implementierung auf das Passport Paket zurückzugreifen.¹⁸ In Kapitel [4.3 \(Wahl des SSO-Protokolls\)](#) wurden bereits die verschiedenen Rollen im OAuth2 erläutert. 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 Angriffe durch z. B. Phishing verhindert werden können. Alle redirect URIs müssen TLS verschlüsselt sein, da der

¹⁸vgl. [OTWELL \[b\]](#)

4 Entwurfsphase

Dienst nur URLs akzeptiert die mit "https"beginnen. So soll verhindert werden, dass Token während des Autorisierungsprozesses abgefangen werden.

Client ID und Secret - Nach der Registrierung der Anwendung wird eine *Client ID* und ein *Client Secret* bereit gestellt. Die Client ID wird als öffentliche Information betrachtet, wird zur Erstellung von Login Seiten genutzt und stellt die öffentliche Kennung des Clients dar. Das Client Secret muss vertraulich behandelt werden und wird genutzt um vom Authorization Server ein Access Token zu erlangen. Das Secret ist ausschließlich der Anwendung und dem Autorisierungsserver bekannt und besteht aus einer zufälligen Zahlen-Buchstaben-Reihe.

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 «Zulassen»klickt, leitet der Dienst ihn mit einem Autorisierungscode zurück zum Client. Ein Beispiel für eine Antwort 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 Statuswert dem vom vorherigen Request entspricht. Dieser Zustandswert wird verglichen, um sicherzustellen, dass er mit dem Ursprungswert übereinstimmt. Der Statuswert kann normalerweise in einem Cookie oder einer Sitzung gespeichert und verglichen werden, vorausgesetzt der Benutzer kehrt zurück. Dies stellt sicher, dass der Umleitungsendpunkt nicht dazu verleitet werden kann, willkürliche Autorisierungscode auszutauschen.

Mittels des Autorisierungscode kann nun der Client mittels folgendem Aufruf ein Access Token erlangen:

```
POST https://wy-connect.wycomco.de/token
grant_type=authorization_code&
```

4 Entwurfsphase

```
code=def502009ae50a0d3dc123a13c16d3281a46e5301eec50483c3bb4e7 [...] &
redirect_uri=https://wy-connect-client.test/login/wyconnect/callback&
client_id=1&
client_secret=ZKKn5QypcIs04iAh2dTdFAWJ47bHinbvMoLCYFP2
```

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 einloggen¹⁹.

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

4.9 Pflichtenheft

Ein Beispiel für das auf dem Lastenheft (siehe Kapitel ??: ??) aufbauende Pflichtenheft ist im Anhang [A.12: Pflichtenheft \(Auszug\)](#) auf Seite [xiv](#) zu finden.

4.10 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

¹⁹vgl. [TORSTEN LODDERSTEDT \[2013\]](#)

5 Implementierungsphase

5.1 Aufsetzen des Grundgerüsts

Zu Beginn wurde ein 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 benötigte 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 Middlewares zur Authentifizierung sowie zum **CSRF**-Schutz (Schutz gegen Anfragenfälschung²⁰) 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 können intuitiv benannt werden und werden so in der View aufgerufen. Unter **A.7** 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.²¹ und die erforderlichen Tabellen wurde mit dem laraveigenen Befehlen migriert.²²

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 xv 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 Anhang **5.2: Middleware** auf dieser Seite) geregelt sind. Alle Routen mit der Middleware auth können nur eingeloggte User benutzen, guest ist nur für nicht eingeloggte Nutzer offen.

²⁰vgl. **CSR**

²¹Befehl zur Installation des Pakets: `composer require laravel/passport` vgl. **COMPOSER**

²²Befehl zur Migration: `php artisan migrate` vgl. **OTWELL** [a]

5 Implementierungsphase

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, fordert den Nutzer auf sich einzuloggen und leitet ihn schließlich auf eine View, auf der er der Autorisierung der Anwendung zustimmen kann. Die Antwort verarbeiten der *ApproveAuthorizationController* und der *DenyAuthorizationController*, die bei fehlerhaftem state Code eine Fehlermeldung zurückgeben, oder über eine Funktion Passports 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 Force²³ umsetzt. Ein Ausschnitt der genutzten Klassen befindet sich im Anhang Anhang A.15: Ausschnitt des Klassendiagramms vom Client auf Seite xvii.

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 Authentifizierungsserver 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 übergebenen Userdaten. Im LoginController muss dann nur noch der OAuth2 User aus dem Single-Sign-On in das Usermodel des Clients umgewandelt werden und das Einloggen erfolgen. 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. Ein Ausschnitt der genutzten Klassen befindet sich im Anhang Anhang ??: ?? auf Seite ??.

²³vgl. HARDT

5.6 Zwischenstand

Tabelle 8 zeigt den Zwischenstand nach der Implementierungsphase.

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 Ausschnitt der Testsuite findet sich im Codeanhang, die Konsolenausgabe ist unter Anhang ??: ?? auf Seite ?? Alle Tests lieferten ein positives Ergebnis und das Projekt konnte über den firmenintern Deployment Prozess ausgerollt werden. Abschließend wurde der Code von der Clientanwendung timy in den master Branch des Projektes gemerged und damit automatisiert auf den Webserver gespielt. Damit konnte produktiv der SSO-Server mit der ersten Client Anwendung laufen. Diese neue Funktionalität wurde vor dem Teamleiter präsentiert, der anschließend das Projekt abnahm.

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 so weit möglich 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 PHPDoc²⁴ wurde eine Entwicklerdokumentation automatisch generiert. Bei dieser Entwicklerdokumentation handelt es sich um eine detaillierte Beschreibung der Klassen, die in

²⁴Vgl. [PHPDoc](#)

8 Fazit

der Anwendung verwendet werden. Diese Dokumentation soll dem Entwickler als Übersicht und Nachschlagewerk dienen. Im Anhang [A.17: Screenshot der Entwicklerdokumentation](#) auf Seite [xix](#) findet sich eine Abbildung dazu.

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 ergebenden 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 auf Grund von Mehraufwand bei der Recherche des OAuth2 Protokolls und gemindertem Aufwand bei der Implementierung auf Grund des verwendeten Frameworks zu Stande.

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 umfangreiche Erfahrungen auf dem Gebiet der Planung und Umsetzung sammeln. Besonders deutlich wurde wie wichtig eine gute Recherche ist, da ohne

8 Fazit

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 dieses Werkzeug nicht bedient werden.

8.3 Ausblick

Das Projekt findet aktuell Anwendung in den ersten Clientanwendungen und soll künftig noch auf Dienste ausgeweitet werden, 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, oder eine Anbindung an die AD einzuführen.

Literaturverzeichnis

CSR

Laravel CSRF Protection. : *Laravel CSRF Protection*, <https://laravel.com/docs/5.6/csrf>, Abruf: 13.03.2018

PHP

PHP Manual. : *PHP Manual*, <http://php.net/manual/de/index.php>, Abruf: 12.03.2018

composer

COMPOSER: *Composer*, <https://getcomposer.org/>, Abruf: 14.03.2018

Darfk

DARFK: Single Sign-On Kerberos vs SAML vs OAuth2. <http://dafrk-blog.com/de/sso-kerberos-saml-oauth-sap/>, Abruf: 13.03.2018

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/>, Abruf: 13.03.2018

Git

GIT: *Git Documentation*, <https://git-scm.com/documentation>, Abruf: 16.03.2018

GitLab

GITLAB, Pages: *GitLab*, <https://docs.gitlab.com>, Abruf: 16.03.2018

Hardt

HARDT, D.: *The OAuth 2.0 Authorization Framework*, <https://tools.ietf.org/html/rfc6749>, Abruf: 12.03.2018

Hübscher u. a. 2007

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

Intersoft Consulting Services 2016

INTERSOFT CONSULTING SERVICES, AG: Single Sign-on: Tipps beim Einsatz der Login-Technologie. (2016). <https://www.datenschutzbeauftragter-info.de/single-sign-on-tipps-beim-einsatz-der-login-technologie/>, Abruf: 14.03.2018

www.schnelle online.info

ONLINE.INFO *www.schnelle*: *Arbeitstage 2018*. <https://www.schnelle-online.info/Arbeitstage/Anzahl-Arbeitstage-2018.html>, Abruf: 15.03.2018

Oracle

ORACLE: *MySQL Documentation*, <https://dev.mysql.com/doc/>, Abruf: 14.03.2018

Otwell a

OTWELL, Taylor: *Laravel Documentation*, <http://www.laravel.com>, Abruf: 2018-03-15

Otwell b

OTWELL, Taylor: *Laravel Passport*, <https://laravel.com/docs/master/passport>, Abruf: 14.03.2018

Otwell c

OTWELL, Taylor: *Laravel Socialite*, <https://laravel.com/docs/5.5/socialite>, Abruf: 16.03.2018

PHPDoc

PHPDOC: *PHPDoc*, <https://www.phpdoc.org/>, Abruf: 14.03.2018

Schonschek 2015

SCHONSCHKEK, Oliver: Worauf es bei SSO Lösungen ankommt. (2015). <https://www.computerwoche.de/a/worauf-es-bei-sso-loesungen-ankommt,2539134>, Abruf: 16.03.2018

Torsten Lodderstedt 2013

TORSTEN LODDERSTEDT, Jochen H.: Flexible und sichere Internetdienste mit OAuth 2.0. (2013). <https://www.heise.de/developer/artikel/Flexible-und-sichere-Internetdienste-mit-OAuth-2-0-2068404.html>, Abruf: 16.03.2018

Univention 2016

UNIVENTION, GmbH: Kurz erklärt: Einmalige Anmeldung per Single Sign-on. (2016). <https://www.univention.de/2016/12/einmalige-anmeldung-per-single-sign-on/>, Abruf: 13.03.2018

Wikipedia 2018

WIKIPEDIA: Testgetriebene Entwicklung. (2018). https://de.wikipedia.org/wiki/Testgetriebene_Entwicklung, Abruf: 12.03.2018

Wolff 2015

WOLFF, Eberhard: *Microservices: Grundlagen flexibler Softwarearchitekturen*. 2015

Eidesstattliche Erklärung

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

A Anhang

- 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 Git²⁵-Repository verwendet werden.
 - 3.4. Die Anwendung soll in der Programmiersprache PHP²⁶ mittels des Frameworks Laravel²⁷ umgesetzt werden.
 - 3.5. Der Server soll auf dem firmeninternen Webserver gehostet werden.
 - 3.6. Bei Einsatz einer MySQL-Datenbank²⁸ soll der firmeninterne MySQL Server Verwendung finden.
 - 3.7. Der Einrichtungsprozess für neue Clienten muss dokumentiert werden.

[...]

²⁵vgl. GIT

²⁶vgl. PHP

²⁷vgl. OTWELL [a]

²⁸vgl. ORACLE

A.4 Use-Case-Diagramm ohne SSO

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

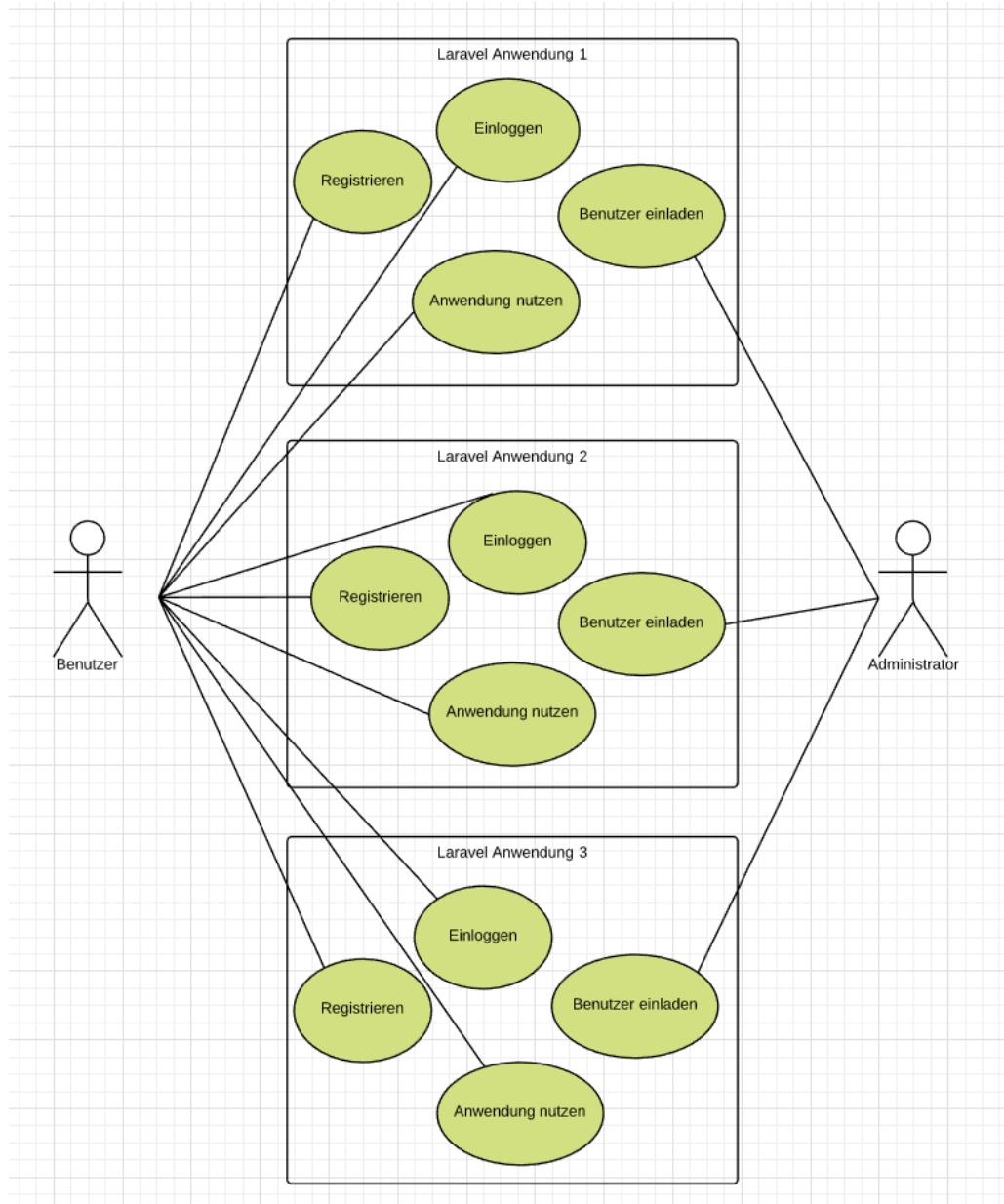


Abbildung 3: Use-Case-Diagramm ohne SSO

A.5 Use-Case-Diagramm mit SSO

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

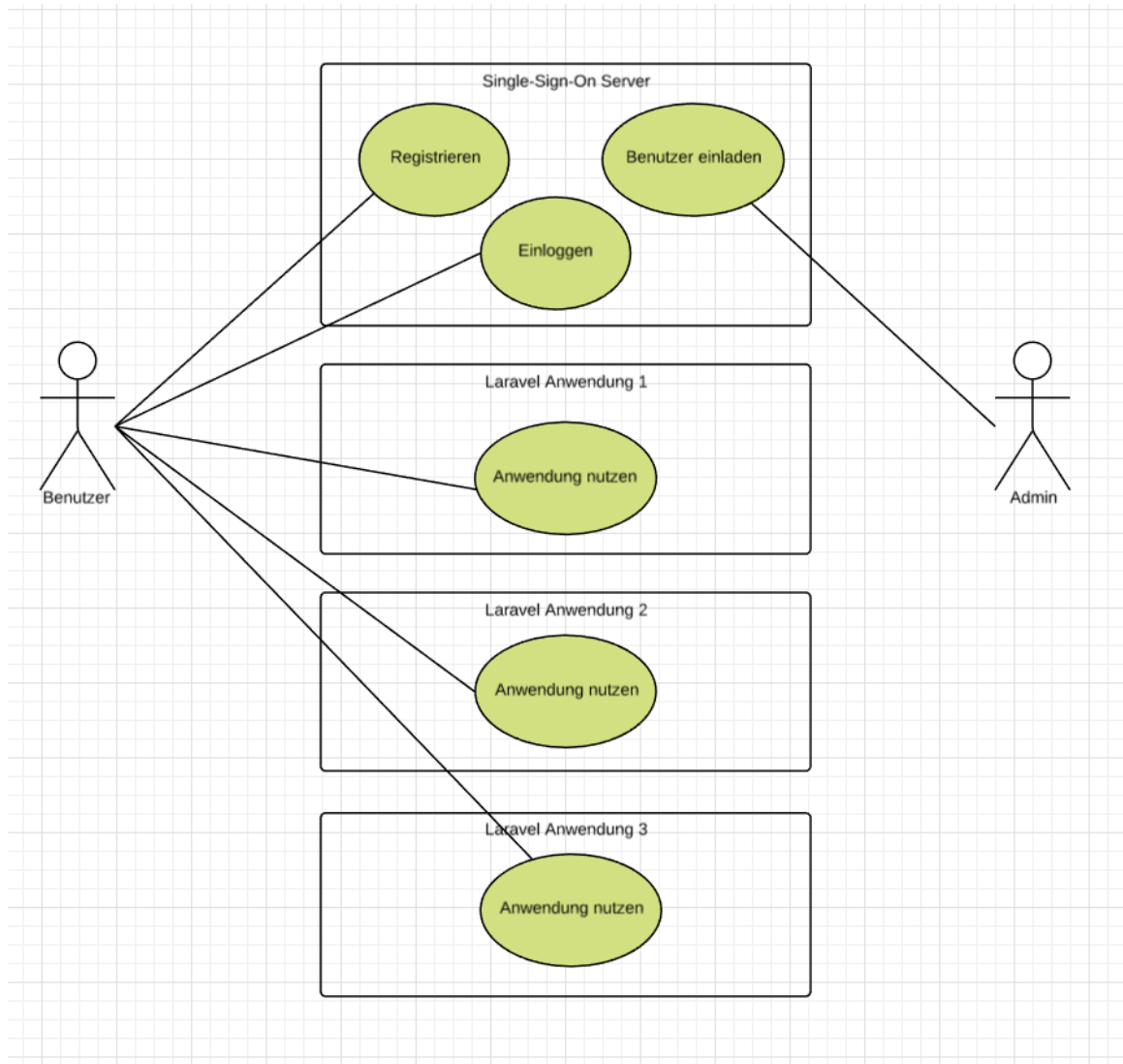


Abbildung 4: Use-Case-Diagramm mit SSO

A.6 Benutzeroberfläche Mockup

Das folgende Template beschreibt den Anmeldeprozess am SSO-Server.

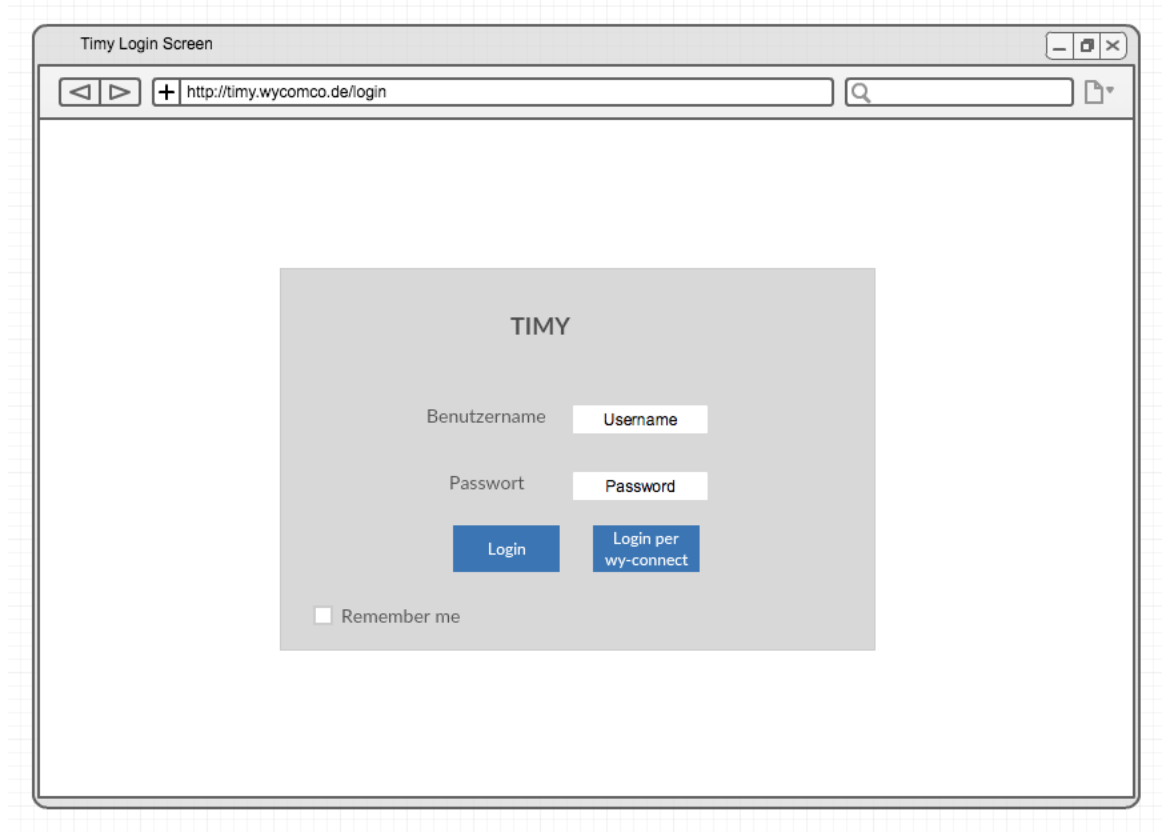


Abbildung 5: Loginscreen Mockup Timy

A.7 Benutzeroberfläche Screenshots

Der folgende Screenshot beschreibt die Loginseite des SSO.

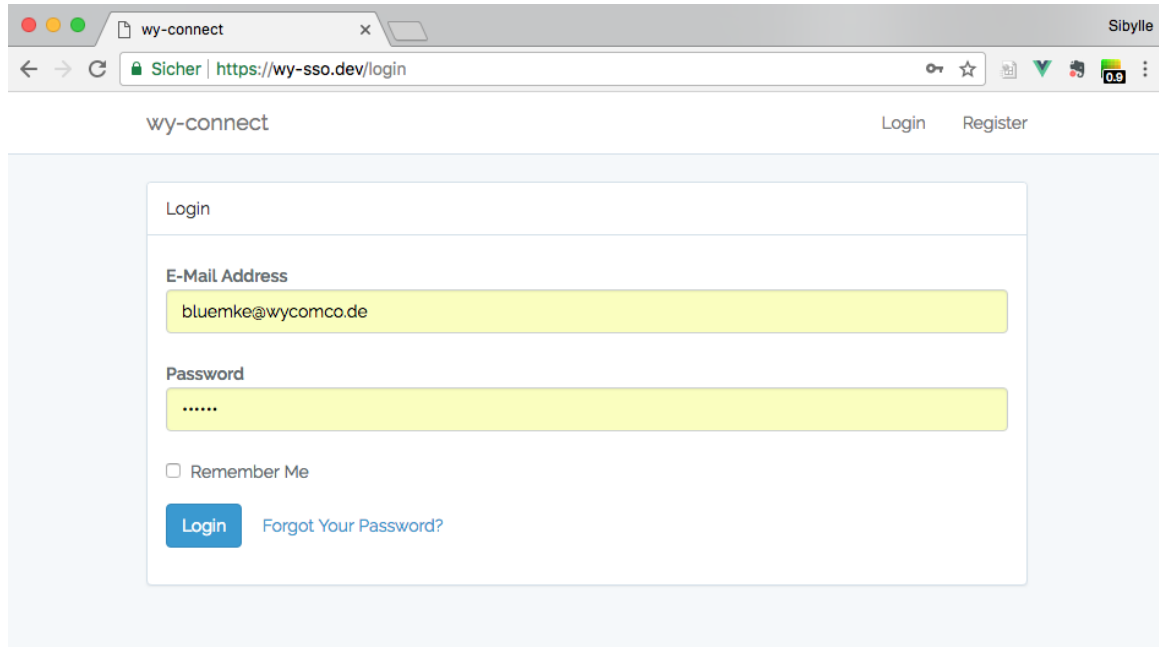


Abbildung 6: Loginseite des SSO-Servers

A Anhang

Der folgende Screenshot beschreibt die Benutzeroberfläche des SSO.

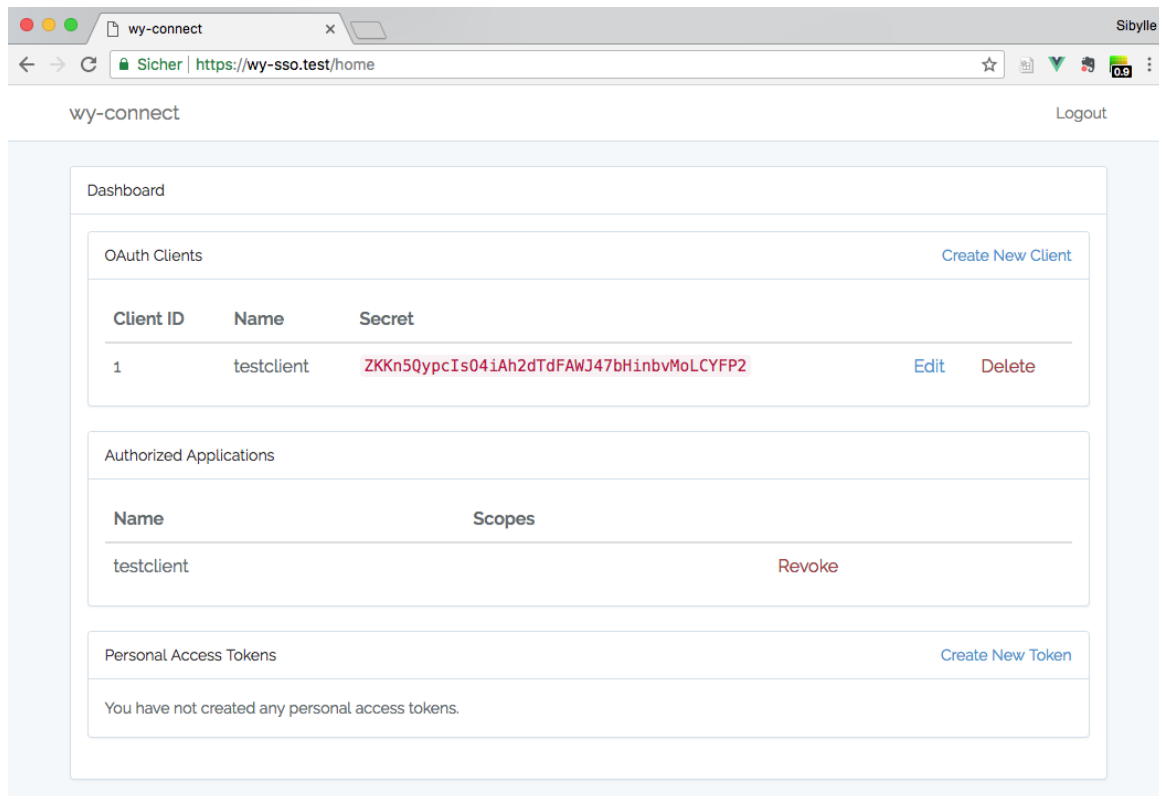


Abbildung 7: Benutzeroberfläche des SSO-Servers

A Anhang

Der folgende Screenshot beschreibt den Anmeldeprozess am Client mit **SSO**.

Timy

Login

Login

E-Mail Address

Password

☐ Remember Me

LOGIN FORGOT YOUR PASSWORD?

LOGIN ÜBER WYCONNECT

Abbildung 8: Login am Client Timy

A.8 Entity Relationship Modell

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

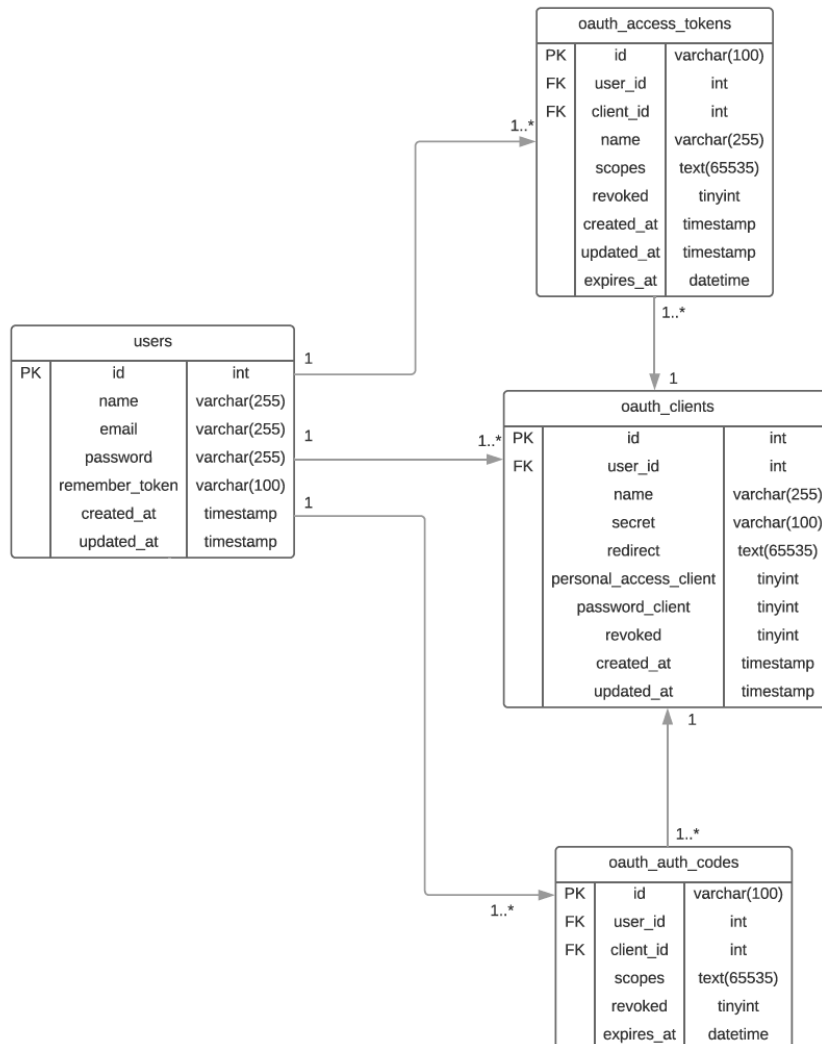


Abbildung 9: ERM

A.9 Amortisationsdiagramm

Das folgende Diagramm zeigt die Amortisation

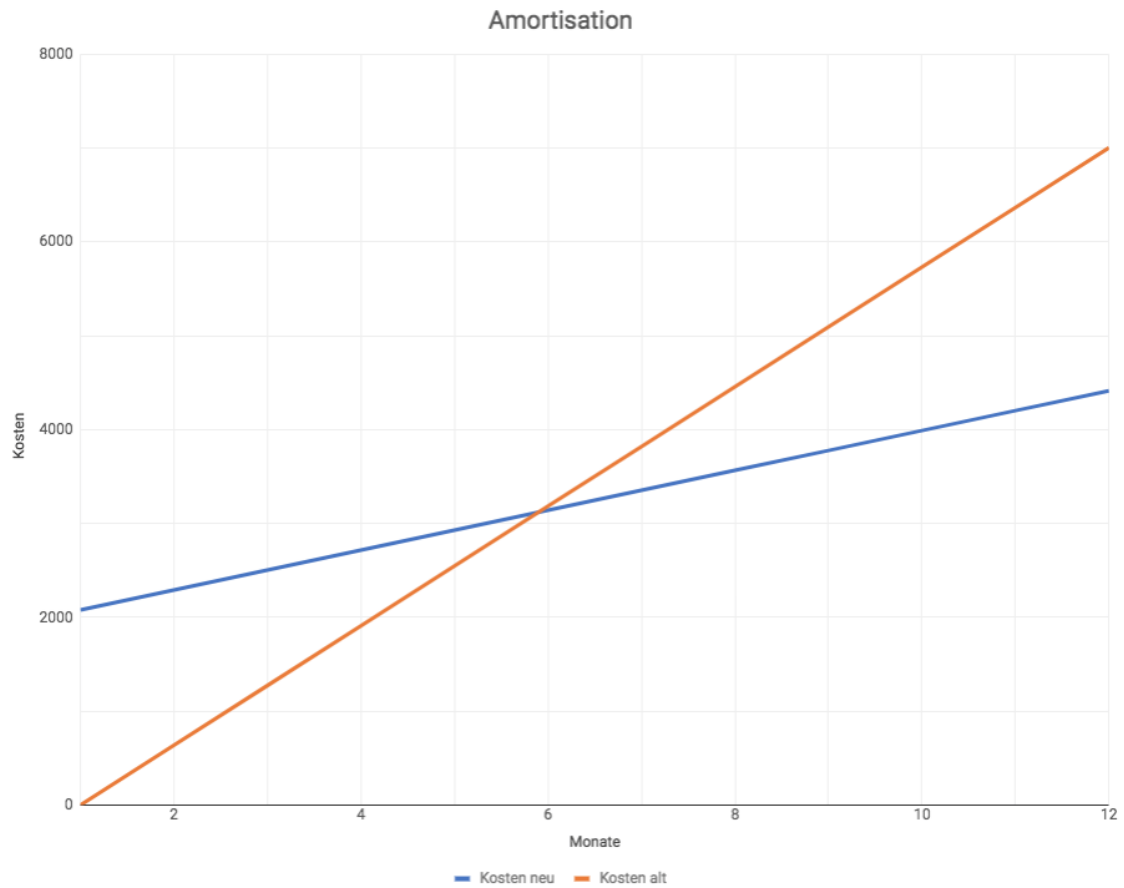


Abbildung 10: Amortisations-Diagramm

A.10 OAuth2 Flow

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

OA2UTH2 AUTHORIZATION CODE GRANT

wycomco GmbH | January 29, 2018

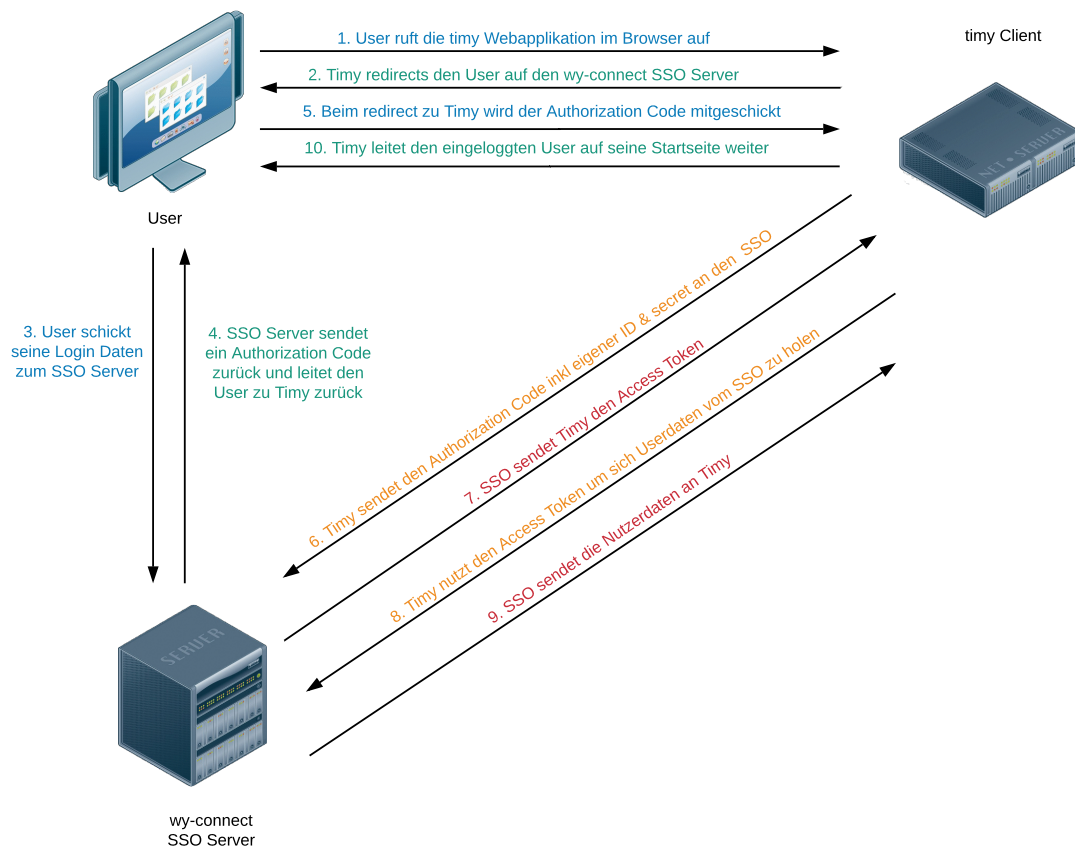


Abbildung 11: OAuth2 Flow

A.11 Sequenzdiagramm OAuth2

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

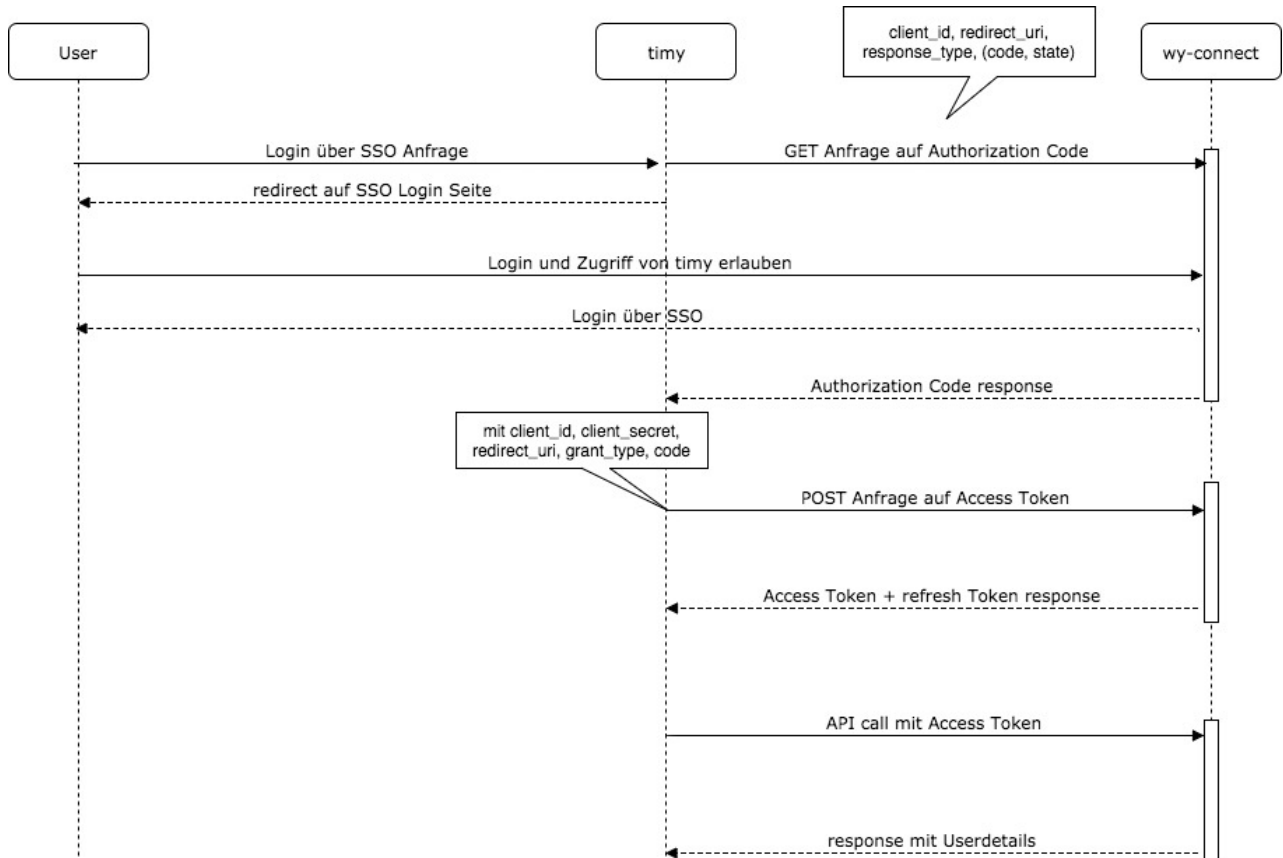


Abbildung 12: Sequenzdiagramm OAuth2

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 ²⁹ genutzt.

- die Autorisierung wird hierbei über die API vorgenommen
- die Authentifizierung erfolgt ausschließlich über den Server
- zur Umsetzung wird das Laravel Paket Passport³⁰ 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 Socialite³¹ 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 Nutzerdaten 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-Server³² genutzt.

[...]

²⁹vgl. **HARDT**

³⁰vgl. **OTWELL** [b]

³¹vgl. **OTWELL** [c]

³²vgl. **GITLAB**

A.13 Routen innerhalb des SSO-Servers

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

```
➔ my-ssso 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 13: Routen des SSO-Servers

A Anhang

A.14 Ausschnitt des Klassendiagramms vom SSO-Server

Das folgende UML Diagramm zeigt einen Ausschnitt aus dem Klassendiagramm des SSO-Servers.

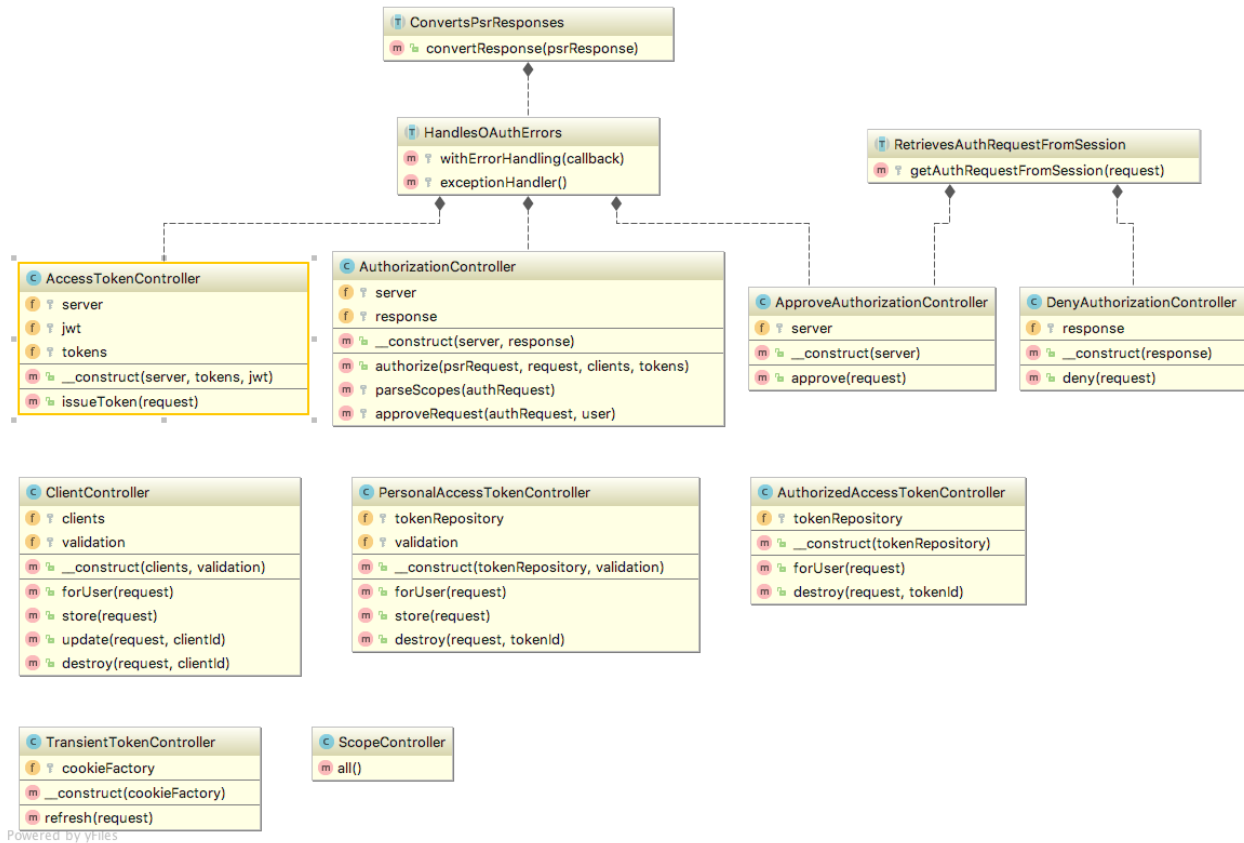


Abbildung 14: Ausschnitt aus dem UML Diagramm des SSO-Servers

```

classDiagram
    class Provider {
        <> redirect()
        <> user()
    }
    class AbstractProvider {
        <> request
        <> httpclient
        <> clientid
        <> redirecturl
        <> scopeseparator
        <> encodingtype
        <> clientsecret
        <> guzzle
        <> stateless
        <> parameters
        +__construct(request, clientid, clientSecret, redirectUrl, guzzle)
        +getAuthUri(state)
        +getTokenUri()
        +getUserByToken(token)
        +mapUserToObject(user)
        +redirect()
        +buildAuthUriFromBase(uri, state)
        +getCodeFields(state)
        +formatScopes(scopes, scopeSeparator)
        +user()
        +userFromToken(token)
        +hasInvalidState()
        +getAccessTokenResponse(code)
        +getTokenFields(code)
        +getCode()
        +setScopes(scopes)
        +getScopes()
        +redirectUri[url]()
        +getHttpClient()
        +setHttpClient(client)
        +setRequest(request)
        +usesState()
        +isStateless()
        +stateless()
        +getState()
        +with(parameters)
    }
    class Exception {
        <> code
        <> file
        <> line
        <> message
        +__construct(message, code, previous)
        +__clone()
        +getMessage()
        +getCode()
        +getFile()
        +getLine()
        +getTrace()
        +getPrevious()
        +getTraceAsString()
        +__toString()
        +__wakeup()
    }
    class LogicException
    class InvalidArgumentException
    class InvalidStateException
    class ProviderInterface {
        <> redirect()
        <> user()
    }
    class WyconnectProvider {
        +getAuthUri(state)
        +getTokenUri()
        +getTokenFields(code)
        +getUserByToken(token)
        +mapUserToObject(user)
        +getRequestOptions()
    }
    class ArrayAccess {
        <> offsetExists(offset)
        <> offsetGet(offset)
        <> offsetSet(offset, value)
        <> offsetUnset(offset)
    }
    class User {
        <> getId()
        <> getNickname()
        <> getName()
        <> getEmail()
        <> getAvatar()
        <> getUserId()
        <> getNickname()
        <> getName()
        <> getEmail()
        <> getAvatar()
        <> setRaw(user)
        <> map(attributes)
        <> offsetExists(offset)
        <> offsetGet(offset)
        <> offsetSet(offset, value)
        <> offsetUnset(offset)
    }
    class AbstractUser {
        <> nickname
        <> name
        <> id
        <> avatar
        <> user
        <> email
        <> getId()
        <> getNickname()
        <> getName()
        <> getEmail()
        <> getAvatar()
        <> setRaw(user)
        <> map(attributes)
        <> offsetExists(offset)
        <> offsetGet(offset)
        <> offsetSet(offset, value)
        <> offsetUnset(offset)
    }
    class User {
        <> expiresIn
        <> token
        <> refreshToken
        <> setToken(token)
        <> setRefreshToken(refreshToken)
        <> expiresIn(expiresIn)
    }
    class ServiceProvider {
        <> app
        <> defer
        <> publishes
        <> publishGroups
        +__construct(app)
        +mergeConfigFrom(path, key)
        +loadRoutesFrom(path)
        +loadViewsFrom(path, namespace)
        +loadTranslationsFrom(path, namespace)
        +loadJsonTranslationsFrom(path)
        +loadMigrationsFrom(path)
        +publishes(paths, group)
        +ensurePublishArrayInitialized(class)
        +addPublishGroup(group, paths)
        +pathsToPublish(provider, group)
        +pathsForProviderOrGroup(provider, group)
        +pathableProviders()
        +publishableGroups()
        +commands(commands)
        +provides()
        +when()
        +isDeferred()
    }
    class SocialiteServiceProvider {
        <> defer
        +register()
        +provides()
    }
    class Manager {
        <> app
        <> customCreators
        <> drivers
        +__construct(app)
        +getDefaultDriver()
        +driver(driver)
        +createDriver(driver)
        +callCustomCreator(driver)
        +extend(driver, callback)
        +getDrivers()
        +__call(method, parameters)
    }
    class Factory {
        +driver(driver)
    }
    class SocialiteManager {
        +with(driver)
        +createWyconnectDriver()
        +createGithubDriver()
        +createFacebookDriver()
        +createGoogleDriver()
        +createLinkedInDriver()
        +createBitbucketDriver()
        +buildProvider(provider, config)
        +createTwitterDriver()
        +formatConfig(config)
        +formatRedirectUri(config)
        +getDefaultDriver()
    }
    Provider --|> AbstractProvider
    ProviderInterface ..|> AbstractProvider
    ProviderInterface ..|> WyconnectProvider
    Exception --|> LogicException
    Exception --|> InvalidArgumentException
    Exception --|> InvalidStateException
    AbstractProvider --|> WyconnectProvider
    User --|> AbstractUser
    SocialiteServiceProvider --|> ServiceProvider
    Manager --|> SocialiteManager
    SocialiteManager --|> Factory

```

The UML class diagram illustrates the architecture of the Socialite PHP library. It features several main components:

- Provider Interface and Implementation:** The `Provider` interface defines methods like `redirect()` and `user()`. `AbstractProvider` implements these with abstract methods, while `WyconnectProvider` provides concrete implementations.
- User Management:** The `User` interface defines methods for retrieving user information. `AbstractUser` implements these as abstract methods, and the `User` class provides the actual implementation.
- Service Providers:** The `ServiceProvider` interface defines methods for managing the application's configuration and publishing assets. `SocialiteServiceProvider` implements these methods.
- Manager and Factory:** The `Manager` class manages the collection of providers. `SocialiteManager` extends it, adding specific methods for creating different types of providers. The `Factory` class is used to instantiate drivers from the manager.
- Exceptions:** A hierarchy of exceptions is defined, starting with `Exception`, which includes `LogicException`, `InvalidArgumentException`, and `InvalidStateException`.

Relationships are shown through inheritance (solid arrows), interfaces (dashed arrows), and associations (solid lines). The diagram uses color-coded boxes to represent different types of elements: yellow for classes, light blue for interfaces, and light green for exceptions.

Abbildung 15: Ausschnitt aus dem UML Diagramm des Clients

A.16 Ausgabe der Unit- und Featuretests

Die folgende Konsolenausgabe zeigt die erfolgreichen Test mit PHPUnit.

```
→ timy git:(socialite-provider) ./vendor/bin/phpunit
PHPUnit 6.5.7 by Sebastian Bergmann and contributors.

..... 63 / 157 ( 40%)
..... 126 / 157 ( 80%)
..... 157 / 157 (100%)

Time: 19.05 seconds, Memory: 50.00MB

OK (157 tests, 255 assertions)
→ timy git:(socialite-provider) █
```

Abbildung 16: Konsolenausgabe der PHPUnit Tests

A.17 Screenshot der Entwicklerdokumentation

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

The screenshot displays the API Documentation for the `\Laravel\Passport\TokenRepository` class. The interface is divided into three main sections:

- Left Sidebar:** A navigation tree showing the project structure, including `Laravel`, `Passport`, `Bridge`, `Console`, `Events`, `Exceptions`, `Guards`, `Http`, `Controllers`, `Middleware`, and `HasApiTokens`.
- Main Content Area:**
 - Summary:** A table showing the class's methods, properties, and constants. The methods section lists `create()`, `find()`, `findForUser()`, `forUser()`, `getValidToken()`, `save()`, `revokeAccessToken()`, `isAccessTokenRevoked()`, and `findValidToken()`. The properties and constants sections indicate that no public, protected, or private properties or constants were found.
 - Methods:** A detailed view of the `create()` method, showing its signature `create(array $attributes) : \Laravel\Passport\Token` and its description: "Creates a new Access Token." The parameters section shows `array $attributes` and the return type `\Laravel\Passport\Token`.
 - find() Method:** A detailed view of the `find()` method, showing its signature `find(string $id) : \Laravel\Passport\Token` and its description: "Get a token by the given ID." The parameters section shows `string $id`.
- Right Sidebar:** A dark sidebar containing information about the file (`src/TokenRepository.php`), package (`Default`), and class hierarchy (`\Laravel\Passport\TokenRepository`). It also includes sections for tags, which are currently "None found".

Abbildung 17: Benutzerdokumentation