Projektdokumentation



Projektdokumentation

LSC - Live Support Chat

Hamburg, 23.Mai 2017

**LSC** – **L**ive **S**upport **C**hat

**Fachinformatiker Fachrichtung Anwendungsentwicklung**

Abschlussprüfung Sommer 2017

Vorgelegt von

Niklas Grieger

Niklas Grieger

**Inhalt**

[1. Inhalt 1](#_Toc480459260)

[2. Einleitung 3](#_Toc480459261)

[2.1. Vorstellung des Betriebes 3](#_Toc480459262)

[2.2. Vorstellung des Entwicklers 3](#_Toc480459263)

[2.3. Vorstellung des Kunden 3](#_Toc480459264)

[2.4. Entwicklungsumgebung 3](#_Toc480459265)

[2.5. Projektziel 4](#_Toc480459266)

[2.6. Projektbegründung 4](#_Toc480459267)

[2.7. Projektschnittstellen 4](#_Toc480459268)

[2.8. Projektabgrenzung 4](#_Toc480459269)

[3. Projektplanung 5](#_Toc480459270)

[3.1. Zeitplan 5](#_Toc480459271)

[3.2. Ressourcenplanung 5](#_Toc480459272)

[3.3. Marktsituation 6](#_Toc480459273)

[4. Analysephase 6](#_Toc480459274)

[4.1. Ist-Analyse 6](#_Toc480459275)

[4.2. Wirtschaftlichkeitsanalyse 6](#_Toc480459276)

[4.2.1. „Make of Buy“ – Entscheidung 6](#_Toc480459277)

[4.2.2. Gründe für die Neuentwicklung 7](#_Toc480459278)

[4.2.3. Projektkosten 7](#_Toc480459279)

[4.2.4. Nutzen / Amortisation 7](#_Toc480459280)

[4.3. Lastenheft 8](#_Toc480459281)

[5. Entwurfsphase 8](#_Toc480459282)

[5.1. Zielplatform 8](#_Toc480459283)

[5.2. Schnittstellen 8](#_Toc480459284)

[5.3. Programmkonzept 8](#_Toc480459285)

[5.4. Pflichtenheft 10](#_Toc480459286)

[6. RealisierungsPhase 10](#_Toc480459287)

[6.1. Datenbank 10](#_Toc480459288)

[6.2. Back-End 11](#_Toc480459289)

[6.3. Front-End 11](#_Toc480459290)

[6.4. Sicherheit 12](#_Toc480459291)

[7. TestPhase 12](#_Toc480459292)

[8. Dokumentation 13](#_Toc480459293)

[8.1. Projektdokumentation 13](#_Toc480459294)

[8.2. Entwicklerdokumentation 13](#_Toc480459295)

[8.3. Kundendokumentation 13](#_Toc480459296)

[9. Quellennachweise 13](#_Toc480459297)

[10. Glossar 14](#_Toc480459298)

[11. Anhang 15](#_Toc480459299)

[11.1. Verwendete Ressourcen 15](#_Toc480459300)

[11.2. Diagramme 16](#_Toc480459301)

[11.2.1. IST-Zustand 16](#_Toc480459302)

[11.2.2. SOLL-Zustand 17](#_Toc480459303)

[11.2.3. Aktivitätsdiagramm 18](#_Toc480459304)

[11.2.4. Anwendungsfalldiagramm 19](#_Toc480459305)

[11.2.5. ERM 20](#_Toc480459306)

[11.2.6. Klassendiagramm – Database Code First 21](#_Toc480459307)

[11.2.7. Klassendiagramm – GUI 22](#_Toc480459308)

[11.3. Entwicklerdokumentation (Auszug) 23](#_Toc480459309)

[11.4. Kundendokumentation 24](#_Toc480459310)

[11.4.1. Einleitung 24](#_Toc480459311)

[11.4.2. Masken und Funktionalitäten 24](#_Toc480459312)

[11.5. Testprotokolle 31](#_Toc480459313)

[11.6. Code-Ausschnitte 36](#_Toc480459314)

[11.6.1. Datenbank - Das Code First Prinzip 36](#_Toc480459315)

[11.6.1. Backend 38](#_Toc480459316)

[11.6.2. Front End 48](#_Toc480459317)

# 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 war die Berenberg-Bank.

## Vorstellung des Betriebes

Die Berenberg-Bank wurde 1590 gegründet und ist heute eine der ältesten Privatbanken der Welt. Über 200 Mitarbeiter sind in dem Bereich IT, Organisation und Verwaltung beschäftigt. Der überwiegende Teil der eingesetzten Software wird selbst entwickelt.

## Vorstellung des Entwicklers

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

Schwerpunkt Anwendungsentwicklung. Seine Ausbildung wird voraussichtlich zum 31.07.2017 enden und er wird als Mitarbeiter in der .Net Web-Entwicklung übernommen. Zu seinen Aufgaben fällt unter anderem die Planung, Umsetzung und Testen von Projekten.

## Vorstellung des Kunden

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

## 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](http://www.draw.io) umgesetzt.

## Projektziel

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

## Projektbegründung

Durch diesen Chat wird es dem Benutzer möglich sein bei Fragen zu einer Bestimmten Kategorie auf der Intranet Seite 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 nicht erreichbar
* Der Hilfesuchende ruft jemanden an, der kein Zuständigkeit für das Thema hat
* Und vieles mehr

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

## Projektschnittstellen

Die Endbenutzer der Applikation sind alle Mitarbeiter der Firma Berenberg.

Um den Chat nutzen zu können müssen die IDs der Mitarbeiter in der Intranet Seite mit denen in der Chat Applikation synchron gehalten werden.

Außerdem ist die Applikation nur im Zusammenhang mit der Intranet Seite aufzurufen.

## Projektabgrenzung

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

Bei anderen „Global Chats“ wie z.B. Microsoft Lync müsste erst ein Ansprechpartner gefunden werden und dies kann mit viel Aufwand verbunden sein.

Ebenfalls die Anwendung Intranet Communication Service (ICS) von Herrn Grünther fokussiert sich auf die „Global Chat“ Funktion.

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

# Projektplanung

## 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 dem des Antrages verändert, da die Diagramme schneller als gedacht erstellt waren und somit mehr Zeit für das Pflichtenheft blieb.

Außerdem habe ich ein wenig Zeit der Refactoring Phase abgezogen und in die Realisierung geplant, da es zu Fehlern im Back-End Bereich kam.

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

## Ressourcenplanung

Im Anhang „[Verwendete Ressourcen](#_Das_Code_First)“ 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.

# Analysephase

## Ist-Analyse

Die Intranet Seite von Berenberg wird von jedem Mitarbeiter im Haus genutzt. Aktuell muss der Benutzer für jedes Problem oder Frage bezüglich der Oberfläche über mehrere Wege kommunizieren um eine Antwort zu erhalten.

Wie bereits im Abschnitt „[1.6 Projektbegründung](#_Projektbegründung)“ erwähnt, gibt es mehrere Szenarien die eintreten können, in dem der Hilfesuchende eventuell keine Hilfestellung bekommt bzw. bekommen kann oder es zu Umständlichkeit führt.

## Wirtschaftlichkeitsanalyse

## „Make of Buy“ – Entscheidung

Derweil existieren viele Angebote zu verschiedensten Chat Applikationen.

Meistens sind diese mit Kosten verbunden, meisten sogar Kosten/Nutzer. Dies ist bei einer großen Firma wie Berenberg mit hohen Kosten verbunden.

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

Zudem stehen die Server, über den die Kommunikation stattfindet, meist auch unter externer Aufsicht, sodass es bei Austausch von sensiblen Daten ein Datenschutz Problem geben könnte.

Aus diesen Gründen wird dieses Projekt intern umgesetzt.

## Gründe für die Neuentwicklung

Die Chat Applikation setzt den Fokus an einer anderen Stelle als das Altsystem. Es soll den Arbeitsprozess und die Kommunikation zwischen Kunde und Support fördern, 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
* Notifikationen
* modale Einbindung in die bestehen Intranet Seite zur unabhängigen Weiterentwicklung des Chats
* Nachrichten werden gepuffert
* Schutz gegen Cross-Site Scripting
* Responsive Design

## Projektkosten

Im Folgenden werden die anfallenden Projektkosten berechnet.

Es wird von folgenden Stundensätzen ausgegangen.

* Auszubildende/r: 0€
* Entwickler: 76€

|  |  |  |  |
| --- | --- | --- | --- |
| Vorgang | Mitarbeiter | Zeit | Gesamt |
| Entwicklung | 1x Auszubildender | 70h | 0€ |
| Test | 2x Auszubildender  2x Entwickler | 4h | 304€ |
| Gesamt |  |  | 304€ |

## Nutzen / Amortisation

Es ist schwierig die Wirtschaftlichkeit dieses Projekts zu errechnen, da kein sofortiger Nutzen erkennbar ist. 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.

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

## Lastenheft

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

Das komplette Lastenheft befindet sich im Anhang „[Lastenheft](#_Lastenheft_1)“.

# Entwurfsphase

## Zielplatform

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

Der Chat soll eine für sich lauffähige Anwendung sein, somit wird die Unabhängigkeit bei Wartung gewährleitet und der Chat kann auch in anderen Webseiten modular eingebunden werden.

## 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.

## Programmkonzept

Allgemein

Der Client des Benutzers kommuniziert über JavaScript und JQuery mit dem Server Hub, welcher die Verbindungen bzw. Connections behandelt und verarbeitet. Der Server Hub ist so gesehen der Kontroller 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 Intranet Seite öffnet wird im Hintergrund eine Methode mit der Mitarbeiter ID des eingeloggten Benutzers als Parameter aufgerufen. Die Mitarbeiter ID existiert bereits in der Intranet Seite 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 Benutzter unten rechts auf der Intranet Seite auf den Button

„Live Support Chat“ klickt.

Bevor die Partial View von der Methode zurückgegeben wird, werden alle relevanten Daten über den eingeloggten bzw. 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 ausgewählt und einen Ansprechpartner ausgewählt hat, kann dieser anfangen Nachrichten zu schicken. Per

„Enter“ Taste oder Klick auf den Senden Button, wird die Funktion aufgerufen um die Nachricht in die Datenbank zu schreiben, sowie die Funktion, die die Nachricht in Echtzeit anzeigt.

Sobald der Benutzer gegenüber anfängt zu schreiben, sieht der Benutzer dass dieser schreibt

„[Benutzer] schreibt…“

Zudem wird in der Benutzer Info angezeigt, ob der gegenüber 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 gegenüber die Nachricht lesen kann oder nicht verfügbar ist.

Außerdem bekommt der gegenüber 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 Messages 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 encodiert.

Sicherheit:

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

## Pflichtenheft

Am Ende der Entwurfsphase wurde mithilfe des Lastenhefts und der Vorarbeit aus der Entwurfsphase das Pflichtenheft verfasst. Diese dient als Leitfaden für die Implementierung des Projektes.

Das komplette Pflichtenheft befindet sich im Anhang „[Pflichtenheft](#_Pflichtenheft)“.

# RealisierungsPhase

Zur Realisierung habe ich folgende Technologien verwendet:

- C#

- ASP.Net MVC

- SignalR

- Entity Framework

- HTML, JavaScript (JQuery)

## Datenbank

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

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](#_Datenbank_-_Das)“.

Die Datenbank wurde mit Beispieldaten befüllt.

In der finalen Version werden die User IDs derjenigen entsprechen, die auch in der internen Mitarbeiter Tabelle vorhanden sind.

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 die ID, der ID aus der internen Mitarbeiter Tabelle entsprechen und Daten wie Vorname, Nachnahmen aus der internen Datenbank geholt
  + SupportGroup
    - Diese Tabelle beinhaltet die verschiedenen Kategorien, die auf der Seite exestieren
  + User2SupportGroup
    - Diese Tabelle dient als Zwischentabelle. Hier sind die User als Support einer bestimmten Kategorie zugewiesen
  + Chatlog
    - Diese Tabelle beinhaltet den eigentlichen Chatverlauf

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

## Back-End

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

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

Außerdem gibt es die Methode, die den fertig gerenderten Chat als Partial View zurückgibt. Diese Methode wird von der Intranet Seite gerufen, sobald auf den Button „Live Support Chat“ geklickt wurde. Mit dem übergebenen Parameter, die Mitarbeiter ID des NT-Users, 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 Ends sind im Anhang „Backend“ zu finden.

## 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 Intranet Seite) mit der Mitarbeiter ID als Parameter aufgerufen wird, wird diese Mitarbeiter 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 User in Echtzeit das dieser User gerade „Online“ ist
* 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 User in Echtzeit das dieser User gerade „Offline“ ist

*Weitere Code Ausschnitte sind im Anhang zu finden*

## Sicherheit

Abfangen von Gefahren durch Cross Site Scripting:

* Durch das Encodieren der Nachricht kann kein Schädlicher Code auf dem Server ausgeführt werden, wie z.B. durch SQL-Injection.

# TestPhase

Getestet wurde die Anwendung einerseits von dem Entwickler selber. Hierbei waren die Testverfahren Funktionalitäts- und Performance-Tests, welche nach jeder Änderung erfolgten. 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 Nachriten empfangen werden
* Funktioniert das „Live Chatting“
* Werden die Nachrichten korrekt angezeigt
* Wird angezeigt, ob der gegenüber gerade schreibt
* Ist die online/offline Anzeige korrekt?
* Funktionieren die Notifikationen, wenn der angeschriebene Offline ist
* Funktionieren die Notifikationen, wenn der angeschriebene Online ist (Anzeigen nur wenn der Chat geschlossen ist)

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

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

Die Testprotokolle befinden sich im Anhang.

# Dokumentation

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

## Projektdokumentation

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

## 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“.

## 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.

# Quellennachweise

Entity Framework im Code First Prinzip:

* <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](https://docs.microsoft.com/en-us/aspnet/signalr/overview/security/introduction-to-security%23authentication)

Hilfe bei Problemen

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

# 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 |
|  |  |

# Anhang

## 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**

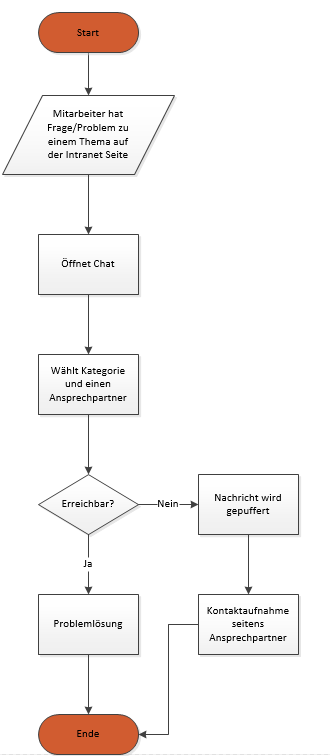
* + Auftraggeber (Festlegung der Anforderungen, Abnahme Projekt)
  + Auszubildender (Umsetzung des Projektes)
  + 2x Auszubildende/r (manuelles Testen)
  + 2x Entwickler (manuelles Testen)

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

## Diagramme

## IST-Zustand

## SOLL-Zustand



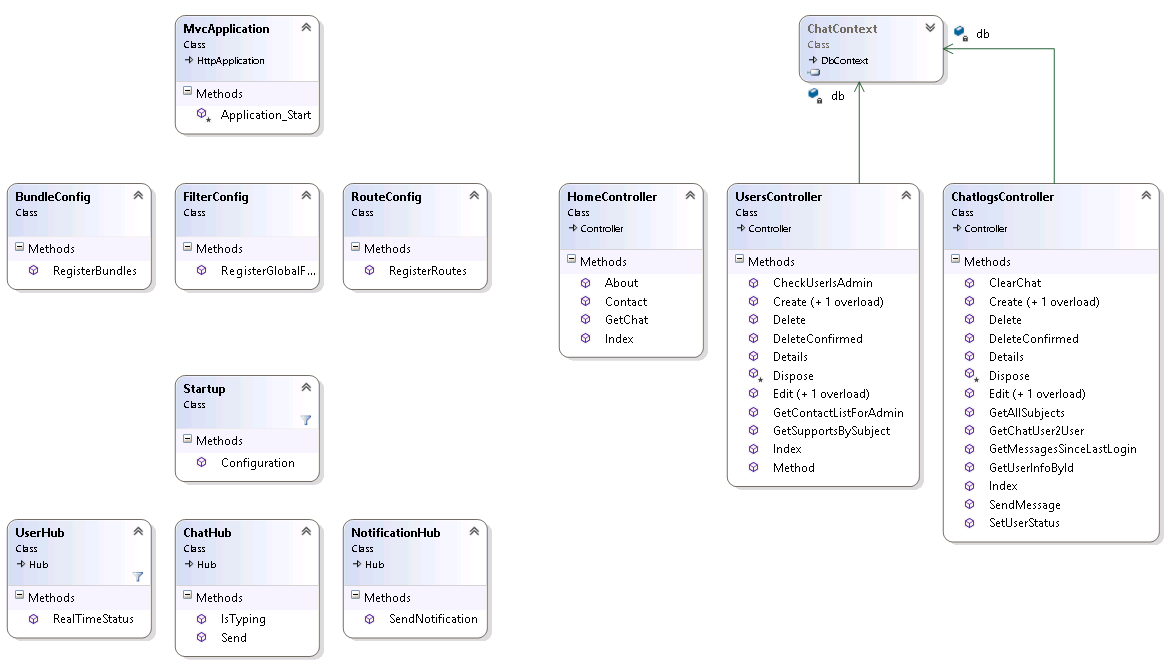
## U:\Projektarbeit\Diagramme\Aktivitätsdiagramm.pngAktivitätsdiagramm

## U:\Projektarbeit\Diagramme\Anwendungsfalldiagramm.pngAnwendungsfalldiagramm

## U:\Projektarbeit\Diagramme\ERM_Datenbank.pngERM

## U:\Projektarbeit\Diagramme\UML_ClassDiagram\DBShema.pngKlassendiagramm – Database Code First

## Klassendiagramm – GUI



## U:\Projektarbeit\Entwicklerdoku\EntwicklerDoku_Auszug.pngEntwicklerdokumentation (Auszug)

## Kundendokumentation

## Einleitung

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

## Masken und Funktionalitäten

Der erste Schritt ist das Öffnen der Intranet Webseite.

Falls Sie nicht wissen, wie Sie zu der Webseite gelangen können, fragen Sie Ihre Kollegen.

Nachdem Sie die Webseite geöffnet haben, sehen Sie folgende Grafische

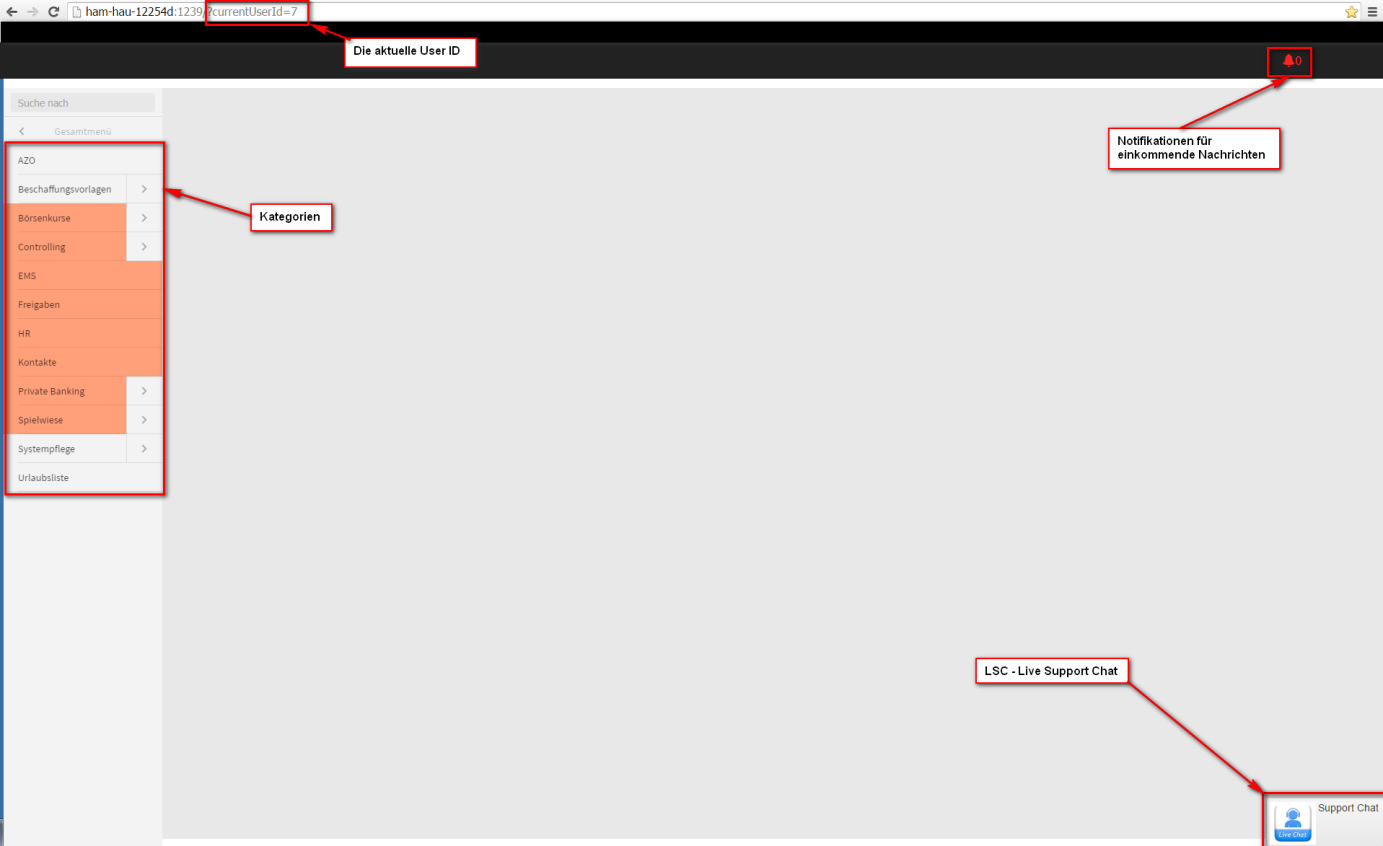
Oberfläche:

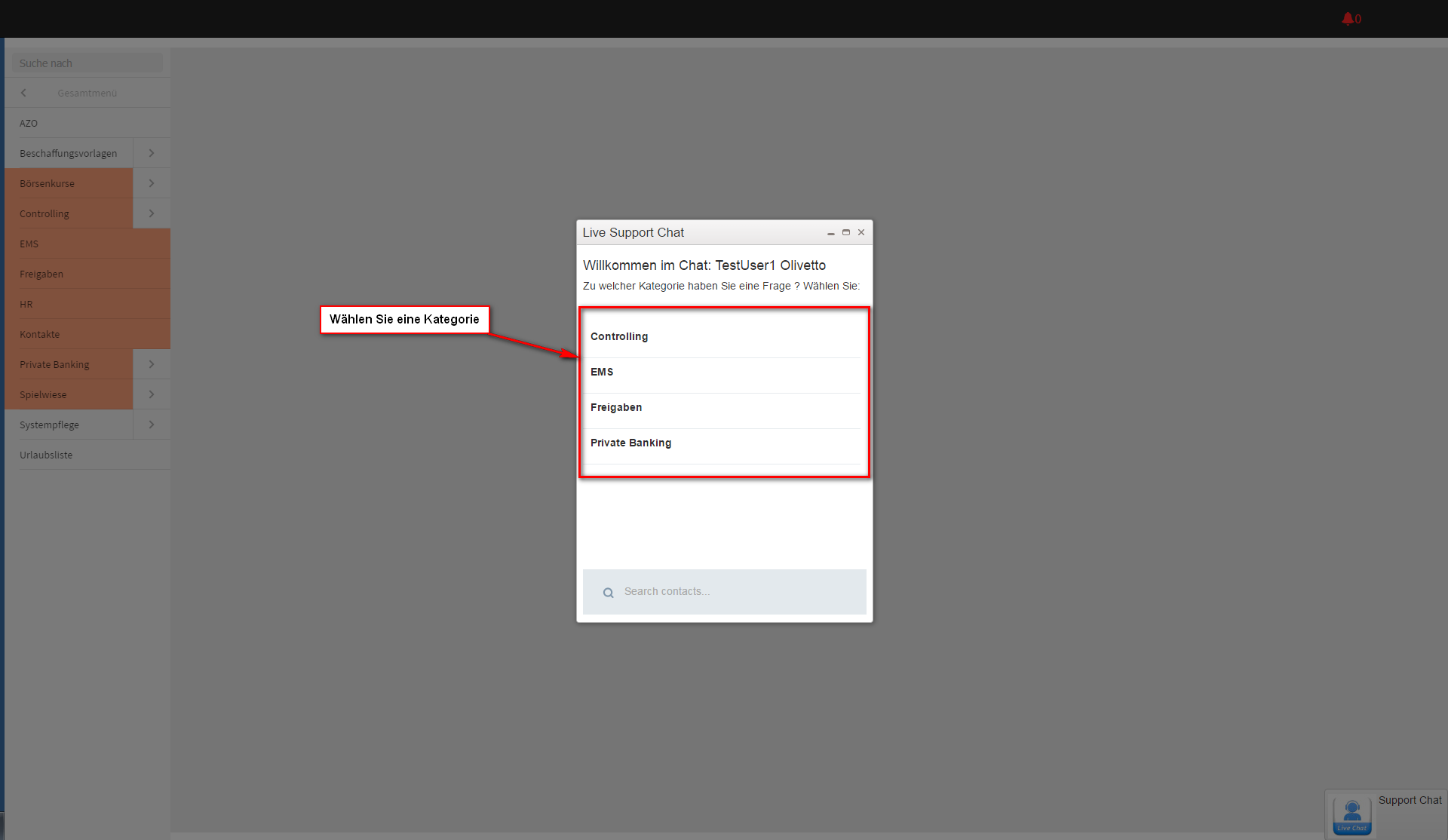
Hier sehen Sie schon im Bild einige Hinweise zu den wichtigsten Themen, die mit dem Chat zusammenhängen.

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 die Button mit der Linken Maustaste klicken, öffnet sich der Chat in einem kleinen Fenster, mittig des Bildschirmes. Siehe nächstes Bild.



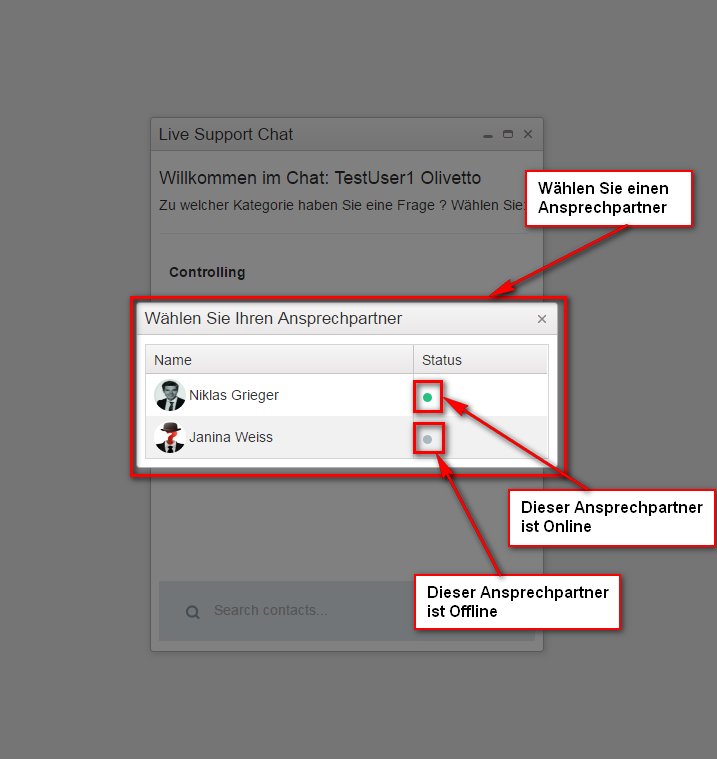
Hier können Sie eine Kategorie, zu dem Sie eine Frage haben, auswählen.

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

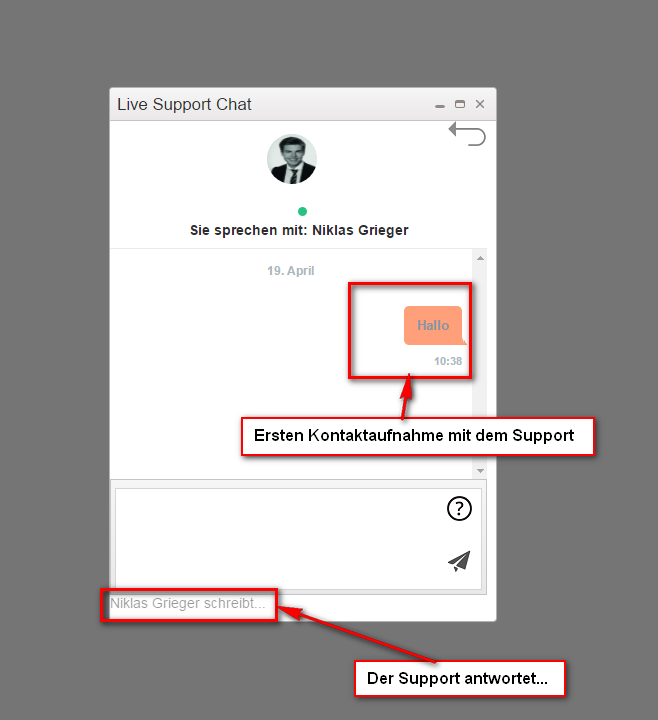
Ihnen ist aber auch die Möglichkeit gegeben jemanden anzuschreiben, der offline ist. Jedoch kann dieser die Nachricht erst lesen, wenn dieser erneut online ist.

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

In den folgenden Bildern werden Beispiele zu einer Konversation aufgeführt.



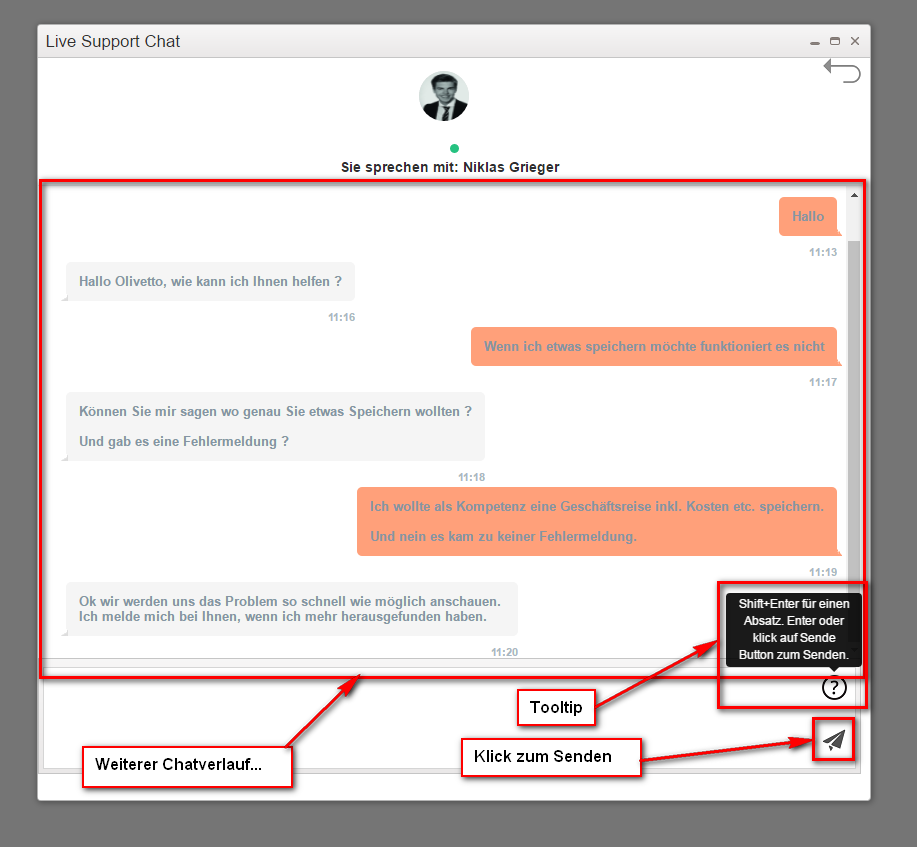
Nachdem linksklick auf einen der Auswahlmöglichkeiten ändert sich die Ansicht und Sie werden mit dem zuvor ausgewählten Ansprechpartner verbunden.

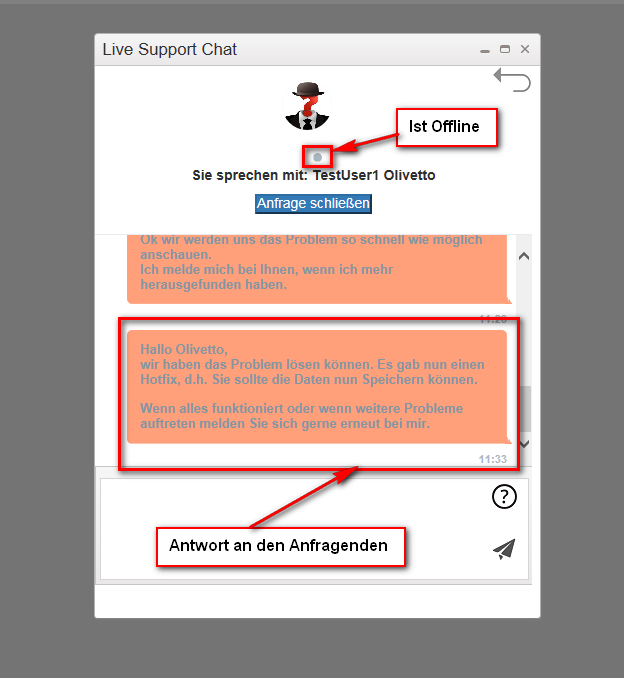


Sie können die Nachricht durch „Enter“ oder durch klick auf den Senden Button abschicken.

Sie können einen Absatz mit der Tastatur Kombination „Shift+Enter“ Einfügen.

Dies finden Sie auch als Hinweis/Tooltip, wenn Sie die Maus über das Fragezeichen im Chat halten.

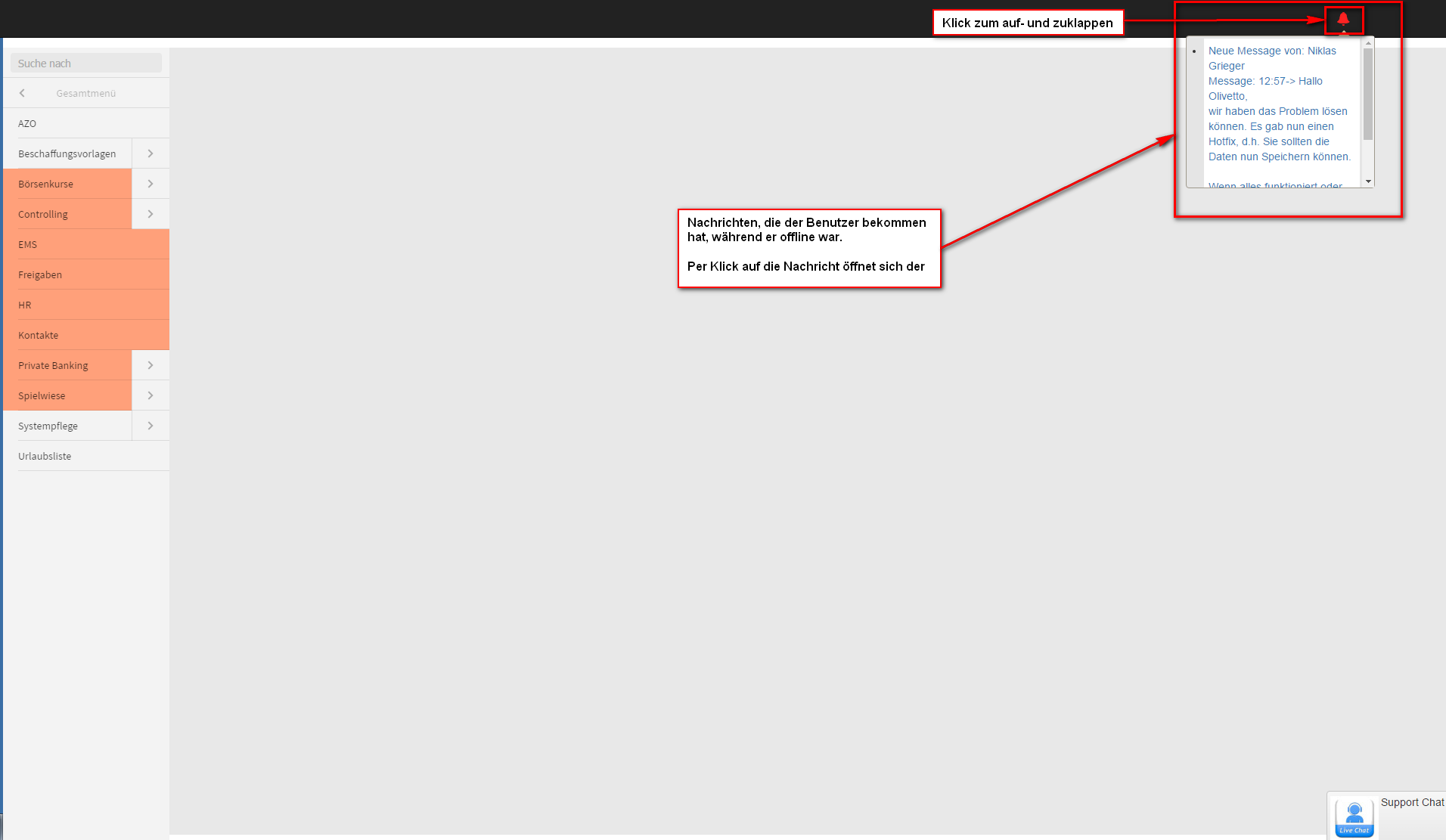
Hier ein Beispiel zum weiteren Chatverlauf.

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.

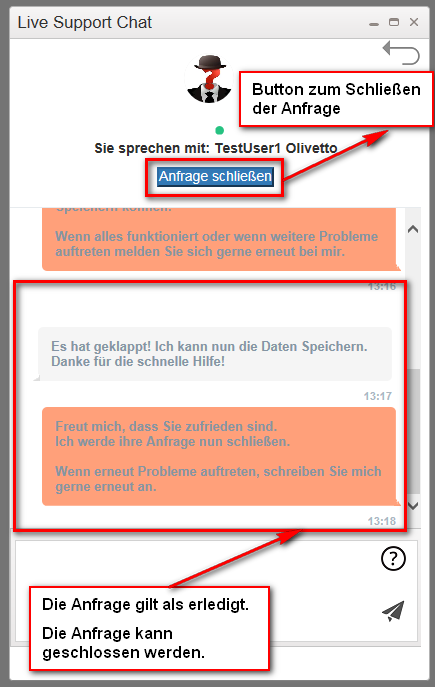
Sobald Sie die Seite Intranet erneut aufrufen bzw. „Online“ gehen bekommen Sie die Nachrichten seit ihrem letzten Besuch übersichtlich dargestellt und Sie können sich per Klick auf diese Notifikation direkt mit dem Sender verbinden.

Der Anfragende betritt die Seite und geht „Online“.



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 dem Beispiel wurde das Problem gelöst und die Anfrage wird geschlossen.



## 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 Nachriten empfangen werden
* Funktioniert das „Live Chatting“
* Werden die Nachrichten korrekt angezeigt
* Wird angezeigt, ob der gegenüber gerade schreibt
* Ist die online/offline Anzeige korrekt?
* Funktionieren die Notifikationen, wenn der angeschriebene Offline ist
* Funktionieren die Notifikationen, wenn der angeschriebene Online ist (Anzeigen 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 gegenüber 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(Anzeigen 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(Anzeigen 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(Anzeigen nur wenn der Chat geschlossen ist) | Keine Vorfälle | OK |
| Cross-Site-Scripting ist nicht möglich | Keine Vorfälle | OK |

## Code-Ausschnitte

## 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 Befehlt 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

## 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 inlude the Layout, if Partial param is false: an Partial View without the Layout </returns>

        [AllowCrossSiteJson]

        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 (Auschnitt)**

...

/// <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.supp ortgroup\_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)

        {

            Conole.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

    {

        //Enthlt 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 System.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);

        }

    }

}

**NotficationHub.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 client 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();

        }

    }

}

## 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 Intranet Seite 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">

    <titleLSC – 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>

                <div class="noti-content">

                    <div class="noti-top-arrow"></div>

                    <ul id="notiContent"></ul>

                </div>

                @\*------------------------------------------ Notification ------------------------------------------------\*@

            </div>

        </div>

    </div>

    <div class="container body-content">

        <div id="chatWindow" style="display:none;"></div>

        @RenderBody()

        <button id="btnChat" style="bottom: 0px; right: 0px; position:fixed;">Support Chat</button>

    </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>

|  |  |  |
| --- | --- | --- |
|  |  | Joh. Berenberg, Gossler & Co. KG Neuer Jungfernstieg 20 20354 Hamburg Telefon +49 40 350 60-0 Telefax +49 40 350 60-900 www.berenberg.de info@berenberg.de |