



BERENBERG
PRIVATBANKIERS SEIT 1590

PROJEKTDOKUMENTATION

Projektdokumentation **LSC - Live Support Chat**

Niklas Grieger

Hamburg, 22.Mai 2017

LSC – Live Support Chat

Fachinformatiker Fachrichtung Anwendungsentwicklung
Abschlussprüfung Sommer 2017

Vorgelegt von
Niklas Grieger
Carsten-Reimers-Ring 120
22175 Hamburg



Inhalt

1. Einleitung	1
1.1. Vorstellung des Betriebes	1
1.2. Vorstellung des Entwicklers	1
1.3. Vorstellung des Kunden	1
1.4. Entwicklungsumgebung	1
1.5. Projektziel	2
1.6. Projektbegründung	2
1.7. Projektschnittstellen	2
1.8. Projektabgrenzung	2
2. Projektplanung	3
2.1. Zeitplan	3
2.2. Ressourcenplanung	3
3. Analysephase	4
3.1. Ist-Analyse	4
3.2. Wirtschaftlichkeitsanalyse	4
3.2.1. „Make of Buy“ – Entscheidung	4
3.2.2. Gründe für die Neuentwicklung	4
3.2.3. Projektkosten	5
3.2.4. Kosten / Nutzen	5
3.3. Lastenheft	5
4. Entwurfsphase	6
4.1. Zielplattform	6
4.2. Qualitätssicherung	6
4.3. Schnittstellen	6
4.4. Programmkonzept	7
4.5. Pflichtenheft	8
5. Realisierungsphase	8
5.1. Datenbank	8
5.2. Back-End	9
5.3. Front-End	10
5.4. Sicherheit	10
6. Testphase	11



7. Dokumentation	12
7.1. Projektdokumentation.....	12
7.2. Entwicklerdokumentation	12
7.3. Kundendokumentation	12
8. Quellennachweise	13
9. Glossar.....	13
10. Anhang.....	14
10.1. Verwendete Ressourcen	14
10.2. Diagramme	15
10.2.1. IST-Zustand.....	15
10.2.2. SOLL-Zustand	16
10.2.3. Aktivitätsdiagramm	17
10.2.4. Anwendungsfalldiagramm.....	18
10.2.5. ERM	19
10.2.6. Klassendiagramm – Database Code First.....	20
10.2.7. Klassendiagramm – GUI.....	21
10.3. Entwicklerdokumentation (Auszug)	22
10.4. Kundendokumentation	23
10.4.1. Einleitung.....	23
10.4.2. Masken und Funktionalitäten	23
10.5. Testprotokolle.....	30
10.6. Code-Ausschnitte.....	35
10.6.1. Datenbank - Das Code First Prinzip.....	35
10.6.1. Backend	37
10.6.2. Front End.....	47

Lastenheft

Pflichtenheft



1. EINLEITUNG

Die folgende Projektdokumentation wurde im Rahmen des IHK Abschlussprojektes erstellt, welches der Autor während seiner Ausbildung zum Fachinformatiker Fachrichtung Anwendungsentwicklung durchgeführt hat.
Der Ausbildungsbetrieb ist Berenberg.

1.1. Vorstellung des Betriebes

Berenberg wurde 1590 gegründet und ist heute eine der ältesten Privatbanken der Welt. Über 250 Mitarbeiter sind in den Bereichen IT und IT-Operations beschäftigt. Der überwiegende Teil der eingesetzten Software wird selbst entwickelt.

1.2. Vorstellung des Entwicklers

Niklas Grieger, geb. am 02.06.1994 in Hamburg macht eine Ausbildung als Fachinformatiker mit Schwerpunkt Anwendungsentwicklung.

1.3. Vorstellung des Kunden

Kunde dieses Projektes ist das IT-Management als Lenkungsausschuss bei Berenberg.

1.4. Entwicklungsumgebung

Zur Erstellung der Anwendung wurde ein Fat-Client mit folgenden (relevanten) Hardwarekomponenten benutzt (dieser FAT-Client war sowohl Client als auch Server):

16 GB Ram

Windows Server 2008 R2 Standard Betriebssystem

Intel® Xeon® CPU E3-1225 V5 @ 3.30GHz Prozessor

NVIDIA NVS 310

119 GB NTFS Festplatte

Die Entwicklung des Projektes wurde ausschließlich mit der Software Microsoft Visual Studio Professional 2015 umgesetzt.

Die Inline-Dokumentation und die Testverfahren während der Entwicklung geschahen ebenfalls über diese Software.

Grafiken und Diagramme wurden erst per Hand entworfen und dann mit Hilfe von Windows Paint und/oder mit Hilfe von Microsoft Visio oder der Web-Anwendung www.draw.io umgesetzt.



1.5. Projektziel

Ziel des Projektes ist die Implementierung eines Live Support Chats, zur schnellen und direkten Kommunikation zwischen Benutzer und Support.

1.6. Projektbegründung

Durch diesen Chat wird es dem Benutzer möglich sein bei Fragen zu einer bestimmten Kategorie auf der Intranetseite direkt einen zuständigen Support zu kontaktieren. Durch diese Kommunikationsmöglichkeit werden u.a. folgende Szenarien abgedeckt:

- Der Hilfesuchende weiß nicht wer für das Thema zuständig ist bzw. weiterhelfen kann
- Der Hilfesuchende muss den Support anrufen, doch dieser ist außerhalb der Supportzeiten nicht erreichbar
- Der Hilfesuchende ruft jemanden an, der für das Thema nicht zuständig ist

Das Resultat ist eine Zeitersparnis bei dem Support sowie bei dem Hilfesuchenden.

1.7. Projektschnittstellen

Die Benutzer der Applikation sind alle Mitarbeiter der Firma Berenberg. Unter anderem ist folgendes gewährleistet:

Um den Chat nutzen zu können muss dem Benutzer eine Mitarbeiter ID zugeordnet sein. Diese Mitarbeiter ID wird beim Initialen Laden der Seite an die Chat Applikation weitergeleitet.

Anhand dieser ID werden Daten, wie z.B. Name und bisheriger Chatverlauf, in eine View gerendert und an die Intranetseite als Partial View zurückgegeben.

Außerdem ist die Applikation nur im Zusammenhang mit der Intranetseite aufzurufen.

1.8. Projektabgrenzung

Das Projekt soll nicht die bestehenden Chat Anwendungen in der Firma erweitern, sondern als neuentwickelter Live Support Chat agieren.

Das Gesamte Projekt wurde von dem Entwickler selbstständig durchgeführt und es gab keine Fremdleistungen.

Bei anderen „Global Chats“ wie z.B. Microsoft Lync müsste unter den Teilnehmern erst der richtige Ansprechpartner gefunden werden und dies kann wiederum nur außerhalb des Chats erfolgen.

Bei dem Live Support Chat hingegen sind die Ansprechpartner in der Datenbank hinterlegt und die Nutzer können direkt den themenbezogenen Kontakt aufnehmen. Zudem kann dies direkt auf der Intranetseite geschehen, d.h. man muss kein weiteres Programm öffnen.



2. PROJEKTPLANUNG

2.1. Zeitplan

Projektphase	Dauer
Analyse	6 Stunden
• IST-Analyse	2 Stunden
• Kosten-Nutzen-Analyse	1 Stunden
• Lastenheft	3 Stunden
Entwurf	9 Stunden
• Aktivitätsdiagramm	1 Stunde
• Anwendungsfalldiagramm	1 Stunde
• ERM Diagramm	2 Stunden
• Pflichtenheft	5 Stunden
Realisierung	39 Stunden
• Datenbank	7 Stunden
• Back-End	17 Stunden
• Front-End	15 Stunden
Test	4 Stunden
Refactoring	2 Stunden
Dokumentation	10 Stunden
Gesamt	70 Stunden

Der Zeitplan hat sich gegenüber dem Antrag verändert, da die **Diagramme** schneller (jeweils 1 Stunde weniger) als gedacht erstellt waren und somit mehr Zeit für das **Pflichtenheft** blieb. Außerdem habe ich mehr Zeit in die **Realisierung** geplant, da es zu Fehlern im **Back-End** Bereich kam.

Zudem habe ich bei dem **Refactoring** 2 Stunden abgezogen, weil die Erstellung der Datenbank mehr Zeit in Anspruch genommen hat als Gedacht, da Kenntnisse über das Code First Prinzip erst gesammelt werden mussten.

2.2. Ressourcenplanung

Im Anhang „[Verwendete Ressourcen](#)“ sind alle Ressourcen aufgelistet, die für dieses Projekt eingesetzt wurden. Damit sind Hard- und Softwareressourcen sowie der Personaleinsatz gemeint.

Um die anfallenden Projektkosten möglichst gering zu halten, wurde bei der verwendeten Software darauf geachtet, dass diese nach Möglichkeit kostenlos zur Verfügung steht, oder Berenberg bereits die Lizenzen für diese besitzt.



3. ANALYSEPHASE

3.1. Ist-Analyse

Die Intranetseite von Berenberg steht jedem Mitarbeiter im Haus zur Verfügung. Aktuell muss der Benutzer für jedes Problem oder Frage bezüglich der Oberfläche über mehrere Wege kommunizieren um eine Antwort zu erhalten.

Eine Visuelle Darstellung befindet sich im Anhang unter „[IST-Zustand](#)“

Wie bereits im Abschnitt „[1.6 Projektbegründung](#)“ erwähnt, gibt es mehrere Szenarien die eintreten können, in denen der Hilfesuchende eventuell keine Hilfestellung bekommt bzw. bekommen kann oder andere Komplikationen auftreten könnten.

3.2. Wirtschaftlichkeitsanalyse

3.2.1. „Make of Buy“ – Entscheidung

Derweil existieren viele Angebote zu verschiedensten Chat Applikationen.

Meistens sind diese mit Kosten verbunden, oft sogar Kosten pro Nutzer.

Dies ist bei einer größeren Firma wie Berenberg mit hohen Kosten verbunden.

Außerdem ist ein häufiger Nachteil bei eingekaufter Software, dass diese nicht erweiterbar und nicht auf die Individuellen Wünsche anpassbar sind.

Aus diesen Gründen wird dieses Projekt intern umgesetzt.

3.2.2. Gründe für die Neuentwicklung

Die Chat Applikation setzt den Fokus an einer anderen Stelle an als das Altsystem.

Es soll den Arbeitsprozess und die Kommunikation zwischen Kunde und Support verbessern, während das alte System nur als „Global Chat“ dient.

Zudem gibt es unter anderem noch weitere vorteilhafte Funktionen gegenüber dem Altsystem:

- Ein schnell gefundener Support für eine Kategorie/Thema
- Echtzeit Benachrichtigungen über eingehende Mitteilungen
- modale Einbindung in die bestehen Intranetseite zur unabhängigen Weiterentwicklung des Chats
- Nachrichten werden gepuffert
- Schutz gegen Cross-Site Scripting
- Responsive Design



3.2.3. Projektkosten

Im Folgenden werden die anfallenden Projektkosten berechnet.

Es wird von folgenden Stundensätzen ausgegangen.

- Auszubildende/r: 15€
- Entwickler: 76€

Vorgang	Mitarbeiter	Zeit	Gesamt
Entwicklung	1x Auszubildender	70h	1.050€
Test	2x Auszubildender 2x Entwickler	4h	728€
Gesamt			1.778€

3.2.4. Kosten / Nutzen

Die Kosten für die Anwendung beschränken sich auf die Zeit der Entwicklung.
Das heißt es bestehen keine weiteren Lizenzkosten oder sonstige laufenden Kosten.

Die Projektkosten werden nicht von einem Kundenauftrag gedeckt, da es sich um ein internes Projekt handelt.

Das Resultat dieser Anwendung soll eine Zeitersparnis sein und soll die Arbeit des Fachbereichs effizienter gestalten.

Aufgrund dessen und der vielen Faktoren, die in der Zeitersparnis eine Rolle spielen, ist es nicht möglich genau zu errechnen, wann die Kosten gedeckt sind.

Der Nutzen ist anhand von Beispielen in Abschnitt „[1.6 Projektbegründung](#)“ beschrieben.

3.3. Lastenheft

Am Ende der Analysephase wurde ein Lastenheft erstellt. Dieses beinhaltet alle Anforderungen des Auftraggebers an die zu erstellende Anwendung.

Das komplette Lastenheft ist dem Anhang beigelegt.



4. ENTWURFSPHASE

4.1. Zielplattform

Die Applikation wird, wie bereits zuvor erwähnt, modular in die bestehende Intranetseite eingebunden.

Der Chat soll eine für sich lauffähige Anwendung sein, somit wird die Unabhängigkeit bei der Wartung gewährleistet.

Die Applikation kann außerdem unabhängig von der gesamten Intranetseite veröffentlicht(deployed) werden.

4.2. Qualitätssicherung

- Authentizität:
 - Der Benutzer wird durch ein Active Directory authentifiziert und besitzt eine zugewiesene Mitarbeiter ID
- Vertraulichkeit:
 - Die Daten kommen aus vertrauter Quelle, da es sich um eine Interne Software handelt und alle Daten im Intranet übertragen und empfangen werden
- Integrität
 - Die Nachrichten werden codiert, bevor diese an den Server gesendet werden. Somit ist gewährleistet das keine schädlichen Daten übertragen werden.

4.3. Schnittstellen

Für das „Live Chatting“ wird die Bibliothek SignalR verwendet.

Mit SignalR ist es möglich in Echtzeit Anfragen zwischen Server und Client mit wenig Netzauslast zu realisieren.

Für die Datenbank wird das Entity Framework genutzt. Es wird eine Datenbank im Code First Prinzip realisiert um somit Unabhängigkeit und schnelle Konfigurationsmöglichkeiten zu gewährleisten.

Mehr zu dem Thema befindet sich im Anhang.



4.4. Programmkonzept

Allgemein

Der Client des Benutzers kommuniziert über JavaScript Funktionen mit dem Server Hub, welcher die Verbindungen bzw. Connections behandelt und verarbeitet. Der Server Hub ist so gesehen der Controller bei einem MVC-Projekt. Der Server Hub ist in C# geschrieben und kommuniziert mit dem Backend bzw. der Daten- und Benutzerverwaltung. Die Echtzeit Kommunikation zwischen Client und Server ohne viel Netzauslastung ermöglicht die sogenannte Library SignalR.

Wenn der Benutzer die Intranetseite öffnet wird im Hintergrund eine Methode mit der Mitarbeiter ID des eingeloggten Benutzers als Parameter aufgerufen. Die Mitarbeiter ID existiert bereits in der Intranetseite und ist als globale Variable definiert.

Diese Methode gibt eine Partial View zurück und dient dazu die Chat Applikation modular in einer Fenster zu laden.

Dieses Fenster wird angezeigt, sobald der Benutzer unten rechts auf der Intranetseite auf den Button „Live Support Chat“ klickt.

Bevor die Partial View von der Methode zurückgegeben wird, werden alle relevanten Daten über den eingeloggten Mitarbeiter anhand der übergebenen Mitarbeiter ID aus mehreren Funktionen und Methoden geladen und es wird eine Partial View erzeugt.

Sobald der Benutzer den Chat geöffnet hat, eine Kategorie und einen Ansprechpartner ausgewählt hat, kann dieser anfangen Nachrichten zu senden.

Per „Enter“-Taste oder Klick auf den Senden-Button, wird die Funktion um die Nachricht in die Datenbank zu schreiben, sowie die Funktion, die die Nachricht in Echtzeit an alle verbundenen Clients schickt, aufgerufen.

Sobald der angeschriebene Benutzer anfängt zu schreiben, sieht der Benutzer dass dieser schreibt ([Benutzer] schreibt...“).

Zudem wird in der Benutzer-Info angezeigt, ob der Chatpartner online oder offline ist. Dieser Status ändert sich ebenfalls in Echtzeit (Seite geschlossen=User ist offline, Seite geöffnet=User ist online). So kann der Benutzer sehen ob, sein Chatpartner die Nachricht lesen kann oder nicht verfügbar ist.

Außerdem bekommt der Chatpartner sogenannte Notifikationen, sobald man ihn angeschrieben hat. Auch dies geschieht in Echtzeit.

So kann der Angeschriebene einkommende Nachrichten auch sehen, wenn er z.B. gerade den Chat nicht offen hat.

Ebenso bekommt der Benutzer auf der Startseite die Nachrichten als Notifikationen angezeigt, die an ihn gesendet wurden, während er offline war.

Um es zu ermöglichen in der Nachricht auch Sonderzeichen und bestimmte HTML tags



(z.B. `</br>` für einen Absatz) zu versenden, wird die Nachricht vor dem Senden an den Server codiert.

Sicherheit:

Zum Abwehren von „Cross-Site-Scripting“, d.h. damit keine missbräuchlichen Skripte an den Server geschickt werden können, wird die Nachricht codiert an den Server geschickt.

Eine Nachricht mit dem HTML Tag „`<script>`“ wird nicht an den Server gesendet und es gibt eine Fehlermeldung beim Client.

Zudem ist die maximale Länge der Nachricht auf 300 Zeichen beschränkt, damit keine endlosen Nachrichten verschickt werden können.

Design:

Die Chat Applikation ist im „Responsive“-Design gestaltet. Das heißt die Elemente der Oberfläche passen sich dynamisch der Größe des Fensters an.

4.5. Pflichtenheft

Am Ende der Entwurfsphase wurde mithilfe des Lastenhefts und der Vorarbeit aus der Entwurfsphase das Pflichtenheft verfasst.

Dieses dient als Leitfaden für die Implementierung des Projektes.

Das komplette Pflichtenheft befindet sich im Anhang „Pflichtenheft“.

5. REALISIERUNGSPHASE

Zur Realisierung habe ich folgende Technologien verwendet:

- C#
- ASP.Net MVC
- SignalR
- Entity Framework
- HTML, JavaScript (jQuery)

5.1. Datenbank

Die Datenbank inklusive der Struktur wird im Code selbst generiert. Es wird eine LocalDB als mdf Datei in die Projektmappe gespeichert.

Die Datenbank habe ich im Code First Prinzip erstellt.

Mit dem Code First Prinzip des Entity Frameworks ist es möglich schnell eine einfach zu wartende Datenbank inkl. Schema zu erstellen.

Der Code, der die Datenbank und die Struktur erstellt befindet sich in der Projektmappe. Somit ist der Entwickler völlig unabhängig von externen Datenbanken und kann die Entwicklung ohne Fremdeinwirkungen fortführen.

Der Code zur Generierung der Datenbank liegt im Anhang „[Das Code First Prinzip](#)“.



Die Datenbank wurde mit Beispieldaten befüllt.

In der finalen Version wird die User Tabelle nicht mehr existieren, da die Daten aus der internen Mitarbeiter-Tabelle geholt werden.

Die Zuordnung eines Users zu einer bestimmten Kategorie als Support wird allerdings weiterhin in der Datenbank gepflegt, damit keine Anpassung der internen Datenbank notwendig ist.

Benutzerinformationen wie Nachname und Vorname werden dann über diese ID aus der internen Datenbank geholt.

Die Datenbank wurde in der 3. Normalform entworfen.

Das Datenbankschema beinhaltet folgende Tabellen:

- User
 - Diese Tabelle beinhaltet die IDs sowie weitere für den Chat notwendige Informationen der Nutzer
 - Im späteren Verlauf der Implementierung wird diese Tabelle nicht mehr existieren, da die Daten bereits in der Internen Datenbank bestehen.
- SupportGroup
 - Diese Tabelle beinhaltet die verschiedenen Kategorien, die auf der Seite existieren
- User2SupportGroup
 - Diese Tabelle dient als Zwischentabelle. Hier sind die User als Support einer bestimmten Kategorie zugewiesen
- Chatlog
 - Diese Tabelle beinhaltet den Chatverlauf zwischen zwei Usern

Noch Detaillierter kann man dieses Schema in dem ERM-Diagramm betrachten. Dies befindet sich im Anhang „[ERM-Diagramm](#)“.

5.2. Back-End

Für den Back-End Bereich wurde die Serversprache C# verwendet.

Im Back-End Bereich habe ich Methoden geschrieben, die Daten aus der Datenbank mithilfe des Entity Frameworks lesen, ändern und löschen können.

Außerdem gibt es die Methode, die den fertig gerenderten Chat als Partial View zurückgibt. Diese Methode wird von der Intranetseite gerufen, sobald auf den Button „Live Support Chat“ geklickt wurde. Mit dem übergebenen Parameter, welcher der Mitarbeiter ID des Net-Users entspricht, wird der aktuelle Nutzer des Chats identifiziert.

Außerdem befinden sich hier die sogenannten „Hubs“ des SignalR Frameworks. Diese „Hubs“ sind sozusagen die Schnittstelle zwischen Client und Server, welche die Echtzeit-Features ermöglichen.

Code-Ausschnitte des Back-End Bereiches sind im Anhang „Back-End“ zu finden.



5.3. Front-End

Im Front-End Bereich habe ich den Chat mit HTML als View entworfen und mit CSS designend.

Sobald der Chat von einem anderen Projekt (hier die Intranetseite) mit der Mitarbeiter ID als Parameter aufgerufen wird, wird diese ID an das JavaScript weitergegeben.

Im JavaScript wird diese ID entgegengenommen und es werden unter anderem diese Funktionen aufgerufen:

- getInfoByCurrentUser(mitarbeiterId)
 - holt die User Infos anhand der ID aus der Datenbank, lädt Messages die gesendet wurden als der User offline war und füllt damit die View
- signalRToClient()
 - beinhaltet signalR-Events, die getriggert werden sobald etwas an den Hub gesendet wurde. Diese Events zeigen die Daten dann in Echtzeit auf der Oberfläche an
- setUserOnline(mitarbeiterId)
 - Sobald der User die Seite aufruft, bekommt der User den Status „Online“. Dies wird per Ajax-Call in die Datenbank geschrieben.
 - Außerdem wird ein SignalR-Event getriggert, somit erkennt jeder verbundene User in Echtzeit das dieser User in den Status „Online“ gewechselt hat
- setUserOffline(mitarbeiterId)
 - Sobald der User die Seite schließt, bekommt der User den Status „Offline“. Dies wird per Ajax Call in die Datenbank geschrieben.
 - Außerdem wird ein SignalR Event getriggert, somit erkennt jeder verbundene User in Echtzeit das dieser User in den Status „Offline“ gewechselt hat

Weitere Code-Ausschnitte sind im Anhang zu finden [„Front-End“](#)

5.4. Sicherheit

Abfangen von Gefahren durch „Cross Site Scripting“:

- Durch das Codieren der Nachrichten kann kein schädlicher Code auf dem Server ausgeführt werden, wie z.B. durch SQL-Injection.



6. TESTPHASE

Getestet wurde die Anwendung einerseits von dem Entwickler selbst. Hierbei wurden die Testverfahren Funktionalitäts- und Performance-Tests, welche nach jeder Änderung erfolgten, angewendet. Zusätzlich gab es mehrere Test-Benutzer die während des Entwicklungsprozesses ebenfalls die Anwendung im Frontend auf Funktionalität und Sicherheit überprüften. Durch diese Verfahren konnte die Qualität des Produktes stetig verbessert werden.

Zu testende Funktionalitäten:

- Wird der Chat Button korrekt angezeigt?
- Wird das Fenster korrekt geladen?
- Ist das Fenster Responsive?
- Wird die Oberfläche korrekt angezeigt?
- Kann man einen Ansprechpartner wählen?
- Werden ältere Nachrichten korrekt geladen?
- Können Nachrichten gesendet werden?
- Können Nachrichten empfangen werden?
- Funktioniert das „Live Chatting“?
- Werden die Nachrichten korrekt angezeigt?
- Wird angezeigt, ob der Chatpartner gerade schreibt?
- Ist die Online/Offline Anzeige korrekt?
- Funktionieren die Notifikationen, wenn der angeschriebene offline ist
- Funktionieren die Notifikationen, wenn der angeschriebene online ist (Anzeige nur wenn der Chat geschlossen ist)?

Nach Abschluss aller Programmierarbeiten wurden alle Frontend-Funktionalitäten von Test-Benutzern unterschiedlicher Berufspositionen und Funktionen getestet und alle Back-End-Funktionalitäten von dem Entwickler getestet. Zu den Testpersonen zählten Auszubildende, Front-End- und Back-End Entwickler.

Die Tests wurden während und nach der Entwicklung durchgeführt, um stetig alle Funktionalitäten auf Korrektheit zu prüfen.

Die Testprotokolle befinden sich im Anhang [„Testprotokolle“](#).



7. DOKUMENTATION

Die Dokumentation besteht aus drei Teilen: Projektdokumentation, Benutzerdokumentation und Entwicklerdokumentation.

7.1. Projektdokumentation

In der Projektdokumentation sind die einzelnen Phasen der Umsetzung des Projektes ausführlich beschrieben.

7.2. Entwicklerdokumentation

Die Entwicklerdokumentation wurde mithilfe des Tools GhostDocs aus den Kommentaren im Code generiert und liegt im „Compiled HTML Help File“ vor.

Ein Auszug der Entwicklerdokumentation befindet sich im Anhang [„Entwicklerdokumentation“](#).

7.3. Kundendokumentation

Die Kundendokumentation richtet sich an die Benutzer der Applikation, in diesem Fall die Mitarbeiter der Firma Berenberg. Sie gibt einen Überblick über alle in der Applikation verfügbaren Funktionen. Ebenfalls vorhanden sind Erläuterungen zu bestimmten Funktionsweisen.

Die vollständige Kundendokumentation befindet sich im Anhang [„Kundendokumentation“](#).



8. QUELENNACHWEISE

Entity Framework im Code First Prinzip:

- [https://msdn.microsoft.com/en-us/library/jj193542\(v=vs.113\).aspx](https://msdn.microsoft.com/en-us/library/jj193542(v=vs.113).aspx)

SignalR Grundlagen

- <https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr>

SignalR Notifications

- <http://www.dotnetawesome.com/2016/05/push-notification-system-with-signalr.html>
- <http://www.codemag.com/Article/1210071>

Sicherheit

- <https://docs.microsoft.com/en-us/aspnet/signalr/overview/security/introduction-to-security#authentication>

Hilfe bei Problemen

- <https://www.codeproject.com/>
- <http://www.stackoverflow.com/>
- <http://www.google.de/>

9. GLOSSAR

Begriff	Bedeutung
SignalR	SignalR ist eine Bibliothek für ASP.NET Entwickler, die es ermöglicht "Real-Time" Daten zwischen Client und Server zu senden.
Hubs	Dies ist eine API von SignalR und repräsentiert eine Verbindung zwischen Server und Clients. Sie ermöglicht Methoden zu entwickeln, die bestimmte Aktionen in Echtzeit an den aktiven Clients durchführen können
C#, Javascript, HTML, CSS	Verwendete Programmiersprachen
Cross-Site-Scripting	Der Client versucht schädlich Code auszuführen
MVC	Ist ein Muster zur Trennung von Software in drei Komponenten (Model, View, Controller)
Entity Framework	Ist ein Framework für objektrelationale Abbildungen. Hier wird es genutzt um die Datenbank zu verwalten.
Code First Prinzip	Die Datenbank wird durch C# generiert und ist somit unabhängig von externen Datenbanken



ERM	Entity-Relationship-Model. Eine Modellierungsform der Datenbank.
Front-End	Oberfläche (Clientseite) der Applikation
Back-End	Hintergrund (Serverseite) der Applikation
Rendern	Dynamisches Erstellen einer Oberfläche
Ajax Call	Eine Funktion aus der JQuery Bibliothek. Kann Serverseitige Methoden aufrufen.
SQL-Injection	Der Client versucht schädlichen SQL Code an die Datenbank zu schicken
LSC	Name des Projekts (Live Support Chat)



10. ANHANG

10.1. Verwendete Ressourcen

Hardware

- Büroarbeitsplatz
 - Thin Client
 - Tastatur
 - Maus
 - 2 Monitore

Software

- Windows 2012 Server R2 Standard
- Visual Studio 2015 Professional
- Microsoft Word 2010
- Microsoft Excel 2010
- Microsoft Visio 2016
- FastStone Capture (Screenshots)
- GhostDoc (Entwicklerdokumentation)

Personal

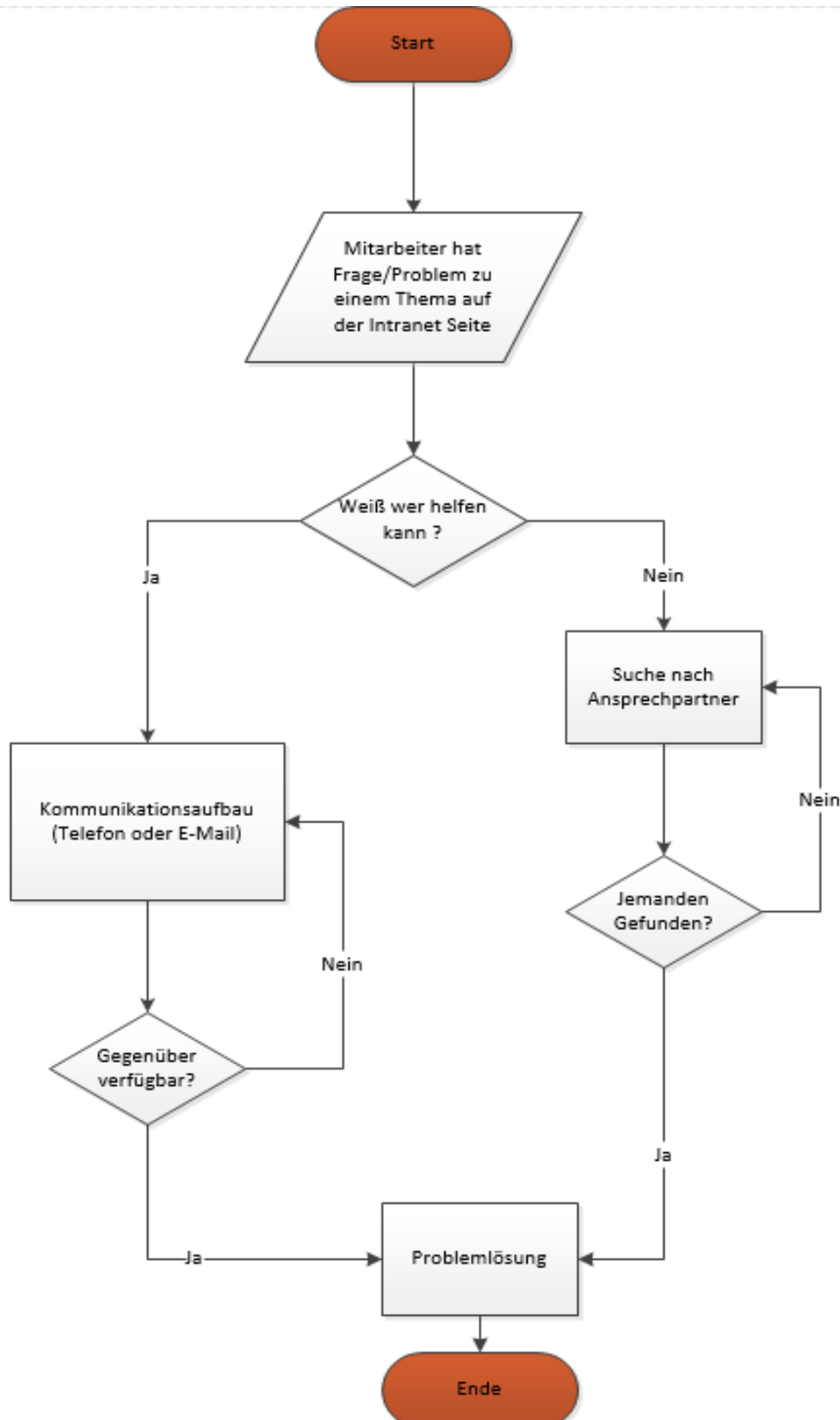
- Auftraggeber (Festlegung der Anforderungen, Abnahme Projekt)
- Auszubildender (Umsetzung des Projektes)
- 2x Auszubildende/r (manuelles Testen)
- 2x Entwickler (manuelles Testen)

Alle oben aufgelisteten Ressourcen sind entweder bereits vorhanden oder unter einer freien Lizenz verfügbar, so dass nichts extra angeschafft werden muss.



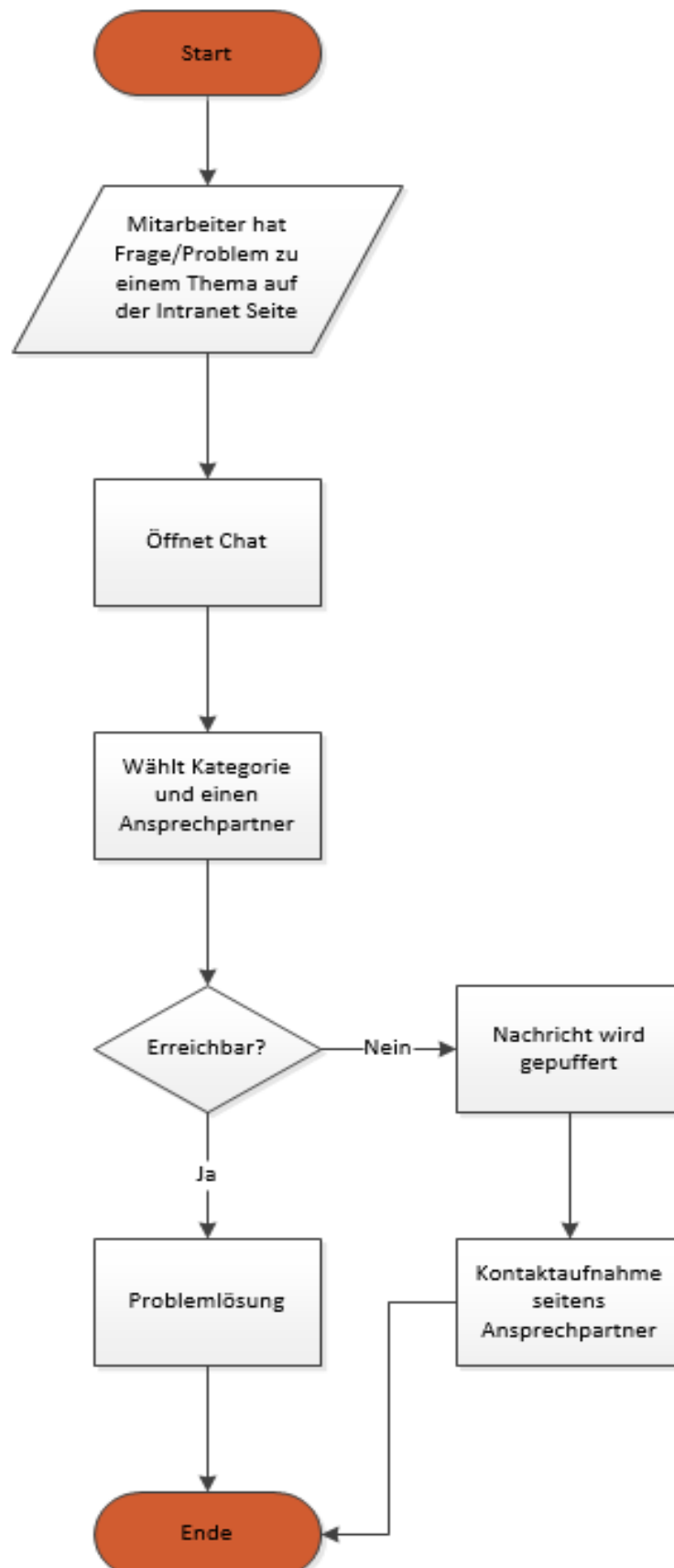
10.2. Diagramme

10.2.1. IST-Zustand



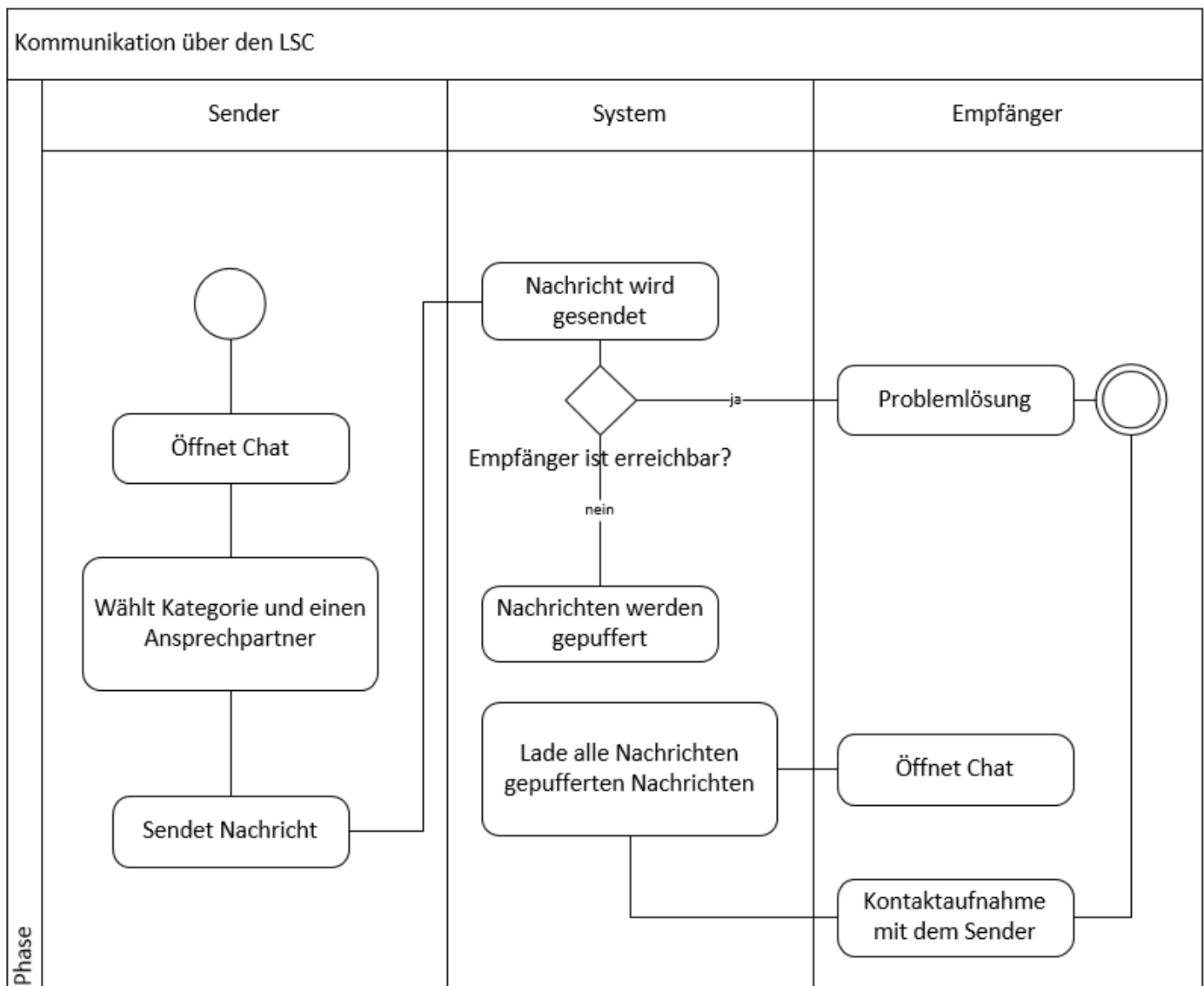


10.2.2. SOLL-Zustand

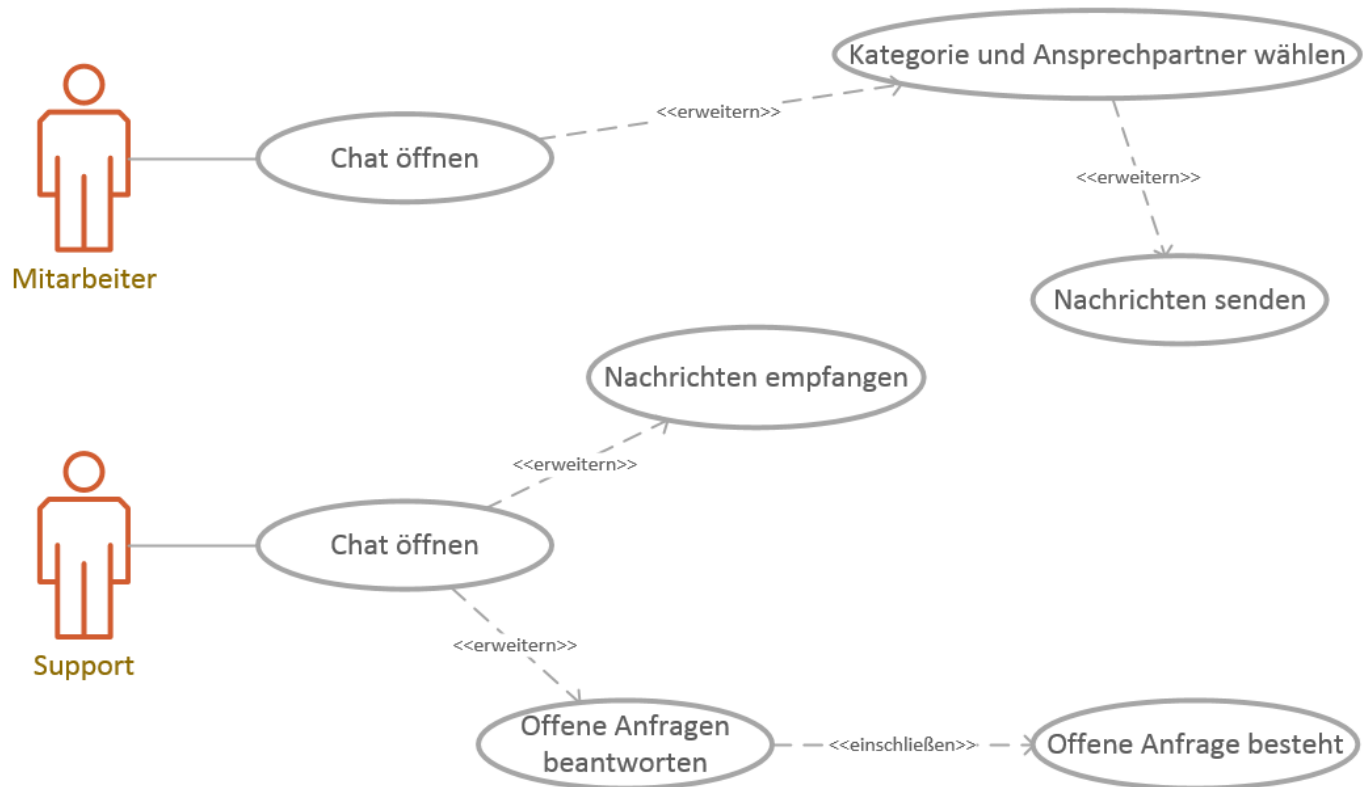




10.2.3. Aktivitätsdiagramm

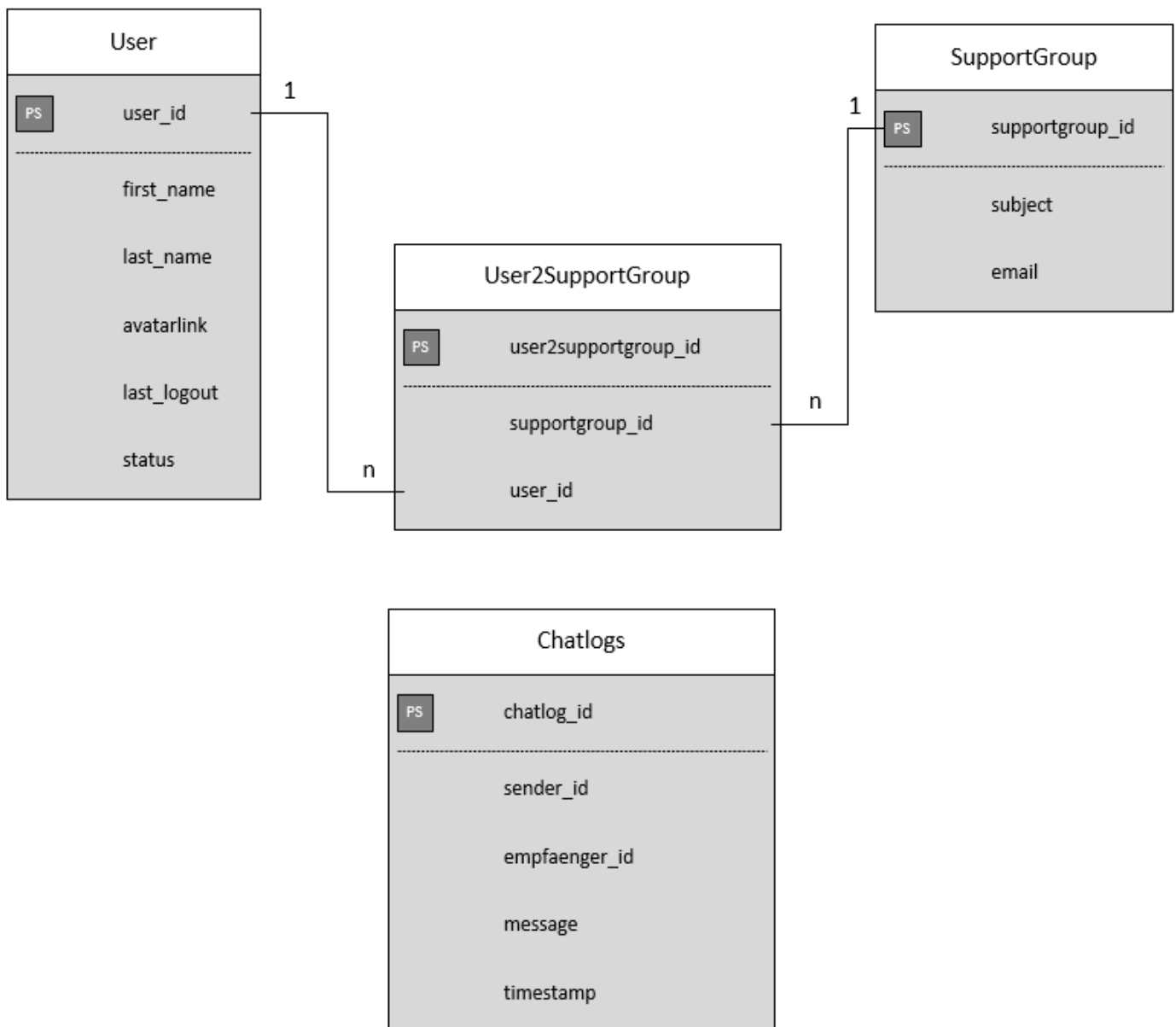


10.2.4. Anwendungsfalldiagramm



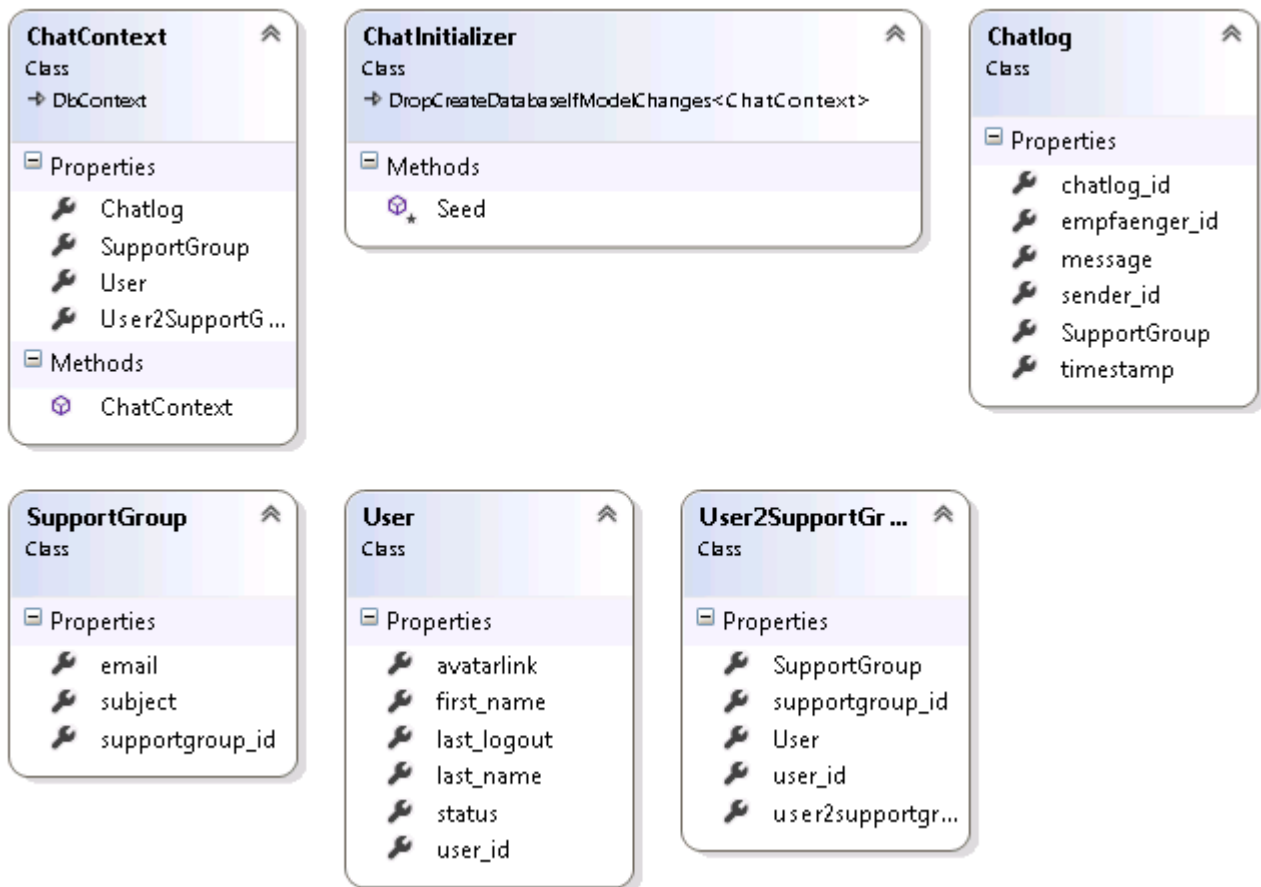


10.2.5. ERM



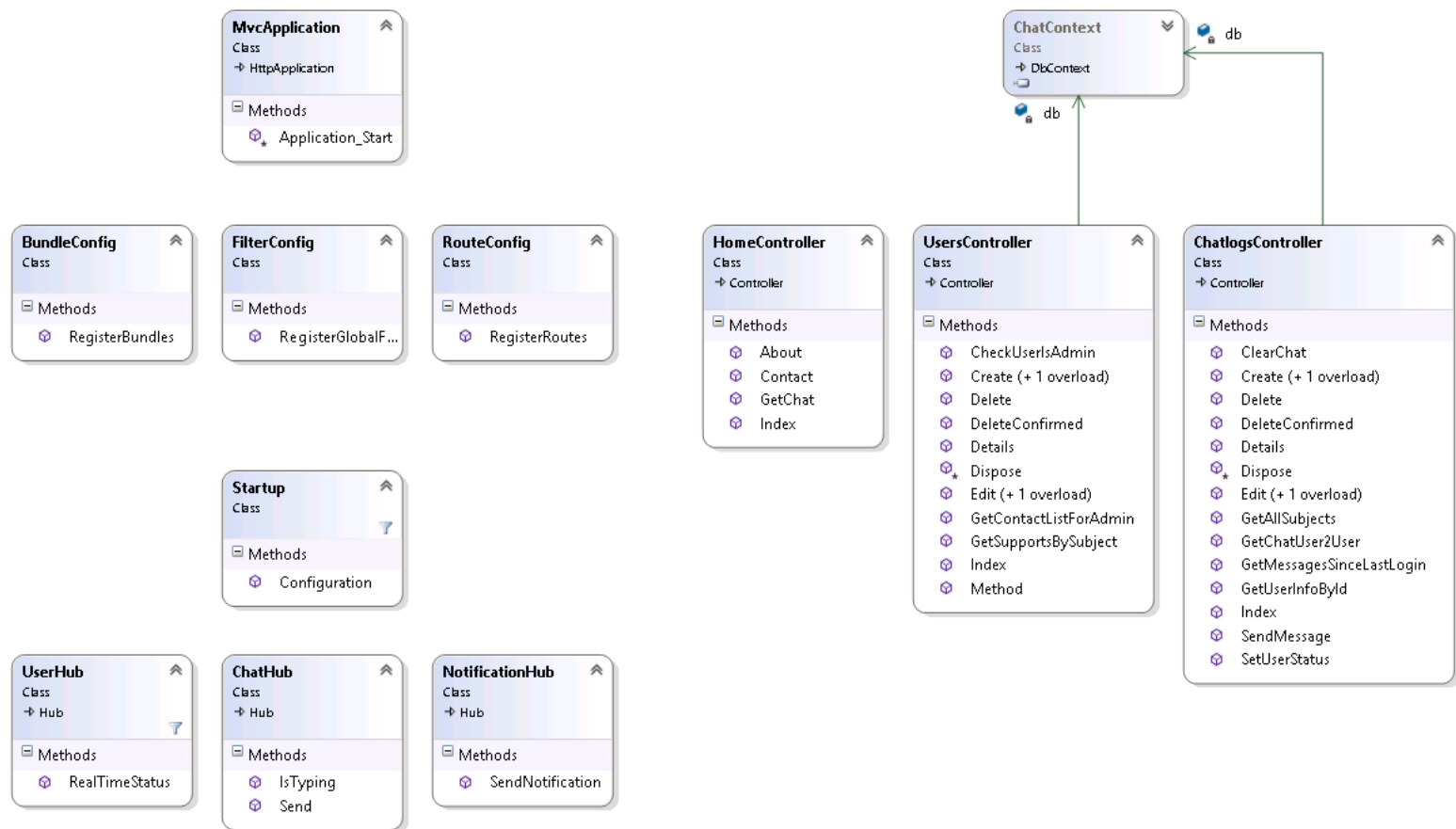


10.2.6. Klassendiagramm – Database Code First





10.2.7. Klassendiagramm – GUI





10.3. Entwicklerdokumentation (Auszug)

ChatlogsController.SendMessage Method

Sends the message.

Namespace: [ChatModule.Controllers](#)
Assembly: ChatModule (in ChatModule.dll)

Syntax

C# **VB** **C++**

```
[ValidateInputAttribute]
public string SendMessage(
    int sender_id,
    int empfaenger_id,
    string message
)
```

Parameters

sender_id
Type: [Int32](#)
The sender identifier.

empfaenger_id
Type: [Int32](#)
The empfaenger identifier.

message
Type: [String](#)
The message.

Return Value

String...The Message parameter

See Also

[ChatlogsController Class](#)
[ChatModule.Controllers Namespace](#)

Help File generated with GhostDoc

10.4. Kundendokumentation

10.4.1. Einleitung

In diesem Benutzer-Handbuch werden ausschließlich die Oberflächen und deren Funktionalitäten beschrieben. Antworten zu technische Fragen entnehmen Sie bitte aus der Dokumentation.

10.4.2. Masken und Funktionalitäten

Der erste Schritt ist das Öffnen der Intranet Webseite.

Nachdem Sie die Webseite geöffnet haben, sehen Sie folgende grafische Oberfläche:

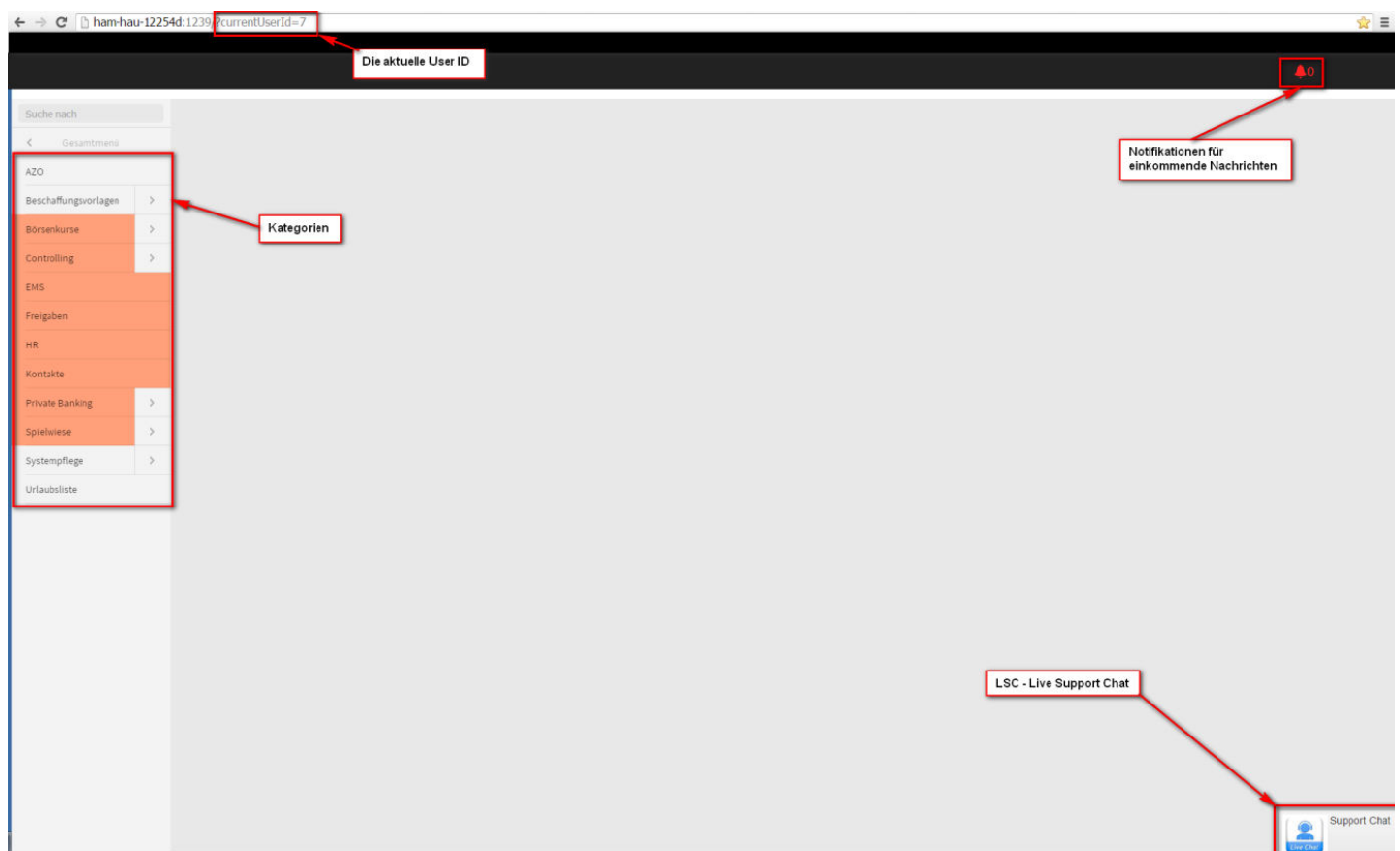
Im folgenden Bild sehen Sie schon im Bild einige Hinweise zu den wichtigsten Themen, die mit dem Chat zusammenhängen.

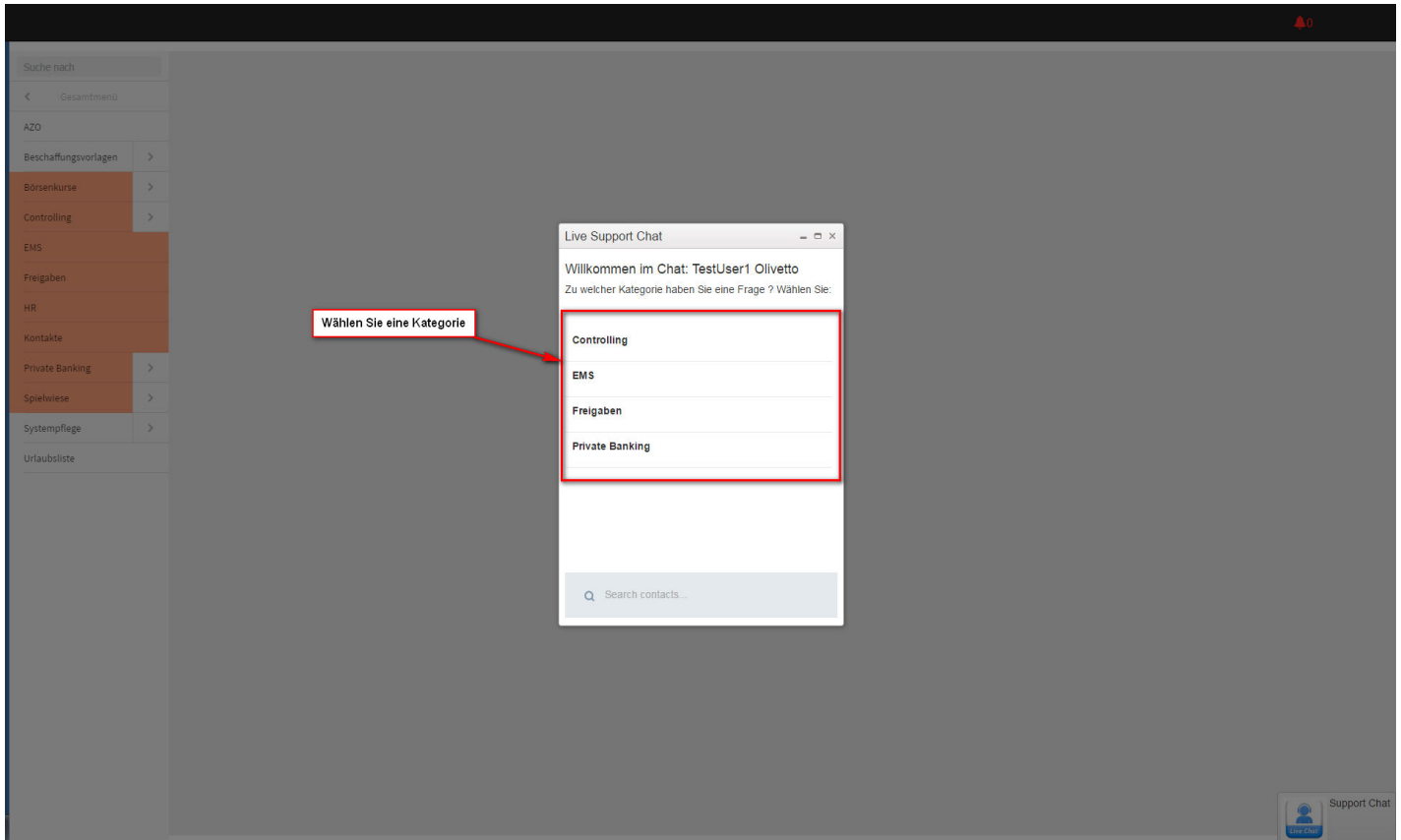
Oben rechts: Hier sehen Sie die Notifikationen über einkommende Nachrichten.

Oben: Die aktuelle User ID ist ihre in der Datenbank zugewiesene ID. Dies dient der Identifikation.

Links: Hier befindet sich das Menü, welches in mehrere Kategorien eingeteilt ist.

Unten rechts: Hier sehen Sie den Button „Support Chat“, wenn Sie den Button mit der linken Maustaste klicken, öffnet sich der Chat in einem kleinen Fenster, mittig des Bildschirmes. Siehe nächstes Bild.





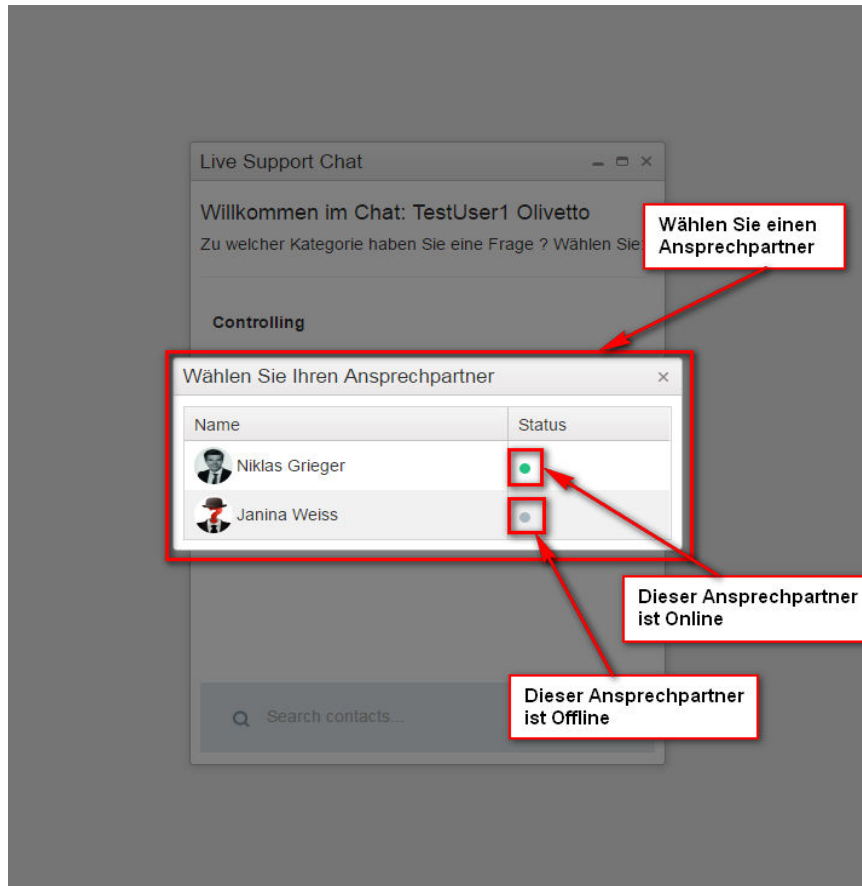
Hier können Sie eine Kategorie, zu dem Sie eine Frage haben, auswählen.

Als nächstes wählen Sie einen zuständigen Support. Vorzugsweise jemanden der online ist.

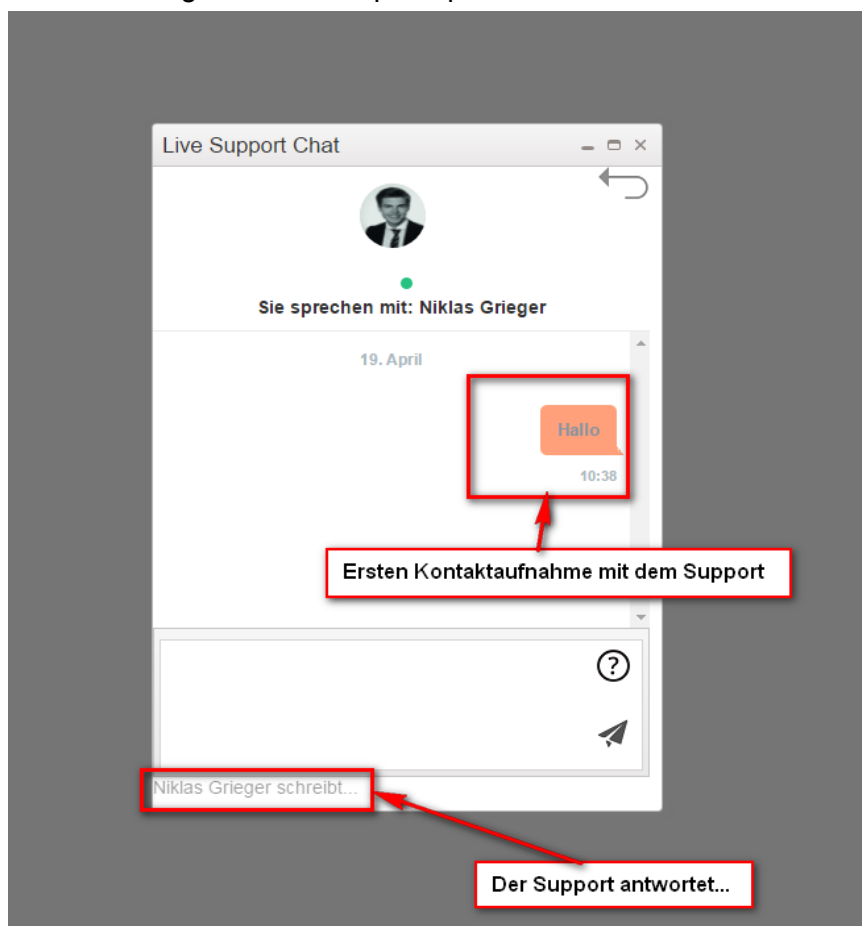
Ihnen ist aber auch die Möglichkeit gegeben jemanden anzuschreiben, der offline ist. Jedoch kann dieser die Nachricht erst lesen, wenn dieser erneut online ist.

In diesem Schritt wird beispielhaft die Kategorie „Controlling“ ausgewählt.

In den folgenden Bildern werden Beispiele zu einer Konversation aufgeführt.

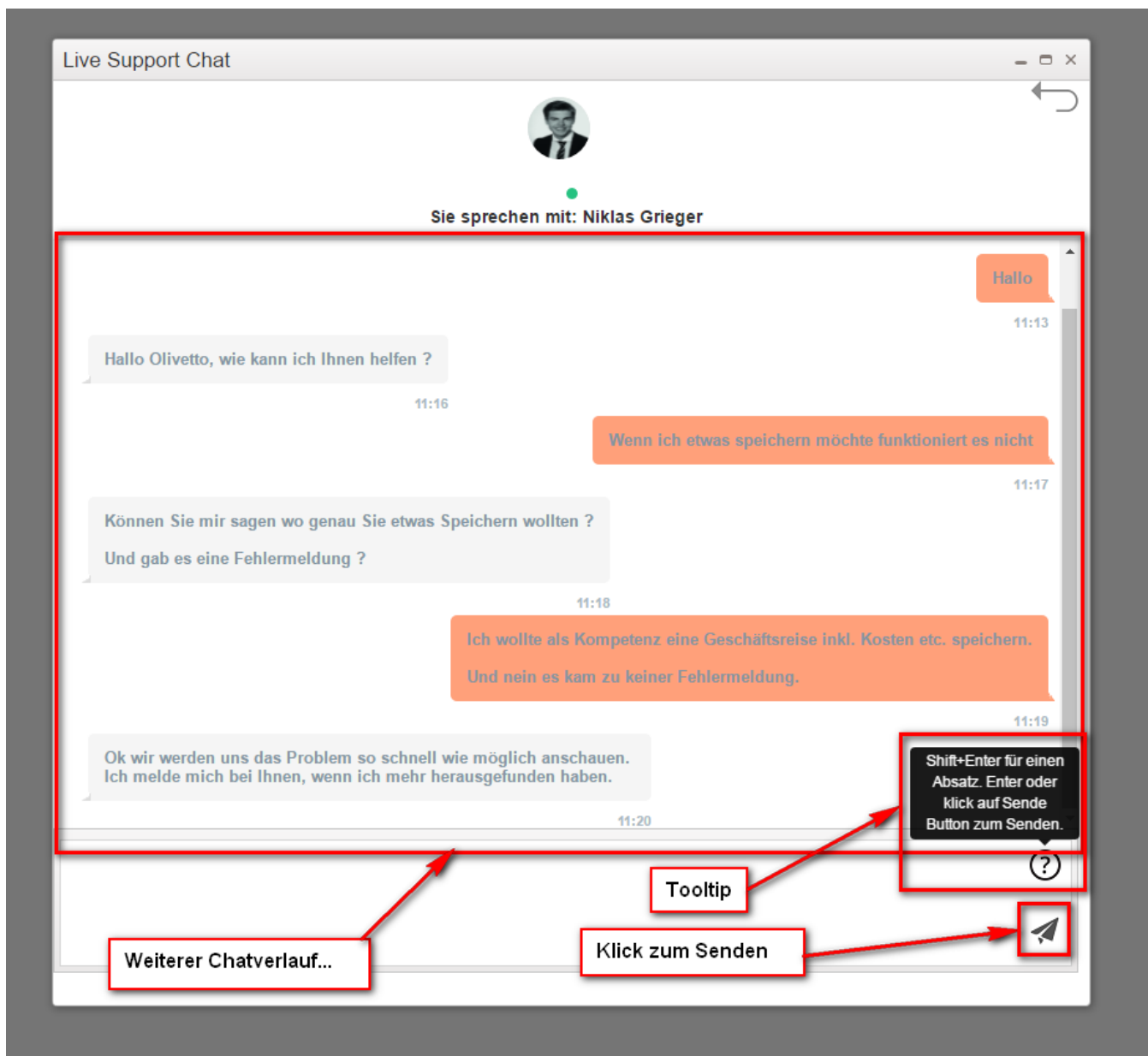


Nachdem Linksklick auf einen der Auswahlmöglichkeiten ändert sich die Ansicht und Sie werden mit dem zuvor ausgewählten Ansprechpartner verbunden.

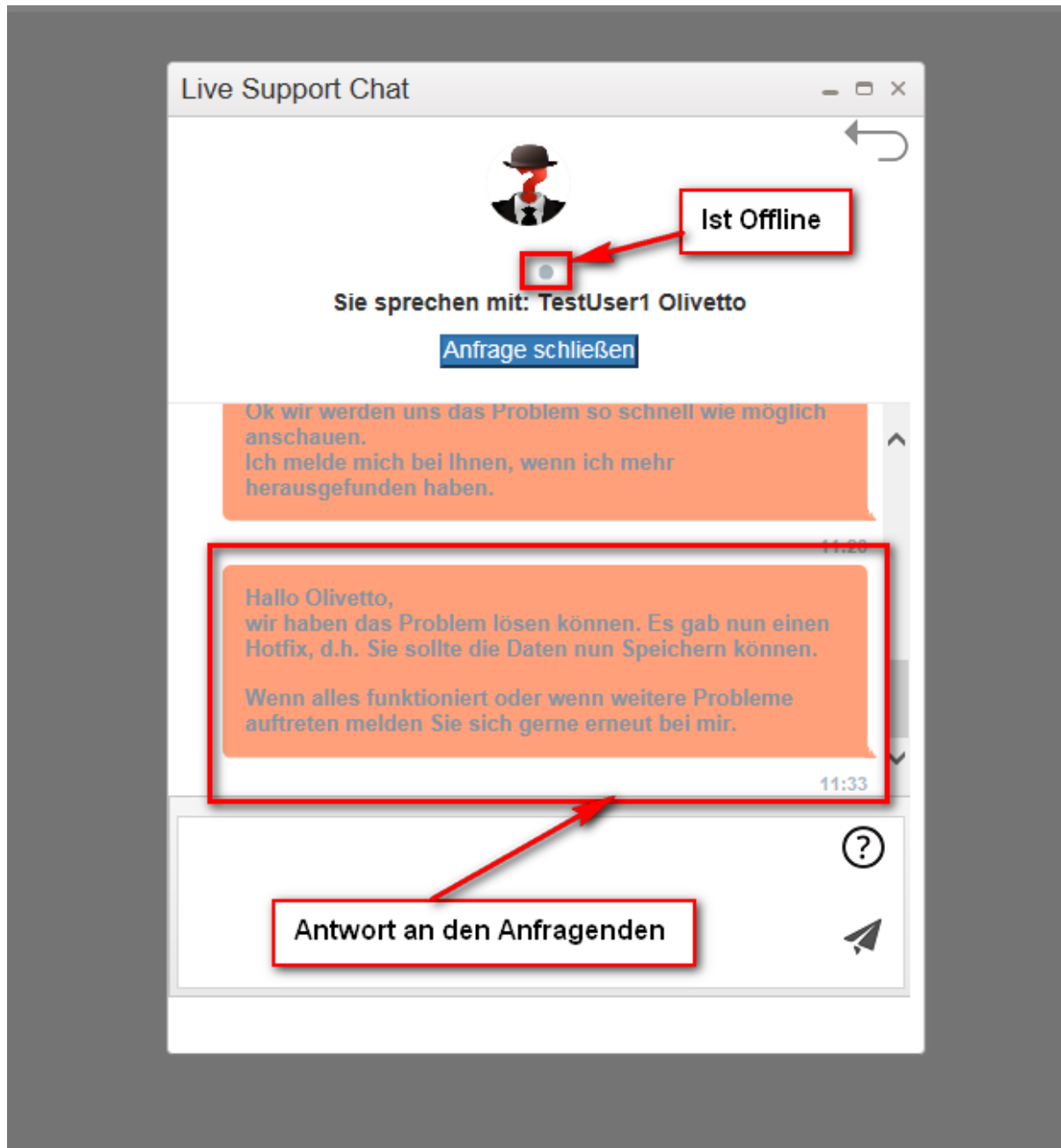


Sie können die Nachricht durch „Enter“ oder durch Klick auf den Senden-Button abschicken. Sie können einen Absatz mit der Tastatur Kombination „Shift+Enter“ einfügen. Dies finden Sie auch als Hinweis/Tooltip, wenn Sie den Mauszeiger über das Fragezeichen im Chat halten.

Hier ein Beispiel zum weiteren Chatverlauf.



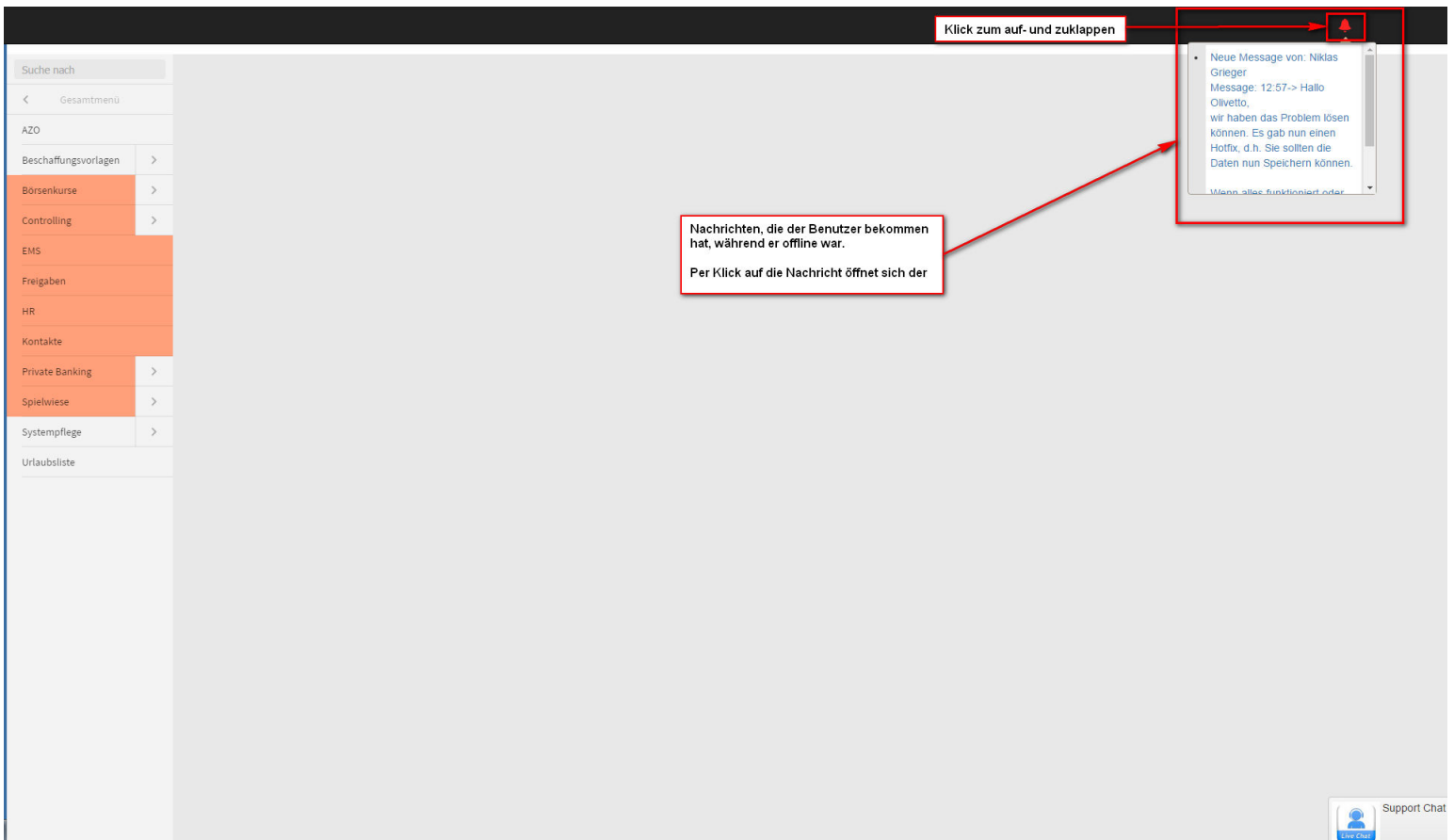
Nun ist der Anfragende offline gegangen und bekommt Antwort vom Support.



Wenn Sie einmal offline sind und Ihre Anfrage vom Support beantwortet wird, werden die Nachrichten gepuffert.

Sobald Sie die Seite Intranet erneut aufrufen bzw. in den Status „Online“ gehen bekommen Sie die Nachrichten seit ihrem letzten Besuch übersichtlich dargestellt und Sie können sich per Klick auf diese Notifikation direkt mit dem Sender verbinden.

Der Anfragende betritt die Seite und hat den Status „Online“.



The screenshot shows the LSC interface with a sidebar menu on the left and a main content area. A notification bubble is visible in the top right corner, indicating a new message. A red box highlights the notification bubble, and a red arrow points to it from a text box. Another red box highlights the notification bubble's content, and a red arrow points to it from a text box.

Klick zum auf- und zuklappen

Suche nach

< Gesamtmenü

AZO

Beschaffungsvorlagen >

Börsenkurse >

Controlling >

EMS

Freigaben

HR

Kontakte

Private Banking >

Spielwiese >

Systempflege >

Urlaubsliste

Nachrichten, die der Benutzer bekommen hat, während er offline war.

Per Klick auf die Nachricht öffnet sich der

Neue Message von: Niklas Grieger

Message: 12:57-> Hallo Olivetto, wir haben das Problem lösen können. Es gab nun einen Hotfix, d.h. Sie sollten die Daten nun Speichern können.

Wenn alles funktioniert oder

Support Chat

Zu guter Letzt geben Sie bitte Rückmeldung an den Support, ob ihr Problem korrekt behoben wurde, damit die Anfrage als erledigt gekennzeichnet und geschlossen werden kann.

In diesem Beispiel wurde das Problem gelöst und die Anfrage wird geschlossen.





10.5. Testprotokolle

Zu testende Funktionalitäten:

- Wird der Chat Button korrekt angezeigt?
- Wird das Fenster korrekt geladen?
- Ist das Fenster Responsive?
- Wird die Oberfläche korrekt angezeigt?
- Kann man einen Ansprechpartner wählen?
- Werden ältere Nachrichten korrekt geladen?
- Können Nachrichten gesendet werden?
- Können Nachrichten empfangen werden?
- Funktioniert das „Live Chatting“?
- Werden die Nachrichten korrekt angezeigt?
- Wird angezeigt, ob der Chatpartner gerade schreibt?
- Ist die Online/Offline Anzeige korrekt?
- Funktionieren die Notifikationen, wenn der angeschriebene offline ist
- Funktionieren die Notifikationen, wenn der angeschriebene online ist (Anzeige nur wenn der Chat geschlossen ist)?

Die folgenden Testprotokolle wurden von dem Entwickler geschrieben und mit einem oder mehreren Test-Benutzern erstellt.

Entwicklungs-Test 1 (White-Box Test): 15.05.2017

Getestete Funktionalität	Testergebnis	Fazit
Öffnen der Webanwendung	Keine Vorfälle	OK
Ist das Fenster Responsive	Nicht Responsive	Nicht OK
Laden älterer Nachrichten	Keine Vorfälle	OK
Nachrichten können gesendet werden über die „Enter“ Taste	Keine Vorfälle	OK
Nachrichten können gesendet werden über den Senden-Button	Keine Vorfälle	OK
Sonderzeichen sind in der Nachricht möglich	Keine Vorfälle	OK
Absätze sind in der Nachricht möglich	Keine Vorfälle	OK
Nachrichten werden „RealTime“ übertragen	Keine Vorfälle	OK
Anzeige der Nachrichten	Keine Vorfälle	OK



Anzeige, das der Chat-Partner gerade schreibt	Funktioniert nur teils, Anzeige im IE ist fehlerhaft	Anpassungsfähig
Online/Offline-Anzeige	Keine Vorfälle	OK
Notifikationen, wenn der Angeschriebene offline ist	Keine Vorfälle	OK
Notifikationen, wenn der Angeschriebene online ist (Anzeige nur wenn der Chat geschlossen ist)	Funktioniert, aber Anzeige ist unschön	Anpassungsfähig
Cross-Site-Scripting ist nicht möglich	Keine Vorfälle	OK
Daten werden korrekt aus der Datenbank gelesen	Keine Vorfälle	OK
Daten werden korrekt in die Datenbank geschrieben	Keine Vorfälle	OK
Daten können in der Datenbank geändert werden	Keine Vorfälle	OK
Performance ist optimal	Ist annehmbar	Anpassungsfähig

Entwicklungs-Test 2 (White-Box Test): 16.05.2017

Getestete Funktionalität	Testergebnis	Fazit
Öffnen der Webanwendung	Keine Vorfälle	OK
Ist das Fenster Responsive	Keine Vorfälle	OK
Laden älterer Nachrichten	Keine Vorfälle	OK
Nachrichten können gesendet werden über die „Enter“ Taste	Keine Vorfälle	OK
Nachrichten können gesendet werden über den Senden-Button	Keine Vorfälle	OK
Sonderzeichen sind in der Nachricht möglich	Keine Vorfälle	OK
Absätze sind in der Nachricht möglich	Keine Vorfälle	OK
Nachrichten werden „RealTime“ übertragen	Keine Vorfälle	OK



Anzeige der Nachrichten	Keine Vorfälle	OK
Anzeige, das der gegenüber gerade schreibt	Keine Vorfälle	OK
Online/Offline Anzeige	Keine Vorfälle	OK
Notifikationen, wenn der Angeschriebene offline ist	Keine Vorfälle	OK
Notifikationen, wenn der Angeschriebene online ist(Anzeigen nur wenn der Chat geschlossen ist)	Keine Vorfälle	OK
Cross-Site-Scripting ist nicht möglich	Keine Vorfälle	OK
Daten werden korrekt aus der Datenbank gelesen	Keine Vorfälle	OK
Daten werden korrekt in die Datenbank geschrieben	Keine Vorfälle	OK
Daten können in der Datenbank geändert werden	Keine Vorfälle	OK
Performance ist optimal	Ist annehmbar	Anpassungsfähig

Frontend-Test 2 (Black-Box Test): 18.05.2015

Getestete Funktionalität	Testergebnis	Fazit
Öffnen der Webanwendung	Keine Vorfälle	OK
Ist das Fenster Responsive	Keine Vorfälle	OK
Laden älterer Nachrichten	Keine Vorfälle	OK
Nachrichten können gesendet werden über die „Enter“ Taste	Keine Vorfälle	OK
Nachrichten können gesendet werden über den Senden Button	Keine Vorfälle	OK
Sonderzeichen sind in der Nachricht möglich	Keine Vorfälle	OK
Absätze sind in der Nachricht möglich	Keine Vorfälle	OK



Nachrichten werden „Real-Time“ übertragen	Keine Vorfälle	OK
Anzeige der Nachrichten	Keine Vorfälle	OK
Anzeige, das der gegenüber gerade schreibt	Keine Vorfälle	OK
Online/Offline-Anzeige	Keine Vorfälle	OK
Notifikationen, wenn der Angeschriebene offline ist	Keine Vorfälle	OK
Notifikationen, wenn der Angeschriebene online ist (Anzeige nur wenn der Chat geschlossen ist)	Keine Vorfälle	OK
Cross-Site-Scripting ist nicht möglich	Keine Vorfälle	OK

Frontend-Test 1 (Black-Box Test): 17.05.2015

Getestete Funktionalität	Testergebnis	Fazit
Öffnen der Webanwendung	Keine Vorfälle	OK
Ist das Fenster Responsive	Nicht Responsive	OK
Laden älterer Nachrichten	Keine Vorfälle	OK
Nachrichten können gesendet werden über die „Enter“-Taste	Keine Vorfälle	OK
Nachrichten können gesendet werden über den Senden-Button	Keine Vorfälle	OK
Sonderzeichen sind in der Nachricht möglich	Keine Vorfälle	OK
Absätze sind in der Nachricht möglich	Keine Vorfälle	OK
Nachrichten werden „Real-Time“ übertragen	Keine Vorfälle	OK
Anzeige der Nachrichten	Keine Vorfälle	OK
Anzeige, das der gegenüber gerade schreibt	Keine Vorfälle	OK
Online/Offline Anzeige	Keine Vorfälle	OK



Notifikationen, wenn der Angeschriebene offline ist	Keine Vorfälle	OK
Notifikationen, wenn der Angeschriebene online ist(Anzeige nur wenn der Chat geschlossen ist)	Keine Vorfälle	OK
Cross-Site-Scripting ist nicht möglich	Keine Vorfälle	OK



10.6. Code-Ausschnitte

10.6.1. Datenbank - Das Code First Prinzip

- **Context**

Repräsentiert eine Session mit der Datenbank, über die Daten gelesen und gespeichert werden können.

Die einzelnen Attribute vom Typ DbSet<> repräsentieren die Tabellen

```
// represents a session with the database, allowing us to query and save data. We define a context that derives from System.Data.Entity.DbContext and exposes a typed DbSet<TEntity> for each class in our model.
public class ChatContext : DbContext
{
    public ChatContext() : base("ChatContext")
    {
    }

    public DbSet<User> User { get; set; }
    public DbSet<Chatlog> Chatlog { get; set; }
    public DbSet<SupportGroup> SupportGroup { get; set; }
    public DbSet<User2SupportGroup> User2SupportGroup { get; set; }
}
```

- **Model (Struktur einer Tabelle)**

```
//Struktur einer Tabelle
public class User
{
    [Key]
    public int user_id { get; set; }

    [Required]
    public string first_name { get; set; }

    [Required]
    public string last_name { get; set; }

    public string avatarlink { get; set; }

    public DateTime? last_logout { get; set; }

    [DefaultValue("offline")]
    public string status { get; set; }
}
```



- **Connection String in der webconfig**

Im Connection String steht der Pfad der Datenbank Datei

```
<connectionStrings>
  <add name="ChatContext" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;
    AttachDBFilename=|DataDirectory|\ContosoUniversity1.mdf;
    Connection Timeout=10"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

- **DB_INITIALIZER**

Der Initializer wird genutzt um die Datenbank mit festgelegten Daten zu bestücken, falls die Datenbank neu erstellt wird.

Durch den Befehl DropCreateDatabaseIfModelChanges<> wird die Datenbank gelöscht und neu erstellt, sobald wenn sich ein Model bzw. die Datenbankstruktur ändert.

```
class ChatInitializer : System.Data.Entity.DropCreateDatabaseIfModelChanges<ChatContext>
{
    protected override void Seed(ChatContext context)
    {
        var user = new List<User>
        {
            new User{first_name="Niklas",last_name="Grieger",
                avatarlink="Chat/img/grieger.bmp", status="inactive",
                last_logout=DateTime.Now},
            new User{first_name="Janina",last_name="Weiss",
                avatarlink="Chat/img/unknown.jpg",
                status="inactive",
                last_logout=DateTime.Now},
            new User{first_name="Martina",last_name="Eiffel",
                avatarlink="Chat/img/unknown.jpg", status="inactive",
                last_logout=DateTime.Now},
            new User{first_name="David",last_name="Osterhagen",
                avatarlink="Chat/img/unknown.jpg", status="inactive",
                last_logout=DateTime.Now},
            new User{first_name="Peter",last_name="Helfer",
                avatarlink="Chat/img/unknown.jpg",
                status="inactive", last_logout=DateTime.Now},
            new User{first_name="Max",last_name="Mustermann",
                avatarlink="Chat/img/unknown.jpg", status="inactive",
                last_logout=DateTime.Now},
            new User{first_name="TestUser1",last_name="Olivetto",
                avatarlink="Chat/img/unknown.jpg", status="inactive",
                last_logout=DateTime.Now},
            new User{first_name="TestUser2",last_name="Olivetto",
                avatarlink="Chat/img/unknown.jpg", status="inactive",
                last_logout=DateTime.Now},
            new User{first_name="TestUser3",last_name="Olivetto",
                avatarlink="Chat/img/unknown.jpg", status="inactive",
                last_logout=DateTime.Now}
        };
    }
}
```




```
user.ForEach(s => context.User.Add(s));
context.SaveChanges();
```

. . . Dasselbe passiert für die anderen Tabellen

10.6.1. Backend

HomeController.cs

```
// *****
// Assembly      : ChatModule
// Author        : grieger
// Created       : 04-20-2017
//
// Last Modified By : grieger
// Last Modified On : 05-10-2017
// *****
// <copyright file="HomeController.cs" company="Berenberg">
//     Copyright © Berenberg 2016
// </copyright>
// <summary></summary>
// *****
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace ChatModule.Controllers
{
    /// <summary>
    /// Class HomeController.
    /// </summary>
    /// <seealso cref="System.Web.Mvc.Controller" />
    public class HomeController : Controller
    {
        /// <summary>
        /// Indexes this instance.
        /// </summary>
        /// <returns>ActionResult.</returns>
        public ActionResult Index()
        {
            return View();
        }
        /// <summary>
        /// Gets the chat as a View.
        /// </summary>
        /// <param name="partial">if set to <c>true</c> [partial].</param>
        /// <param name="currentUserId">The current user identifier.</param>
        /// <returns>ActionResult...if Partial param is true: an full loaded View
        include the Layout, if Partial param is false: an Partial View without the Layout <
        /returns>
    }
}
```



```

public ActionResult GetChat(bool partial, int currentUserId)
{
    var request = Request;
    ViewBag.Partial = partial;
    ViewBag.CurrentUserId = currentUserId;
    if (partial == true)
    {
        return PartialView("~/Views/Chatlogs/Index.cshtml");
    }
    else
    {
        return View("~/Views/Chatlogs/Index.cshtml");
    }
}
}

```

UsersController.cs (Ausschnitt)

```

...
/// <summary>
/// Gets the supports by subject.
/// </summary>
/// <param name="subject">The subject.</param>
/// <returns>JsonResult of the Supports assigned to this subject</returns>
public JsonResult GetSupportsBySubject(string subject)
{
    return Json((from a in db.SupportGroup
        join b in db.User2SupportGroup on a.supportgroup_id
        equals b.supportgroup_id
        join c in db.User on b.user_id equals c.user_id
        where a.subject == subject
        select new
        {
            supporter_id = b.user_id,
            user_name = c.first_name + " " + c.last_name,
            status = c.status,
            avatarlink = c.avatarlink
        }), JsonRequestBehavior.AllowGet);
}
/// <summary>
/// Gets the contact list for admin.
/// </summary>
/// <param name="currentUserId">The Current User</param>
/// <returns>JsonResult with the Contact list for this User</returns>

```



```

public JsonResult GetContactListForAdmin(int currentUserId)
{
    var result = db.Database.SqlQuery<User>("SELECT * FROM Users u
        LEFT JOIN User2SupportGroup u2u ON u2u.user_id = u.user_id " +
        "WHERE u2u.user_id IS NULL " +
        "AND u.user_id in
        (Select c.sender_id from Chatlogs c where c.empfaenger_id="
        + currentUserId + ")").ToList();
    return Json(result, JsonRequestBehavior.AllowGet);
}

/// <summary>
/// Checks the user is admin.
/// </summary>
/// <param name="user_id">The user identifier.</param>
/// <returns><c>true</c> if user is assigned to an subject, <c>false</c> otherwise
/// </returns>
public bool CheckUserIsAdmin(int user_id)
{
    var result = db.Database.SqlQuery<int>("Select user_id from User2SupportGroup
where user_id=" + user_id).Count();
    ...

    if (result > 0)
    {
        return true;
    }

    else if (result == 0)
    {
        return false;
    }
    return false;
}

```

ChatlogsController.cs (Ausschnitt)

```

...
/// <summary>
/// Sends the message.
/// </summary>
/// <param name="sender_id">The sender identifier.</param>
/// <param name="empfaenger_id">The empfaenger identifier.</param>
/// <param name="message">The message.</param>
/// <returns>String...The Message parameter</returns>
public String SendMessage(int sender_id, int empfaenger_id, string message)
{
    var decoded = HttpUtility.HtmlDecode(message);
    var chatlog = new Chatlog
    {
        sender_id = sender_id,
        empfaenger_id = empfaenger_id,
        message = decoded,
        timestamp = DateTime.Now
    };
}

```



```

try
{
    db.Chatlog.Add(chatlog);
    db.SaveChanges();
    return "Message send success!";
}
catch (DbEntityValidationException e)
{
    foreach (var eve in e.EntityValidationErrors)
    {
        Console.WriteLine("Entity of type \"{0}\" in state \"{1}\" has the following validation errors:",
            eve.Entry.Entity.GetType().Name, eve.Entry.State);
        foreach (var ve in eve.ValidationErrors)
        {
            Console.WriteLine("- Property: \"{0}\", Error: \"{1}\"",
                ve.PropertyName, ve.ErrorMessage);
        }
    }
    return "Message send failed!";
    //throw;
}
}

//Gibt die Chat Messages in der Datenbank als JsonObject zurück
/// <summary>
/// Gets the chat user2 user.
/// </summary>
/// <param name="sender_id">The sender identifier.</param>
/// <param name="empfaenger_id">The empfaenger identifier.</param>
/// <returns>JsonResult...The Chat history of the 2 Users</returns>
public JsonResult GetChatUser2User(int sender_id, int empfaenger_id)
{
    return Json((from a in db.Chatlog
        where (a.sender_id == sender_id
            && a.empfaenger_id == empfaenger_id)
            || (a.sender_id == empfaenger_id
            && a.empfaenger_id == sender_id)

        select new
        {
            sender_id = a.sender_id,
            empfaenger_id = a.empfaenger_id,
            message = a.message,
            timestamp = a.timestamp
        }), JsonRequestBehavior.AllowGet);
}

/// <summary>
/// Gets all subjects.
/// </summary>
/// <returns>JsonResult...Get All Subjects in Database</returns>
public JsonResult GetAllSubjects()
{
    return Json((from a in db.SupportGroup

```



```

        select new
        {
            supportgroup_id = a.supportgroup_id,
            subject = a.subject
        }), JsonRequestBehavior.AllowGet
    );
}
/// <summary>
/// Sets the user status.
/// </summary>
/// <param name="status">The status.</param>
/// <param name="currentUserId">The current user identifier.</param>
public void SetUserStatus(string status, int currentUserId)
{
    using (var db = new ChatContext())
    {
        var result = db.User.SingleOrDefault(a => a.user_id == currentUserId);
        if (result != null)
        {
            if (status == "inactive")
            {
                result.last_logout = DateTime.Now;
            }
            result.status = status;
            db.SaveChanges();
        }
    }
}
/// <summary>
/// Gets the messages since last login.
/// </summary>
/// <param name="currentUserId">The current user identifier.</param>
/// <returns>JsonResult.</returns>
public JsonResult GetMessagesSinceLastLogin(int currentUserId)
{
    var result = from a in db.Chatlog
        join b in db.User on a.empfaenger_id equals b.user_id
        where a.empfaenger_id == currentUserId
        where a.timestamp > b.last_logout
        select new
        {
            sender_name = (db.User.Where(u => u.user_id == a.sender_id)
                .Select(u => u.first_name + " " + u.last_name)),
            sender_id = a.sender_id,
            supportgroup_id = (db.User2SupportGroup
                .Where(u => u.user_id == a.sender_id)
                .Select(u => u.supportgroup_id)),
            empfaenger_id = a.empfaenger_id,
            message = a.message,
            timestamp = a.timestamp,
            last_logout_empfaenger = b.last_logout
        };
}

```



```

        return Json(result, JsonRequestBehavior.AllowGet);
    }
    /// <summary>
    /// Clears the chat.
    /// </summary>
    /// <param name="sender_id">The sender identifier.</param>
    /// <param name="empfaenger_id">The empfaenger identifier.</param>
    public void ClearChat(int sender_id, int empfaenger_id)
    {
        var rows = from o in db.Chatlog
                    where (o.sender_id == sender_id &&
                           o.empfaenger_id == empfaenger_id) || (o.sender_id == empfaenger_id
                                                                    && o.empfaenger_id == sender_id)
                    select o;
        foreach (var row in rows)
        {
            db.Chatlog.Remove(row);
        }
        db.SaveChanges();
    }
    /// <summary>
    /// Gets the user information by identifier.
    /// </summary>
    /// <param name="currentUserId">The current user identifier.</param>
    /// <returns>JsonResult.</returns>
    public JsonResult GetUserInfoById(int currentUserId)
    {
        //var result = db.User.Where(a => a.user_id == currentUserId).FirstOrDefault();
        ;
        var result = from a in db.User
                      where a.user_id == currentUserId
                      select new
                      {
                          user_id = a.user_id,
                          full_name = a.first_name + " " + a.last_name,
                          avatarlink = a.avatarlink,
                          last_logout = a.last_logout
                      };

        return Json(result.FirstOrDefault(), JsonRequestBehavior.AllowGet);
    }
    ...

```

**Hubs:****UserHub.cs**

```
// *****
// Assembly      : ChatModule
// Author        : grieger
// Created       : 04-22-2017
//
// Last Modified By : grieger
// Last Modified On : 06-10-2017
// *****
// <copyright file="UserHub.cs" company="Berenberg">
//     Copyright © Berenberg 2016
// </copyright>
// <summary></summary>
// *****
using Microsoft.AspNet.SignalR;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace ChatModule.Hubs
{
    /// <summary>
    /// Class UserHub.
    /// </summary>
    /// <seealso cref="Microsoft.AspNet.SignalR.Hub" />
    public class UserHub : Hub
    {
        /// Enthält den Status der Users
        /// <summary>
        /// Reals the time status.
        /// </summary>
        /// <param name="userId">The user identifier.</param>
        /// <param name="status">The status.</param>
        public void RealTimeStatus(int userId, string status)
        {
            /// Sendet den Status des User an alle Clients
            Clients.All.getRealTimeStatus(userId, status);
        }

        public void Method()
        {
            throw new NotImplementedException();
        }
    }
}
```



ChatHub.cs

```
// *****
// Assembly      : ChatModule
// Author        : grieger
// Created       : 04-22-2017
//
// Last Modified By : grieger
// Last Modified On : 06-10-2017
// *****
// <copyright file="ChatHub.cs" company="Berenberg">
//     Copyright © Berenberg 2016
// </copyright>
// <summary></summary>
// *****
using System;
using System.Web;
using Microsoft.AspNet.SignalR;

namespace SignalRChat
{
    /// <summary>
    /// Class ChatHub.
    /// </summary>
    /// <seealso cref="Microsoft.AspNet.SignalR.Hub" />
    public class ChatHub : Hub
    {
        //Enthält Daten für eine Chat message
        /// <summary>
        /// Sends the specified sender name.
        /// </summary>
        /// <param name="sender_name">Name of the sender.</param>
        /// <param name="empfaenger_id">The empfaenger identifier.</param>
        /// <param name="message">The message.</param>
        /// <param name="timestamp">The timestamp.</param>

        Public void Send(string sender_name, int empfaenger_id,
            string message, string timestamp)
        {
            //Sendet die Message an die Oberfläche aller verbundenen Clients
            Clients.All.addNewMessageToPage(sender_name, empfaenger_id
                , message, timestamp);
        }
        //Enthält Daten wer gerade schreibt
        /// <summary>
        /// Determines whether the specified name is typing.
        /// </summary>
        /// <param name="name">The name.</param>
        /// <param name="sender_id">The sender identifier.</param>
        /// <param name="empfaenger_id">The empfaenger identifier.</param>
        /// <param name="currentUserId">The current user identifier.</param>
    }
}
```




```

        public void IsTyping(string name, int sender_id,
                               int empfaenger_id, int currentUserId)
        {
            //Sendet die Info, wer gerade schreibt, an alle verbundenen Clients
            Clients.All.sayWhoIsTyping(name, sender_id,
                                       empfaenger_id, currentUserId);
        }
    }
}

```

NotificationHub.cs

```

// *****
// Assembly      : ChatModule
// Author        : grieger
// Created       : 04-22-2017
//
// Last Modified By : grieger
// Last Modified On : 06-10-2017
// *****
// <copyright file="NotificationHub.cs" company="Berenberg">
//     Copyright © Berenberg 2016
// </copyright>
// <summary></summary>
// *****
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Microsoft.AspNet.SignalR;

namespace ChatModule.Hubs
{
    //Now, we will create a SignalR Hub class, which makes possible to invoke the cli-
    //ent side JavaScript method from the server side. Here in this application, we will
    //use this for showing notification.
    //SignalR uses 'Hub' objects to communicate between the client and the server.
    /// <summary>
    /// Class NotificationHub.
    /// </summary>
    /// <seealso cref="Microsoft.AspNet.SignalR.Hub" />
    public class NotificationHub : Hub
    {
        /// <summary>
        /// Sends the notification.
        /// </summary>
        /// <param name="notification">The notification.</param>
        /// <param name="sender_id">The sender identifier.</param>
        /// <param name="empfaenger_id">The empfaenger identifier.</param>
    }
}

```



```

    public void SendNotification(string notification,
                                int sender_id, int empfaenger_id)
    {
        GlobalHost.ConnectionManager.GetHubContext<NotificationHub>()
            .Clients.All.getNotification(notification, sender_id, empfaenger_id);
    }
}

```

Startup.cs

```

// *****
// Assembly      : ChatModule
// Author        : grieger
// Created       : 04-22-2017
//
// Last Modified By : grieger
// Last Modified On : 06-10-2017
// *****
// <copyright file="Startup.cs" company="Berenberg">
//     Copyright © Berenberg 2016
// </copyright>
// <summary></summary>
// *****
using Owin;
using Microsoft.Owin;
[assembly: OwinStartup(typeof(SignalRChat.Startup))]
namespace SignalRChat
{
    /// <summary>
    /// Class Startup.
    /// </summary>
    public class Startup
    {
        /// <summary>
        /// Configurations the specified application.
        /// Initilize all the SignalR Hubs
        /// </summary>
        /// <param name="app">The application.</param>
        public void Configuration(IAppBuilder app)
        {
            // Any connection or hub wire up and configuration should go here
            app.MapSignalR();
        }

        public void Method()
        {
            throw new System.NotImplementedException();
        }
    }
}

```



10.6.2. Front End

Index.cshtml (Ausschnitt):

```
<div id="chatContent">
  <div id="window_chooseSupport">
    <div id="grid_chooseSupport" style="display:none;" ></div>
  </div>
  <div id="friendslist">
    <header id="top">
      <h4></h4>
      <h5></h5>
    </header>
    <main id="friends">

      </main>
    <footer id='search'>
      <input type='text' id='searchfield'
        placeholder='Search contacts...' />
    </footer>
  </div>

  <div id="chatview">
    <header id="profile">
    </header>

    <main id="chat-messages">

      </main>
    <footer class="footer">
      <form id="messageForm">
        <div id="sendMessage" style="position:relative;">
          <span id="notAllowed" style="position:absolute;
            z-index:1;top:20%;left:50%"></span>
          <a href="#" data-toggle="tooltip"
            class="tooltip-icon" style="right:15px;top:15%"
            title="Shift+Enter für einen Absatz.
            Enter oder klick auf Sende Button zum Senden."></a>
          <button id="send"></button>
          <div id="kendoValidationMsg"
            style="position:absolute;z-index:1;">
            <span data-for='message' class='k-invalid-msg'></span>
          </div>
          <textarea id="messageeditor" name="message"
            style="position:relative;"></textarea>
        </div>
      </form>
      <div id="isTyping" style="position:absolute;opacity:0.5;"></div>
    </footer>
  </div>
</div>
```

**ChatCustomStyle.css (Ausschnitt):**

```

/*START-----MAIN-----*/
#chat-messages{
    opacity:0;
    margin-top:30px;
    height: 100%;
    overflow-y:scroll;
    overflow-x:hidden;
}
#chat-messages.animate{
    opacity:1;
    margin-top:0;
}
#chat-messages label{
    color:#aab8c2;
    font-weight:600;
    font-size:12px;
    text-align:center;
    margin:15px 0;
    display:block;
}
#chat-messages div.message{
    padding:0 0 15px 15px;
    clear:both;
    margin-bottom:45px;
}
#chat-messages div.message.right{
    padding: 0 15px 15px 0;
    margin-right: -19px;
    margin-left: 19px;
}
.message .bubble{
    background:whitesmoke;
    font-size:13px;
    font-weight:600;
    padding:12px 13px;
    margin: 13px;
    border-radius:5px 5px 5px 0px;
    color:#8495a3;
    position:relative;
    float:left;
}
#chat-messages div.message.right .bubble{
    float:right;
    border-radius:5px 5px 0px 5px ;
    background: lightsalmon;
}
/*Zeichen links der message*/
.bubble .corner {
    background: url('../img/left_corner_lightgrey.png') 0 0 no-repeat;
    position: absolute;
    width: 7px;
    height: 7px;
}

```



```

    left: -5px;
    bottom: 0;
}
/*Zeichen rechts der message*/
div.message.right .corner {
    background: url('../img/right_corner_lightsalmon.png') 0 0 no-repeat;
    left: auto;
    right: -5px;
}

```

NotificationCustomStyle.css

```

/*----- Style for Notification ----- */
----- */

.noti-content {
    position: fixed;
    right: 100px;
    background: #e5e5e5;
    border-radius: 4px;
    top: 47px;
    width: 250px;
    display: none;
    border: 1px solid #9E988B;
}

ul#notiContent {
    max-height: 200px;
    overflow: auto;
    padding: 0px;
    margin: 0px;
    padding-left: 20px;
}

ul#notiContent li {
    margin: 3px;
    padding: 6px;
    background: #fff;
}

.noti-top-arrow {
    border-color: transparent;
    border-bottom-color: #F5DEB3;
    border-style: dashed dashed solid;
    border-width: 0 8.5px 8.5px;
    position: absolute;
    right: 32px;
    top: -8px;
}

span.noti {
    color: #FF2323;
    margin: 15px;
    position: fixed;
    right: 100px;
}

```



```

        font-size: 18px;
        cursor: pointer;
    }

    span.count {
        position: relative;
        top: -3px;
    }

    /*-----
    Style for Notification -----
    -----*/

```

NotificationCustomScript.js(Ausschnitt)

```

/**
 * Created by Niklas Grieger on 02.12.2016.
 * js for the Chat Widget
 */
var Notification = Notification || (function () {
    .
    .
    .

    function init() {
        // Click on notification icon for show notification
        $('span.noti').click(function (e) {
            e.stopPropagation();
            $('.noti-content').show();
            var count = 0;
            count = parseInt($('span.count').html()) || 0;
            //only load notification if not already loaded
            if (count > 0) {
                updateNotification();
            }
            $('span.count', this).html('&nbsp;');
        })
        // hide notifications
        $('html').click(function () {
            $('.noti-content').hide();
        })
        // signalr js code for start hub and send receive notification
        var notificationHub = $.connection.notificationHub;
        $.connection.hub.start().done(function () {
            console.log('Notification hub started');
        });
        //signalr method for push server message to client
        notificationHub.client.notify = function (message) {
            if (message && message.toLowerCase() == "added") {
                updateNotificationCount();
            }
        }
    }
    return {
        Init: init
    }
}

```



```
}
})();
```

ChatCustomScript.js (Ausschnitt):

```
/**
 * Created by Niklas Grieger on 02.12.2016.
 * js for the Chat Widget
 */
/// <var>The chat javascript as a global variabel</var>
var Chat = Chat || (function () {
.
.
.

//Init
function init(currentUserIdParam, Partial) {
    var defaultTools = kendo.ui.Editor.defaultTools;

    //Zum setzen eines Absatzes in der Message = shift+Enter
    defaultTools["insertLineBreak"].options.shift = true;
    //Deaktiviert die funktion zum einfügen eines Paragraphen
    delete defaultTools["insertParagraph"].options;

    $("#messageeditor").kendoEditor({
        tools: [
            // "bold", "italic", "underline"
        ],
        //Wird getriggert, wenn ein Item in der Toolbar geklickt wird
        execute: function(e) {
            console.log("executing command", e.name, e.command);
        },
        keydown: function(e) {
            $.connection.hub.start().done(function () {
                var encodedName = currentUserDataSource.full_name;
                //Sendet das Signal, das jemand Schreibt an den Hub (Server)

                console.log("SignalR an Server: isTyping getriggert");
                chat.server.isTyping(encodedName,
                    currentUserDataSource.user_id,
                    empfaenger_id, currentUserDataSource.user_id);

            }).fail(function (reason) {
                console.log("SignalR connection in UserisTyping failed: "
                    + reason);
            });
            if (e.keyCode === 13 && !e.shiftKey) {
                e.preventDefault();
                console.log("Enter im Editor gedrückt");
                sendMessage();
            }
        }
    });
});
```



```

//Laden des Exception Handlers
kendoValidator();
//Zum einstellen der Höhe des Kendo Editors
setTimeout(function () {
    $(".k-editor iframe.k-content").css("height", "100px");
    $(".table.k-editor").css("height", "100px");
},100);
$('#send').on('click', function (e) {
    e.preventDefault();
    sendMessage();
});
partial = Partial;
if(currentUserIdParam != null){
    //Liest Daten aus der Datenbank zu dem aktuellen User
    //Beim erfolg, werden die Messages seit dem letzten Logout abgefragt
    //Beim erfolg, wird geprüft, ob der User ein Admin ist
    //Beim erfolg, wird die Oberfläche entsprechend des Usertyps angezeigt
    getInfoByCurrentUser(currentUserIdParam);

    //Enthält die SignalR Methoden zum anzeigen am Client
    signalRToClient();

    //Setzt den CurrentUser auf den Status available/online
    setUserOnline(currentUserIdParam);

    //Setzt den CurrentUser auf den Status inactiv/offline
    setUserOffline(currentUserIdParam);

}
else{

}

}
//Return functions
return {
    Init: init,
    //SendMessage: sendMessage,
    OpenChatUser2User: openChatUser2User,
    CloseRequest:closeRequest
}
})();

```




_Layout.cshtml

Die Demo Page, die den Chat modular lädt

Später wird dies in der Intranetseite passieren

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>LSC – Live Support Chat </title>

  <link rel="stylesheet" href="~/Scripts/KendoUI/styles/kendo.common.min.css" />
  <link rel="stylesheet" href="~/Scripts/KendoUI/styles/kendo.default.min.css" /
>
  <link rel="stylesheet" href="~/Scripts/KendoUI/styles/kendo.default.mobile.min
.css" />

  <script src="~/Scripts/KendoUI/js/jquery.min.js"></script>
  <script src="~/Scripts/KendoUI/js/kendo.all.min.js"></script>

  <link href="~/Scripts/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
  <script src="~/Scripts/bootstrap/js/bootstrap.min.js"></script>
  <link href="~/Scripts/css/NotificationCustomStyle.css" rel="stylesheet">
  <script src="~/Scripts/notify.min.js"></script>
  <style>
    #chatWindow{
      min-width: 360px;
      min-height: 500px;
      overflow: hidden;
      margin-right: .58em;
    }
  </style>
</head>
<body>
  <div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
      <div class="navbar-header">
        @*----- Notification -----
        @*
        @* added HTML code for showing notification icon top-
right corner of the page. *@
        <span class="noti glyphicon glyphicon-
bell"><span class="count">&nbsp;</span></span></span>
        <div class="noti-content">
          <div class="noti-top-arrow"></div>
          <ul id="notiContent"></ul>
        </div>
        @*----- Notification -----
        @*
        </div>
      </div>
    </div>
  </div>
```



```

<div class="container body-content">
  <div id="chatWindow" style="display:none;"></div>
  @RenderBody()
  <but-
ton id="btnChat" style="bottom: 0px; right: 0px; position:fixed;">Support Chat</bu
tton>
</div>
<script>
  $(document).ready(function () {
    // Click on notification icon for show notification
    $('span.noti').click(function (e) {
      e.stopPropagation();
      $('.noti-content').show();
      var count = 0;
      $('span.count', this).html('&nbsp;');
    })
    // hide notifications
    $('.container.body-content').click(function () {
      $('.noti-content').hide();
    })

    $("#btnChat").kendoButton({
      imageUrl: "/img/ChatIcon64.jpg"
    });

    $("#chatWindow").kendoWindow({
      minWidth: 360,
      minHeight: 500,
      content: "/Chat/Home/GetChat?partial=true&currentUserId="
        + @ViewBag.CurrentUserId,
      refresh: function (e){
        console.log(e);
        kendo.ui.progress(e.sender.element, false);
      },
      visible: false,
      actions: [
        "Minimize",
        "Maximize",
        "Close"
      ],
      close:function(e){
        $('#chatview').fadeOut();
        $('#friendslist').fadeIn();
        $("#top").show();
      },
      title: "Live Support Chat",
      modal:true
    });
    var chatWindow = $("#chatWindow").data("kendoWindow");
    kendo.ui.progress(chatWindow.element, true);
    $("#btnChat").click(function () {
      //reset forms

```



```
        $("#messageeditor").data("kendoEditor").value("");
        $("#searchfield").val("");
        chatWindow.center().open();
    });
});
</script>
</body>
</html>
```



BERENBERG

PRIVATBANKIERS SEIT 1590

LASTENHEFT

Lastenheft LSC – Live Support Chat

IT-Management
12.04.2017



Inhalt

1. Ausgangssituation	3
2. Zielsetzung	3
3. Produkteinsatz	3
4. Funktionale Anforderungen.....	3
5. Nichtfunktionale Anforderungen.....	4
6. Lieferumfang	4
7. Phasenplanung und Meilensteine des Projektes	4
8. Offene Punkte, die noch zu klären sind	5
9. Abnahmekriterien und Qualitätsanforderungen	5



1. Ausgangssituation

Es hat sich des Öfteren deutlich gezeigt, dass der Fachbereich bei Problemen nicht genau weiß, wo dieser sich zu melden hat.

Unter anderem aus diesem Grund soll es eine zusätzliche Kommunikationsmöglichkeit über die Intranet Seite geben.

2. Zielsetzung

Es soll ein Live Support Chat entwickelt werden und in die bestehende Intranet Seite implementiert werden.

Der Erfolg wird daran gemessen, ob der Fachbereich mit dem Endresultat des Live Support Chats zufrieden ist.

Es muss gewährleistet sein, dass der Benutzer die Möglichkeit hat einen themenzuständigen Ansprechpartner zu kontaktieren und in Echtzeit mit diesem zu Chatten.

3. Produkteinsatz

Das Produkt soll in der bereits bestehenden Intranet Seite eingebunden werden und dort jederzeit aufrufbar sein.

4. Funktionale Anforderungen

- Eine Datenbank mit folgenden Tabellen
 - o Benutzerinfos
 - o Die Kategorien der Intranet Seite
 - o Die Zuweisung der Kategorien zu den zuständigen Benutzer
 - o Chatlogs
- Möglichkeit zur einfachen Auswahl eines Ansprechpartners
- Eine Echtzeitkommunikation mit möglichst wenig Netzauslast
- Online/Offline Anzeige der Benutzer
- Wenn eine Konversation vorhanden ist, soll der Empfänger sehen können, wenn der Sender schreibt
- Falls der Empfänger der Nachrichten den Status offline hat, sollen die eingehenden Nachricht gepuffert werden
- Sobald der Empfänger wieder online ist, sollen ihm die gepufferten Nachrichten übersichtlich dargestellt werden
- Es soll die Möglichkeit gegeben sein, die Anfrage als erledigt zu markieren



5. Nichtfunktionale Anforderungen

Das Produkt soll so entwickelt sein, dass es auch in zukünftigen Projekten einfach eingebunden werden kann.

Das Produkt soll hauptsächlich die bereits benutzten Ressourcen der IT verwenden, um die Kosten so gering wie möglich zu halten.

Das Produkt soll die Nachrichten zuverlässig vom Sender zum Empfänger übertragen.

Das Produkt soll einfach erweiterbar sein.

6. Lieferumfang

Das Produkt soll in Form einer eigenständigen Projektmappe bereitstehen, damit eine einfache Wartung und modulare Implementierung möglich ist.

Diese Projektmappe soll später in der Intranet Seite verlinkt sein und somit modular aufgerufen werden können.

Zudem sollen eine Entwicklerdokumentation und eine Kundendokumentation/Benutzerhandbuch bereitstehen.

7. Phasenplanung und Meilensteine des Projektes

Folgende **Projektphasen** sind im Projektverlauf vorgesehen:

1. Analyse
2. Entwurf
3. Realisierung
4. Test
5. Refactoring
6. Dokumentation

Folgende **Meilensteine** sind einzuhalten:

1. Analyse und Entwurf
2. Entwickeln der Datenbank
3. Entwickeln des Front Ends
4. Test und anschließende Fehlerbereinigung
5. Dokumentation



8. Offene Punkte, die noch zu klären sind

Es konnten folgende Dinge noch nicht geklärt werden:

- Zuständigkeiten der Benutzer zu den Kategorien/Themen der Intranet Seite

Änderungen des Lastenhefts dürfen nicht durchgeführt werden.

Abweichungen zu den festgelegten Punkten aus diesem Dokument werden später in der Dokumentation festgehalten.

9. Abnahmekriterien und Qualitätsanforderungen

Das Produkt ist auf funktionale und nicht funktionale Kriterien durch den Auftragnehmer (AN) qualitätszusichern. Die Abnahme erfolgt durch Berenberg in Form eines Abnahmetests sowie der Sichtung der Testprotokolle, die im Verlauf der Entwicklung durch den AN erstellt worden sind.

Die Lieferung des Softwareprodukts muss neben den ausgewiesenen Anforderungen unter Punkt 4 und 5 vollständig und fehlerfrei sein.

Mängel und Fehler sind über ein von dem Arbeitnehmer geführtes Fehlertrackingtool zu dokumentieren. Eine Priorisierung von Mängeln und Fehlern wird anhand der Bewertungskriterien aus der Tabelle: *Priorisierung* vorgenommen.

Eine Abnahme mit noch nicht behobenen Fehlern der Priorisierung hoch und mittel ist nicht vorgesehen. Ausnahmen können im gegenseitigen Einvernehmen durch Arbeitgeber und Arbeitnehmer entschieden werden.

Priorität	Definition
1 - hoch (high)	Der Betrieb oder die beabsichtigte Nutzung der Funktion/des Geschäftsprozesses ist nicht oder nicht vollständig möglich. Der Test kann nicht durchgeführt werden. Der Fehler verhindert / verzögert weitere Tests.
2 - mittel (medium)	Der Fehler hat Auswirkungen auf die Funktion des Systems oder den Geschäftsprozess und ist nicht akzeptabel. Die weitere Testdurchführung ist mit geringfügiger Beeinträchtigung möglich. Der Fehler kann in einer folgenden Lieferung gelöst werden.
3 - niedrig (low)	Der Fehler hat niedrige oder keine Auswirkung auf die Funktion des Systems oder den Geschäftsprozess und ist nicht produktionsverhindernd. Die Lösung des Fehlers kann bis zur nächsten Softwarelieferung warten.



BERENBERG
PRIVATBANKIERS SEIT 1590

PFLICHTENHEFT

Pflichtenheft

LSC - Live Support Chat

Niklas Grieger
18.04.2017

Datum	Autor	Änderungsgrund / Bemerkungen	Version
18.04.2017	Niklas Grieger	Erstanlage des Pflichtenheftes	0



19.04.2017	Niklas Grieger	Ausarbeitung des Pflichtenheftes	0.1
20.04.2017	Niklas Grieger	Genauere Beschreibung des Pflichtenheftes	0.2
22.04.2017	Niklas Grieger	Anpassung von Angaben	0.3
22.05.2017	Niklas Grieger	Aktualisierung von Diagrammen	1.0



Inhaltsverzeichnis

1. Projektbeschreibung	1
1.1. Zweck und Ziel des Dokuments	1
1.2. Projektbezug	1
1.3. Ablage, Gültigkeit und Bezüge zu anderen Dokumenten	1
2. Beteiligte	1
3. Projektmanagement	2
4. Ist-Zustand	3
4.1 Momentaner Zustand	3
4.2 Geplante Veränderungen	3
4.3 Ablaufplan des Ist-Zustands	3
5. Soll-Zustand	4
5.1 Sollzustand-Beschreibung	4
5.2 Anforderungskatalog	4
5.3 Nutzen für den Anwender	5
5.4 Zielgruppe	5
5.5 Schnittstellen	5
5.6 Ablaufplan des Soll-Zustands	6
6. Ablauf	7
6.1 Projektplan	7
6.2 Kritische Punkte/Projektrisiken	7
6.3 Testfälle	8
7. Ressourcen	9
8. Kostenverteilung	10
9. Genehmigung	10



1. PROJEKTBE SCHREIBUNG

Beschreibung und Erklärung des Projektes

1.1. Zweck und Ziel des Dokuments

Das Pflichtenheft beschreibt die Anforderungen des Auftraggebers, an den Auftragnehmer und dieser in kurzer Form eine Lösungsrichtung.

Erst wenn der Auftraggeber das Pflichtenheft akzeptiert, sollte die eigentliche Umsetzungsarbeit beim Auftragnehmer beginnen.

1.2. Projektbezug

Bei dem Projekt handelt es sich um die Projektarbeit zur Abschlussprüfung von Niklas Grieger.

1.3. Ablage, Gültigkeit und Bezüge zu anderen Dokumenten

Die elektronische Version des Pflichtenheftes liegt in einem internen Netzlaufwerk. Die unterschriebene Version ist bei der IT-Leitung hinterlegt.

Dieses Pflichtenheft ist Teil der Projektdokumentation.

2. BETEILIGTE

Aufzählung der Beteiligten

Name	Vorname	Rolle
IT-Management		Auftraggeber
Delfs	Alexander	Ausbilder
Grieger	Niklas	Entwickler
Brückner	Torsten	Tester
Schreiber	Till	Tester
Grünther	Tim	Tester
Grünthal	Gerrit	Tester



3. PROJEKTMANAGEMENT

Das Projekt soll innerhalb des von der Handelskammer gesetzten Zeitrahmens vom 12. April 2017 bis zum 23. Mai 2015 durchgeführt werden.

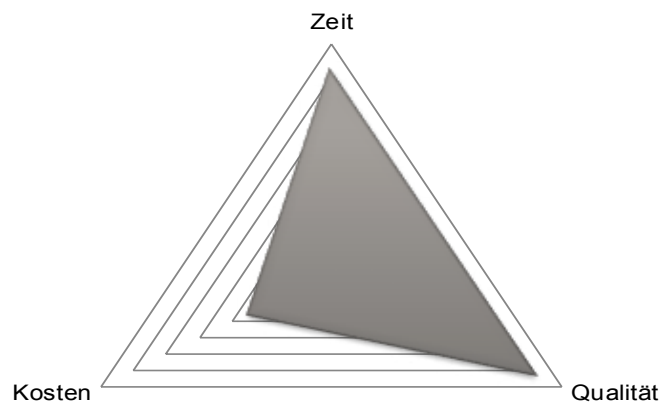
Die im Projektantrag aufgeführte Funktionalität soll in jedem Fall umgesetzt werden. Der Arbeitsaufwand wird bei Auszubildenden bei Berenberg mit 15€ angesetzt. Sollte das Projekt in der vorgegebenen Zeit und mit der Qualität nicht fertiggestellt werden können, würden weiteren Entwickler herangezogen werden.

Folgende Grafik eines „Magischen Dreiecks“ zeigt folgendes:

- Auf die Werte Qualität und Zeit wird gleichermaßen viel Wert gelegt
- Da die Kosten bekannt sind und sehr gering ausfallen wird hier der kleinste Wert genommen

Zeit	12,5
Qualität	12,5
Kosten	5

Musswert **30**





4. IST-ZUSTAND

Beschreibung des momentanen Zustandes.

4.1 Momentaner Zustand

Für Support Anfragen und Problemlösungen werden Kommunikationsmittel wie Email und/oder Telefon genutzt.

Dies kann unter anderem dazu führen, dass Mitarbeiter in der falschen Abteilung anrufen.

4.2 Geplante Veränderungen

Um dem Mitarbeiter die schnellstmögliche Hilfe anbieten zu können, soll ein Live Support Chat in die bereits bestehende Intranet Seite entwickelt werden.

4.3 Ablaufplan des Ist-Zustandes

Dies ist eine Darstellung vom Ist-Zustand.

Die Darstellung des Ablaufes ist nicht genau, da die Prozesse der Kommunikation personenabhängig verlaufen. Das nachfolgende Modell stellt einen allgemeinen Ablauf dar.



5. SOLL-ZUSTAND

Beschreibung des zukünftigen Zustandes

5.1 Sollzustand-Beschreibung

Der Live Support Chat soll die Arbeit des Fachbereichs und der IT effizienter und agiler gestalten.

Der Chat soll in unser Intranet eingebunden werden und nach Bedarf allen Mitarbeitern zur Verfügung gestellt werden.

Den Mitarbeitern soll mit diesem Chat die Möglichkeit gegeben sein, am Ort des Geschehens (die Intranetseite) direkt Fragen oder Anregungen an einen zuständigen Ansprechpartner zu verschicken, ohne dass diese sich erst informieren müssen, wer zu diesem Thema Auskunft geben kann.

Der Support soll offene Anfragen übersichtlich dargestellt bekommen, so dass er diese schnellstmöglich beantworten kann.

Die Nachrichten sollen solange wie nötig erhalten bleiben.

Der Chat soll nicht die herkömmlichen Kommunikationsmittel wie z.B. Telefon oder Email ablösen, sondern lediglich als weitere Kommunikationsmöglichkeit existieren.

5.2 Anforderungskatalog

Nr.	Anforderung
1.	Echtzeit-Kommunikation mit wenig Netzauslast
2.	Online/Offline Status
3.	Der Empfänger soll erkennen, sobald der Sender etwas schreibt.
4.	Integration in die Intranet Webanwendung
5.	Eigenständige Applikation/Einfache Einbindung
9.	Autorisierung durch den NT-User
10.	Notifikationen, wenn der Empfänger offline ist und angeschrieben wird
11.	Notifikationen, wenn der Empfänger online ist, aber die Chat geschlossen hat
12.	Möglichkeit die Anfrage als geschlossen zu markieren
13.	Sicherheit der Nachrichten muss gewährleistet sein
14.	Mehrere Sessions zulassen



5.3 Nutzen für den Anwender

Der Anwender wird die Möglichkeit haben, direkt aus der Intranet-Anwendung zu chatten. Es besteht die Option allen Mitarbeitern den Chat anzubieten.

5.4 Zielgruppe

Die Zielgruppen dieses Projektes sind folgender Tabelle zu entnehmen

Name(Zielgruppe)	Name(Ansprechperson)	Vorname(Ansprechperson)	Kürzel
IT-Management	Delfs	Alexander	AD

5.5 Schnittstellen

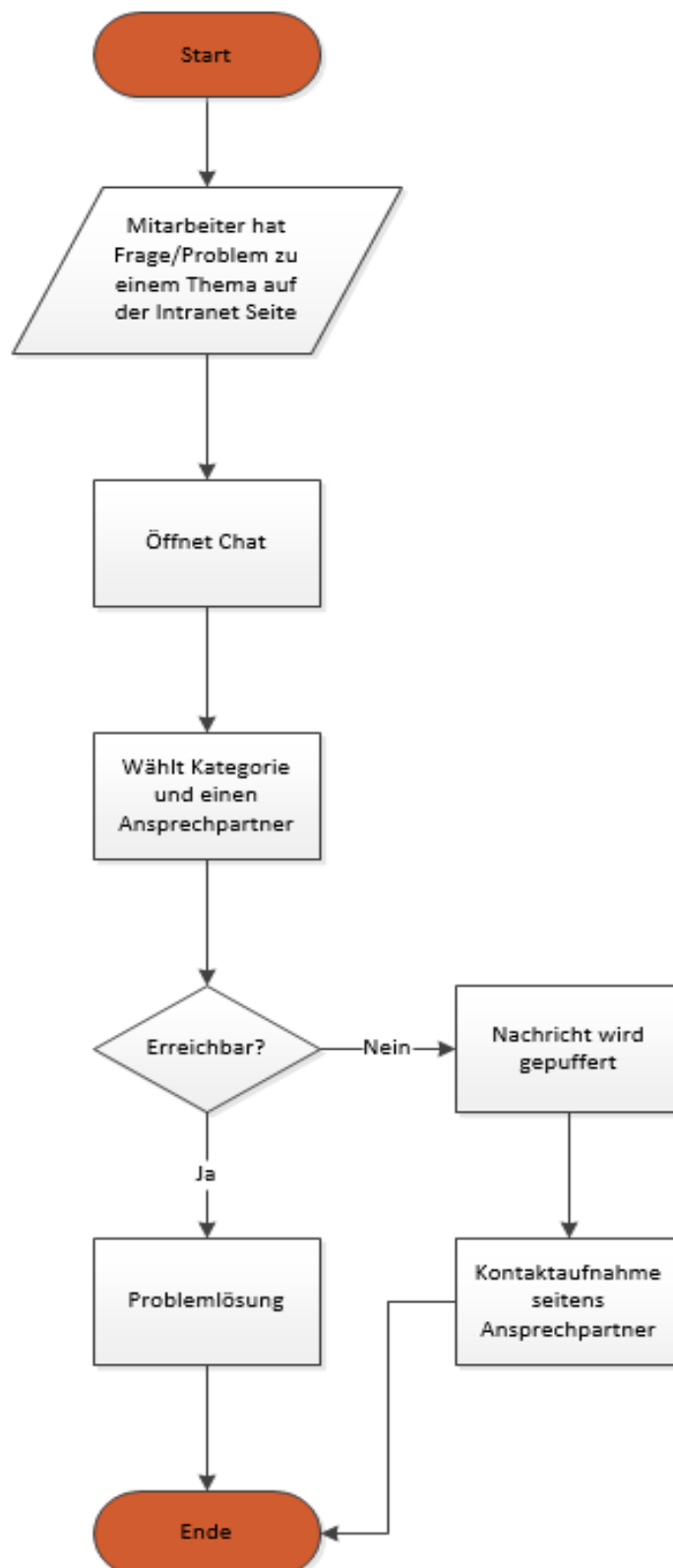
Anbindung an die Windows-Authentifizierung. Das bedeutet, dass der Mitarbeiter an einen NT-User im Active Directory gebunden ist. Mit diesem NT-User wird der Mitarbeiter dann authentifiziert.



5.6 Ablaufplan des Soll-Zustands

Dies ist eine Darstellung vom Soll-Zustand.

Der Ablauf kann personenabhängig von dem folgenden Modell abweichen. Das hier gezeichnete Modell stellt den allgemeinen Ablauf dar.





6.3 Testfälle

Nr.	Beschreibung	Erwartetes Testergebnis
1.	Starten der Anwendung	Komponenten werden geladen und initialisiert
2.	Authentifikation	Der Benutzer wird selbstständig authentifiziert
3.	Auswahl eines Ansprechpartners	Der Ansprechpartner ist auswählbar und der Chat öffnet sich
4.	Anzeige des bisherigen Chat Verlaufs zu der Anfrage	Der Chat Verlauf wird korrekt angezeigt
4.	Verfügbarkeit des Chats	Schreib und Lesefunktionen funktionieren
5.	Echtzeit Kommunikation	Nachrichten werden in Echtzeit beim Empfänger angezeigt
6.	Anzeige beim Empfänger, wenn der Sender schreibt	Erkenntliche Anzeige
7.	Online/Offline Anzeige	Der Mitarbeiter sowie der Support sehen wenn der Partner online/offline ist. Dies wird ebenfalls in Echtzeit aktualisiert
8.	Notifikationen, wenn der Angeschriebene offline ist	Erkenntliche Anzeige
9.	Notifikationen, wenn der Angeschriebene online ist (Anzeige nur wenn der Chat geschlossen ist)	Erkenntliche Anzeige
10.	Sicherheit gegen Ausführung von Scripts	Kein „Cross-Site Scripting“ möglich

Kosten der Ressourcen

Die Folgende Auswertung zeigt die Gesamtkosten für das Projekt.
Die Kosten belaufen sich lediglich auf die Projektzeit, sprich nachdem das Projekt beendet ist fallen keine weiteren Kosten, wie z.B. Lizenzkosten, an.
Die Kosten werden je Entwickler mit 76 € kalkuliert.
Die Kosten je Azubi werden mit 15€ kalkuliert.
In der Testphase sind 5 Personen involviert.

Aufgabe	Arbeitszeit/h	Zwischensumme (€)
Analyse	6	90,00 €
Entwurf	9	135,00 €
Realisierung	39	585,00 €
Test	4	788,00 €
Refactoring	2	30,00 €
Projektdokumentation	10	150,00 €
	70 Stunden	1.778,00 €
		Gesamtkosten
		1.778,00 €



8. KOSTENVERTEILUNG

Verteilung der Kosten für einen oder mehrere Kostenträger

Name	Anteil (%)
Allgemeine Bankprojekte	100

9. GENEHMIGUNG

Auflistung von Genehmigenden

Mit einer Unterschrift hier, wird zugestimmt, dass das Projekt, wie es hier im Pflichtenheft beschrieben ist, ablaufen/umgesetzt werden kann und vorerst keine Änderungswünsche bestehen.

Zu unterschreiben haben hier die Personen, die an diesem Projekt beteiligt sind bzw. mit diesem Projekt in Verbindung stehen.

Alexander Delfs

IT Ausbilder