UNIVERSITÄT OSNABRÜCK

Institut für Informatik
AG Medieninformatik

*Bachelorarbeit*

# Konzeption und prototypische Implementierung eines Live Chats mit Affiliate-Marketing-Komponente

Milutin Culibrk

März 2014

Erstgutachter: Prof. Dr. Oliver Vornberger
Zweitgutachter: Prof. Dr.-Ing. Elke Pulvermüller

# Zusammenfassung

Anhand dieser Arbeit soll festgestellt werden, inwiefern es möglich ist, Methoden aus dem Affiliate Marketing in einem Live Chat zu nutzen. Dazu entsteht ein Prototyp einer Live-Chat-Webanwendung, der mit Funktionen erweitert ist, welche es ermöglichen, Produkte aus Affiliate System in eine Unterhaltung einzubinden. Genauer gesagt, es entsteht eine Microsoft ASP.NET MVC 4 Webanwendung, die neben gewöhnlichen Funktionen eines Live Chats, Funktionen enthält, um auf Schnittstellen (APIs) von Affiliate System zuzugreifen. Diese Schnittstellen liefern Links zu Produkten, die über die Agenten-Konsole in einen Live Chat eingebunden werden. Dadurch bekommt der Live Chat die Funktion einer Werbefläche.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Listings

# Abkürzungsverzeichnis

**ADO** ActiveX Data Objects

**API** Application Programming Interface

**ASP** Active Server Pages

**CAMC** Chat with Affiliate Marketing Component

**CLR** Common Language Runtime

**CLI** Common Language Infrastructure

**CSV** Comma-separated values

**EDM** Entity Data Model

**EF** Entity Framework

**GUI** Graphical User Interface

**GUID** Global Unique Identifier

**HATEOAS** Hypermedia as the engine of application state

**HTML** Hypertext Markup Language

**HTTP** Hypertext Transfer Protocol

**IP** Internet Protocol

**JIT** Just in Time

**JSON** JavaScript Object Notation

**LINQ** Language Intermediate Query

**MVC** Model View Controller

**MSIL** Microsoft Intermediate Language

**PHP** Hypertext Preprocessor

**REST** Representational State Transfer

**SDK** Software Development Kit

**SQL** Structured Query Language

**TCP** Transmission Control Protocol

**UML** Unified Modeling Language

**URI** Uniform Resource Identifier

**URL** Uniform Resource Locator

**VB** Visual Basic

**VoIP** Voice over IP

**WWW** World Wide Web

**XML** Extensible Markup Language

# Kapitel 1

# Einführung

Dieses Kapitel gibt eine kurze Einführung in die Thematik und Motivation dieser Arbeit. Anschließend wird ein kurzer Überblick, über die inhaltliche Struktur dieser Ausarbeitung gegeben.

## 1.1 Motivation

Im World Wide Web bezeichnet der Begriff *Live Chat* einen Kommunikationsweg, der es Besuchern einer Webseite ermöglicht direkt mit Webseitenbetreibern und zugehörigen Personen zu kommunizieren. Die Kommunikation geschieht über den direkten Austausch von gewöhnlichen Textnachrichten. Neben der E-Mail und der Telefonischen-Kontaktaufnahme, hat sich der *Live Chat* als immer beliebterer Kommunikationsweg etabliert [25].

Weiter werden mit *Live Chats* Webanwendungen bezeichnet, die Webseitenbetreibern eine Infrastruktur bieten, um in Echtzeit mit ihren Besuchern, textuell zu kommunizieren. Dies kann beispielsweise zu Online-Marketing-Zwecken geschehen. So ist es Kundenberatern möglich, durch Live Chats ihre Kunden in Echtzeit zu erreichen und ihnen aktuelle Information zu Produkten zu geben, oder Hilfestellung zu leisten. Laut großen Live Chat - Anbietern [22][20] soll diese Art der Kundenbetreuung, im Vergleich zur E-Mail, oder Ticketsystemen, effizienter sein und die Kundenzufriedenheit steigern.

Live Chats werden auch genutzt, um Hilfesuchenden und Helfern im Internet, eine Infrastruktur für den Informationsaustausch zu bieten. Ein Beispiel hierfür sind *Experten-Communities* [23][21]. In *Experten-Communities* können sich Hilfesuchende in verschiedensten Angelegenheiten durch *Experten* beraten und helfen lassen. Die Kommunikation geschieht in der Regel entweder audiovisuell durch VoIP-Telefonie, Video-Telefonie, oder textuell durch E-Mail, Live Chats oder Ticketsysteme.

In manchen Beratungsangelegenheiten oder Problemlösungen ist dem Hilfesuchenden geholfen, indem ihm ein Produkt empfohlen wird. Das Produkt ist dann Teil der Problemlösung. An dieser Stelle knüpft das *Affiliate Marketing* an. Affiliate Marketing bezeichnet ein Provisionsgeschäft, basierend auf Vermittlungsprovision, dass im Internet stattfindet.

Ein typischer Vertreter dieses Provisionsgeschäfts ist die herkömmliche Bannerwerbung, wie sie auf unzähligen Webseiten vorkommt. Ein Webseitenbetreiber (Affiliate) stellt eine Fläche auf seiner Webseite einem Werbetreibenden (Advertiser) zur Verfügung. Der Werbetreibende möchte ein Produkt vermarkten und platziert dort sein Werbemittel, wie zum Beispiel einen Werbebanner oder einen Link. Nimmt ein Besucher dieses Werbemittel wahr und führt diesbezüglich eine Aktion aus, beispielsweise einen Klick auf das Werbemittel, so wird dem Webseitenbetreiber, je nach Provisionsmodell, eine Provision ausgezahlt.

Dieses Modell lässt sich auf eine Kommunikation mittels Live Chat übertragen. Beispielsweise befinden sich Experte und Hilfesuchender in einem Live Chat. Nimmt der Hilfesuchende, ein ihm empfohlenes Produkt wahr und erwirbt es, so wird dem Experten eine Provision ausgezahlt. Das Werbemittel ist in diesem Fall die Produktempfehlung und der Chat fungiert als Werbefläche.

Ziel dieser Arbeit ist es, ein Konzept einer Live Chat-Anwendung (CAMC)[1], zu entwerfen und zu implementieren. Diese Anwendung soll zeigen, ob und wie es möglich ist, Live Chats und Affiliate Marketing zu verbinden. So soll beispielsweise Experten die Möglichkeit gegeben werden, Produkte von Advertisern in die Unterhaltung einzubinden, um beim Erwerb dieser Produkte durch einen Hilfesuchenden, eine Provision zu erhalten.

Die Idee, Affiliate Marketing in Live Chats anzuwenden stammt aus einem Praktikum bei der Firma Comtrade d.o.o. in Bosnien und Herzegowina. Dort habe ich mich unter Anderem im Rahmen eines internen Projektes mit dem Thema Live Chats auseinandergesetzt, Ideen für dieses Projekt gesammelt und mich in die Technologien des Microsoft .NET Frameworks eingearbeitet.

## 1.2 Gliederung

Diese Ausarbeitung gliedert sich in zwei Teile. Im ersten Teil werden wichtige Konzepte, Technologien und Frameworks vorgestellt, die im Rahmen der Implementierung von CAMC zum Einsatz kommen. Erst wird eine kurze Einführung in die Funktionsweise von Live Chats gegeben. Anschließend folgt eine kurze Einführung in das Affiliate Marketing.

[1] Chat with Affiliate Marketing Component

Der Schwerpunkt liegt jedoch auf dem Microsoft MVC 4 Framework zum Erstellen von Webanwendungen und zugrundeliegenden Konzepten.

Im zweiten Teil wird die Implementierung von CAMC anhand der vorgestellten Technologien erläutert. Auf eine kurze Erläuterung der Vorgehensweise, folgt eine Einführung in die Architektur der Serveranwendung. Anschließend werden grundlegende, in CAMC implementierte Technologien, wie die Affiliate-System-Integration und die Verarbeitung eines Chats behandelt. Schließlich folgt eine Analyse kritischer Systempunkte und eine kurze Zusammenfassung der Arbeit.

# Kapitel 2

# Grundlagen und Stand der Technik

In diesem Kapitel werden Grundlagen und verwendete Technologien erklärt, die im Rahmen dieser Arbeit zum Einsatz kamen. Im ersten Abschnitt wird die Funktionsweise von Live Chats erläutert und eine kleine Einführung in das Affiliate Marketing gegeben. Anschließend wird der Fokus auf das Microsoft MVC 4 Framework gelegt und für das Verständnis dieser Arbeit, grundlegende Konzepte und Technologien besprochen.

## 2.1 Live Chats

Live Chats sind Webanwendungen, die Webseitenbetreibern eine Infrastruktur bieten, um in Echtzeit mit ihren Besuchern, textuell zu kommunizieren. Sie werden in der Regel als *Software as a Service* angeboten. Das heißt, Live Chat-Anwendungen werden nicht auf der eigenen IT-Infrastruktur betrieben, sondern ein externer Dienstleister stellt die nötige IT-Infrastruktur und Software zur Verfügung. Kommunikationspartner eines Live Chats sind in der Regel Besucher einer Webseite, im Weiteren *Klienten* genannt und *Agenten*, die zu einem Webseitenbetreiber gehören. Agenten können beispielsweise Kundenberater sein, die durch ihre Unternehmenswebseite Kunden beraten oder akquirieren.

Typischerweise besteht ein Live Chat aus drei Komponenten, jeweils einer Benutzeroberfläche für Klient und Agent und einer Komponente für die Datenverarbeitung. Die Benutzeroberfläche für Klienten ist meist ein einfaches Chat-Fenster, welches im Browser geöffnet ist. Die Benutzeroberfläche für Agenten, als Webanwendung, oder Desktop-Anwendung enthält wesentlich mehr Funktionen. Es ist beispielsweise möglich Website-Besucher zum Chat aufzufordern, oder automatische Antworten zu versenden. Außerdem werden dem Agenten weitaus mehr Informationen zum Klienten angezeigt. Diese umfassen, falls vorhanden, sowohl Personenbezogene Daten, als auch beispielsweise ortsbezogene

Daten. Das Chat-Fenster auf Seite der Klienten, wird *Klient-Chat* genannt, die Benutzeroberfläche für Agenten, *Agenten-Konsole*. Große Live Chat-Anbieter wie LivePerson [22], oder LiveChat Inc. [20] bieten darüber hinaus Klient-Chat und Agenten-Konsole auch als mobile Anwendung für Smartphones an. Oft wird zusätzlich eine Programmierschnittstelle (API) angeboten, um beispielsweise eigene Chat-Anwendungen zu erstellen, oder vorhandene Anwendungen um Funktionalitäten zu ergänzen. Die dritte Komponente eines Live Chats ist eine serverseitige Anwendung und regelt unter Anderem die Nachrichtenübermittlung.

## 2.2 Affiliate Marketing

Affiliate Marketing bezeichnet ein im Internet stattfindendes Provisionsgeschäft, das auf Vermittlungsprovision basiert. Ein kommerzieller Anbieter *(Advertiser)*, der beispielsweise ein Produkt vermarkten möchte, vergütet seine Vertriebspartner *(Publisher)* erfolgsorientiert durch eine Provision. Typischerweise stellt ein Publisher, eine freie Fläche auf seiner Webseite zur Verfügung stellt, um für Produkte des Advertiser zu werben. Dazu werden zum Beispiel Links oder Banner auf der Webseite platziert. Weitere Vertriebskanäle sind vergleichsweise das E-Mail- oder Suchmaschinenmarketing. Für jede Transaktion, die durch die Werbemaßnahme des Publisher generiert wird, erhält dieser je nach Provisionsmodell, eine Provision.

Häufige Provisionsmodelle sind unter Anderem [18]:

- *Pay-per-Click*, Provisionierung per Klick auf ein Werbemittel.
- *Pay-per-Lead*, Provisionierung bei Kontaktaufnahme des Kunden, beispielsweise durch Anmeldung zu einem Newsletter.
- *Pay-per-Sale*, Provisionierung bei Erwerb des Produktes durch den Kunden.

Es gibt verschiedene Möglichkeiten, wie Publisher und Advertiser zusammenfinden. Ein häufiges Szenario ist folgendes: Ein Publisher bewirbt sich bei einem *Partnerprogramm* des Advertisers. Entspricht der Publisher den Kriterien des Partnerprogramms, so wird dieser berechtigt für den Advertiser zu werben und seine Werbemittel zu nutzen. Kriterien sind beispielsweise die Zielgruppe und der Inhalt der Webseite. Ein Beispiel für ein Partnerprogramm ist das Amazon PartnerNet [6]. Dort kann man sich als Publisher registrieren und anschließend Werbemittel zu Produkten von Amazon auf seiner Webseite platzieren. Werbemittel sind unter Anderem, Links oder Banner. Gelangt ein Webseiten-Besucher über ein Werbemittel zu einem Produkt auf Amazon und kauft es, so wird dem Publisher eine Provision ausgeschüttet (*Pay per Sale*).

Ein weiteres Szenario: Die Webseite eines Publishers ist gut besucht und daher eine at-

traktive Werbefläche. Möchte ein Advertiser seine Werbung auf dieser Webseite platzieren, so kann er dort Werbeflächen für einen bestimmten Zeitraum mieten. Yahoo [1] ist eine der meistbesuchten Webseiten weltweit (Alexa Rank: Platz 4 [3]). Und damit eine attraktive Werbeplattform. Will ein Advertiser auf den Webseiten von Yahoo werben, so stellt ihm Yahoo Tools zur Verfügung, um gezielt Werbemittel zu platzieren [2]. Je nach vereinbartem Provisionsmodell, erhält Yahoo dafür ein Entgeld.

Diese Szenarien beinhalten gewisse Nachteile. Eine steigende Zahl an Partnerprogrammen seitens der Publisher, oder gemieteter Werbeflächen seitens Advertiser, enthält einen hohen Verwaltungsaufwand. Weiter sind die Kosten, um Werbung auf viel besuchten Webseiten zu platzieren sehr hoch und können das Budget vieler Advertiser überschreiten. Andererseits haben Publisher oft Schwierigkeiten ihre Restwerbeflächen erfolgreich zu besetzen. Diese und weitere Gründe führten dazu, dass sogenannte *Werbenetzwerke*, im Laufe des letzten Jahrzehnts, immer beliebter wurden [15].

Werbenetzwerke sind Unternehmen die Advertiser und Publisher zusammenbringen. Sie aggregieren Werbeflächen und kategorisieren sie nach Zielgruppe, Benutzerverhalten, Geschlecht, Alter und anderen Merkmalen. Bei Übereinstimmung der Zielgruppen von Website und Werbemittel werden passende Werbemittel eingeblendet[33]. Diesen Mechanismus nennt man *Targeting*.

Werbenetzwerke können durch folgende Eigenschaften charakterisiert werden [15]:

- Sie wickeln die Platzierung und Auslieferung von Werbemitteln ab.
- Sie protokollieren Aktionen von Webseiten-Besuchern (*Tracking*).
- Sie ermöglichen es Advertisern, ihre Werbemittel über eine zentrale Stelle, an mehrere Publisher auszuliefern.
- Sie übernehmen den Aufwand seitens der Publisher, Advertiser zu suchen die auf ihrer Webseite werben wollen.
- Sie ermöglichen es Advertisern, verschiedene Zielgruppen anzusprechen.

Auf einer höheren Ebene werden Patnerprogramme und Werbenetzwerke als *Affiliate Systeme* zusammengefasst. Wie die obigen Beispiele zeigen, sind Affiliate Systeme internetbasierte Vertriebslösungen die Publisher und Advertiser zusammenführen.

### 2.2.1 Targeting

Unter Targeting im Online Marketing versteht man das Adressieren von Werbung an bestimmte Benutzergruppen. Das Ziel des Targeting ist es die Relevanz der Werbung für den Nutzer zu maximieren und damit Streuverluste zu vermeiden [8]. Mit Streuverlusten

bezeichnet man geschaltete Werbung, die nicht die gewünschte Zielgruppe erreicht. Es gibt verschiedene Techniken mit denen Targeting umgesetzt wird. Im Folgenden werden drei häufig angewendete Targeting-Technologien kurz erläutert[8].

**Technisches Targeting** basiert auf Information über Soft- und Hardwareumgebung des Benutzers. Es werden beispielsweise Informationen über Uhrzeit, Datum, Betriebssystem, Browser, Bandbreite, Provider, Bildschirmauflösung, etc. genutzt, um passende Werbung zu schalten. Diese Informationen können auch genutzt werden um Werbung nach geografischen Merkmalen des Benutzers zu schalten. So lässt sich in vielen Fällen an Hand der IP-Adresse des Benutzers eine Aussage zu seinem Ort treffen, beispielsweise in welchen Land er sich befindet.

**Keyword Targeting** nutzt Suchanfragen auf Suchmaschinen, um auf den Ergebnisseiten passende Werbung zu schalten. Ein prominentes Beispiel ist Google AdWords [14]. Dort werden passend, zu den in die Suchmaske eingegebenen *Keywords*, Anzeigen auf der Ergebnisseite geschaltet.

**Contextual Targeting** basiert auf den Inhalten einer Webseite. Hier wird Werbung bei thematischer Übereinstimmung mit der Webseite eines Publisher geschaltet. Auch dazu ein prominentes Beispiel von Google. Google AdSense [13] schaltet Anzeigen die das Thema der Webseite berücksichtigen.

### 2.2.2 Tracking

Ein zentraler Mechanismus im Affiliate Marketing ist das *Tracking*. Häufige Tracking-Methoden sind zum Beispiel das *URL-Tracking* und das *Cookie-Tracking*. Alle Tracking-Methoden verfolgen die selben Ziele [18]. Erstens das Identifizieren von Transaktionen eines Webseiten-Besuchers, beispielsweise einen Klick auf ein Werbemittel, oder den Erwerb eines Produktes. Und die Zuordnung des Webseiten-Besuchers und dessen Transaktion zu einem Publisher.

Im Folgenden werden als Beispiele für häufige Tracking-Technologien, das URL-Tracking und das Cookie-Tracking kurz erläutert.

**URL-Tracking** ist die einfachste Art des Trackings. Gelangt ein Besucher über ein Werbemittel auf die Seite eines Advertisers, so wird in die URL der Weiterleitung eine Kennung (Publisher ID) eingefügt, die den Publisher eindeutig identifiziert. Ein Nachteil dieser Methode ist, dass Transaktionen die zu einem späteren Zeitpunkt durchgeführt

werden, einem Publisher nicht mehr zugeordnet werden können. Daher sind Tracking-Methoden entwickelt worden, die Transaktionen von Besuchern über einen längeren Zeitraum verfolgen.

**Cookie-Tracking** basiert auf *Cookies*. Cookies sind Informationen, die ein Web-Server beim Besuch einer Webseite an den Browser sendet. Der Browser speichert sie in Form einer Text-Datei auf dem Computer ab. Sie enthalten Informationen, um einen Webseitenbesucher über mehrere Seitenaufrufe zu identifizieren [35]. Analog dazu nutzen Werbenetzwerke diese Technologie, um spätere Transaktionen von Besuchern zu erfassen. Gelangt ein Besucher durch ein Werbemittel zu einem Advertiser, so wird ein Cookie angelegt, das die Publisher-ID enthält [18]. Führt der Besucher zu einem späteren Zeitpunkt eine Transaktion aus, so kann diese durch das Cookie dem Publisher zugeordnet werden.

## 2.3 Microsoft .NET Framework

Das Microsoft .NET Framework ist eine Sammlung von Technologien zum Erstellen von Software für verschiedene Microsoft Betriebssysteme und Technologien. Das .NET Framework besteht aus zwei Komponenten, der *common language runtime* (CLR) und der *.NET Framework class library* [30]. Die CLR ist die Laufzeitumgebung des .NET Framework. Ihre Hauptaufgabe ist das Ausführen von Programmen, die für das .NET Framework erstellt wurden. Darüber hinaus übernimmt sie Aufgaben wie die Speicher-, Ressourcen- und Threadverwaltung, Exception-Handling, Garbage-Collection und viele mehr. Sie enthält mehrere Compiler, sodass es möglich ist, Code auszuführen, welcher in verschiedenen Programmiersprachen geschrieben wurde. Genauer gesagt alle Programmiersprachen, die dem *Common Language Infrastructure* (CLI) - Standard [11] entsprechen, können durch die CLR interpretiert und ausgeführt werden [28]. Bekannte Vertreter sind Visual Basic (VB).NET, C# und J#. Dies hat den Vorteil, dass verschiedene CLR - kompatible Sprachen untereinander kompatibel sind. So kann beispielsweise eine Software-Komponente geschrieben in C# problemlos mit einer Komponente, geschrieben in VB.NET interagieren. Code der dem CLI-Standard entspricht, wird von der CLR in die *Microsoft intermediate language* (MSIL) übersetzt. Soll MSIL - Code ausgeführt werden, so übersetzt ihn die CLR erst während der Laufzeit eine Maschinensprache und führt ihn aus [29]. Diesen Vorgang nennt man *Just-in-Time (JIT) Compilation*. Abbildung 2.1 zeigt zusammengefasst, den Ausführungsablauf der CLR.

### 2.3.1 Microsoft Visual Studio

Visual Studio ist eine integrierte Entwicklung Umgebung von Microsoft. In der aktuellsten Version *Visual Studio 2013* lassen sich, von einer simplen Konsolenanwendung für Windows bis hin zur MVC Webanwendung verschiedenste Softwareprojekte realisieren. Dabei Unterstützt Visual Studio unter Anderen, Hochsprachen wie C#, VB.NET und C++. In dieser Arbeit wurde Visual Studio 2012 als integrierte Entwicklungsumgebung eingesetzt.

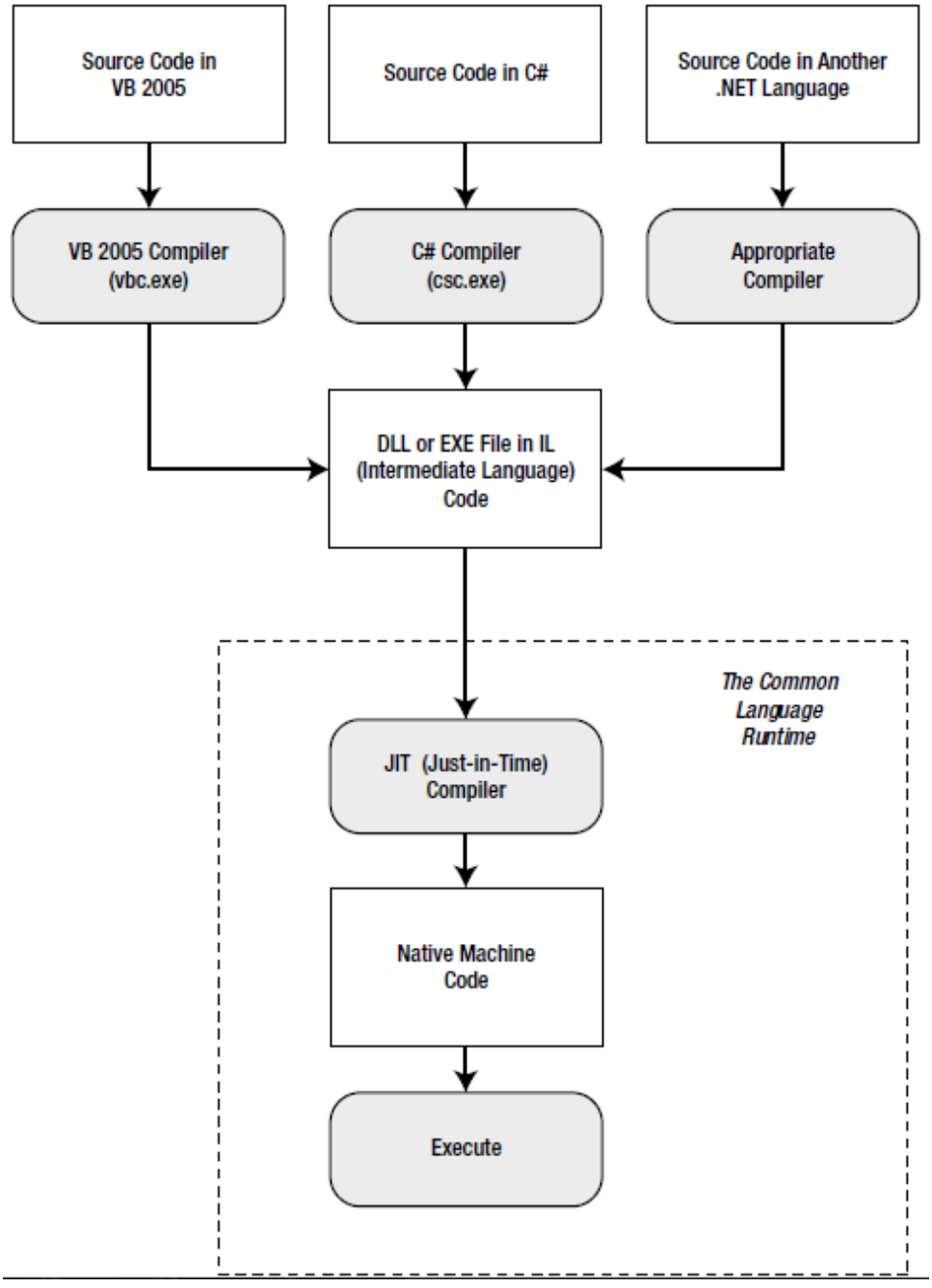


**Abbildung 2.1:** Ausführungsablauf der CLR [24]

### 2.3.2 C#

C# ist eine objektorientierte und standardisierte Programmiersprache von Microsoft [10]. Sie gehört zu den .NET-Programmiersprachen und ähnelt syntaktisch der Programmiersprache Java, siehe Listing 2.1. Es folgt stichwortartig ein kurzer Überblick an Konzepten, die in C# enthalten sind. An dieser Stelle sei der Leser aufgefordert, ihm unbekannte Konzepte zu recherchieren. Listing 2.1 zeigt das wohlbekannte Hello-World-Beispiel in C#.

**Konzepte in C#**

Folgende Konzepte sind in Java enthalten: *Objektorientierung, Typsicherheit, Garbage Collection, Namensräume, Threads, Generizität, Reflection, Attribute, Bibliotheken*

Folgende Konzepte sind nicht in Java enthalten: *Referenzparamter (call by reference), goto-Anweisung, Objekte im Keller, Delegates, Lambda-Ausdrücke, Query-Ausdrücke*

```
1  using System;
2  namespace HelloWorld
3  {
4      class Hello
5      {
6          static void Main()
7          {
8              Console.WriteLine("Hello World!");
9
10             // Keep the console window open in debug mode.
11             Console.WriteLine("Press any key to exit.");
12             Console.ReadKey();
13         }
14     }
15 }
```

**Listing 2.1:** Hello World in C#

#### LINQ

LINQ (Language intermediate Query) ist eine Technik, die es ermöglicht Datenstrukturen im Hauptspeicher mit SQL-artigen Abfragen zu verarbeiten. So können Arrays und Listen, anstatt mit Schleifen, mit *Query-Ausdrücken* durchsucht werden. Query-Ausdrücke erinnern an SQL-Abfragen und können auf allen Objekten angewendet werden, die das Interface `IEnumerable` implementieren.

Listing 2.2 durchsucht ein Array nach Städten, die mit dem Buchstaben **B** beginnen und Kopiert diese in eine Liste. Die Suche geschieht durch Iterieren und Vergleichen.

```
1  string[] cities = { "Bremen", "Berlin", "Bielefeld", "Oldenburg", "Osnabrück" };
2  List<string> CitiesStartWithB = new List<string>();
3  for (int i = 0; i < cities.Length; i++)
4  {
5       if (cities[i].StartsWith("B"))
6       {
7            CitiesStartWithB.Add(cities[i]);
8       }
9  }
```

**Listing 2.2:** Suche mittels Schleife

Listing 2.3 führt analog zum vorherigen Beispiel eine Suche nach Städten die mit dem Buchstaben **O** beginnen durch und kopiert diese in eine Liste. In diesem Beispiel wird jedoch ein Query-Ausdrück zum Suchen verwendet.

```
1  String[] cities = { "Bremen", "Berlin", "Bielefeld", "Oldenburg", "Osnabrück" };
2  List<string> CitiesStartWithO = (from city in cities where
3                                   city.StartsWith("O")
4                                   select city).ToList();
```

**Listing 2.3:** Suche mittels Query-Ausdruck

Auffällig ist die Ähnlichkeit zu SQL-Abfragen. Der Query-Ausdruck enthält aus SQL bekannte Schlüsselwörter wie *select*, *from* und *where*. Außerdem zeigen diese Beispiele, dass Query-Ausdrücke die Übersichtlichkeit von Code steigern können. So hat Listing 2.3 beispielsweise weniger Zeilen.

## 2.4 ASP .NET

Acitve Server Pages (ASP) .NET ist eine auf dem .NET Framework basierende Software Plattform von Microsoft, die es ermöglicht Webseiten, Webanwendungen und Webservices zu erstellen. ASP.NET bietet eine skalierbare Entwicklungsplattform, die das Erstellen von Webanwendungen erleichtert, indem sie oft genutzte Funktionen wie beispielsweise User-Authentifizierung oder Datenbankanbindung zur Verfügung stellt. Diese Eigenschaft ermöglicht das Erstellen von Webanwendungen auf einer höheren Abstraktionsebene und erspart und oft fehleranfällige Implementieren von Funktionen im Detail. Da ASP .NET zum .NET Framework gehört können Webanwendungen in allen CLR-kompatiblen Sprachen geschrieben werden. Gebräuchliche Sprachen sind beispielsweise C# oder VB.NET [24]. Technologien die zur ASP.NET-Plattform gehören und im Rahmen dieser Arbeit zum Einsatz kommen sind das MVC Framework, Web API und Signal R.

ASP .NET hat einen Martkanteil von 18% im Vergleich zu anderen serverseitigen Technologien. PHP beispielsweise, hat einen Marktanteil von 81.7% [32].

### 2.4.1 MVC Framework

Mirosoft MVC ist ein Framework zum Erstellen von Webanwendungen, das auf dem *Model View Controller-Entwurfsmuster* basiert. Der Vollständigkeit halber sei erwähnt, dass im Folgenden mit MVC das Microsoft MVC Framework gemeint ist und nicht das MVC-Entwurfsmuster. Dieser Abschnitt bezieht sich auf die Version 4 des MVC Frameworks.

**Exkurs: Das Model-View-Controller-Entwurfsmuster.**

*Model View Controller* unterteilt ein Softwaresystem in drei Komponenten: Model, View und Controller. Diese Strukturierung ermöglicht eine lose Kopplung der Komponenten, mit der Absicht sie möglichst unabhängig voneinander entwickeln, testen und warten zu können. Insgesamt soll unter Verwendung des MVC-Entwurfsmusters eine Komplexitätsreduktion, Wiederverwendbarkeit, Wartbarkeit und Flexibilität erreicht werden [31].

- Das *Model* enthält die Logik, die für die Datenverwaltung zuständig ist. Häufig ist das Model beispielsweise die Schnittstelle zu einer Datenbank und führt Datenbankabfragen aus.
- Die *View* dient zur Datendarstellung und Benutzeraktion. Typischerweise wird die View aus den Daten des Models erzeugt und "beobachtet"(Stichwort Observer-Entwurfsmuster) das Modell, sodass sie sich bei Änderung des Modells entsprechend anpasst.
- Der *Controller* ist das Bindeglied zwischen Model und View. Er reagiert auf Benutzerinteraktionen der View und veranlasst gegebenenfalls Änderungen am Model.

#### Model und Entity Framework

Es gibt verschiedene Möglichkeiten das Model zu gestalten. Grundsätzlich stellt sich erstmal die Frage nach einer Datenbank. Innerhalb des .NET Framework ist es möglich, sowohl relationale Datenbanken, als auch NoSql-Datenbanken zu nutzen. Ist die Wahl der Datenbank getroffen, stellt sich anschließend die Frage nach einem *Database Management System* (DBMS). Die folgenden Beispiele basieren auf dem DBMS SQL Server 2008 und einer relationalen Datenbank.

ADO.NET ist eine Sammlung von Klassen im Rahmen des .NET Framework, die die Kommunikation mit unterschiedlichen Datenquellen ermöglichen, darunter beispielsweise relationale Datenbanken oder XML-Dateien. Es gibt verschiedene Möglichkeiten, um

über ADO.NET mit relationalen Datenbanken zu kommunizieren. Eine Möglichkeit ist es, mittels in ADO.NET enthaltenen Klassen SQL-Abfragen direkt auf der Datenbank auszuführen. Eine weitere Möglichkeit ist *LINQ to SQL*, eine Technologie, die das Datenmodell einer relationalen Datenbank, einem Objektmodell zuordnet. Auf dem Objektmodell ausgeführte LINQ-Abfragen werden in SQL übersetzt und auf der Datenbank ausgeführt [27].

Das *Entity Framework* (EF) ist eine weitere Technologie, die die Kommunikation mit relationalen Datenbanken ermöglicht. Das EF ist ein *object-relational-mapping Framework* und bietet eine abstrakte Sicht auf die Datenbank. Es ermöglicht Objekte, die in einer objektorientierten Programmiersprache geschrieben sind, in eine Datenbank zu speichern und umgekehrt das Erzeugen von Objekten aus einer Datenbank. Das EF wurde in CAMC als Schnittstelle zur Datenbank verwendet und wird daher näher erklärt.

Das EF basiert auf einem konzeptuellen Datenmodell, dem *Entity Data Model* (EDM). Es gibt drei Vorgehensweisen um dieses zu erzeugen [19]:

- *database first*, beschreibt den Ansatz, das EDM mittels *Reverse Engineering* aus einer bestehenden Datenbank zu erzeugen.
- *model first*, beschreibt den Ansatz, das EDM von Grund auf selbst zu entwerfen. Dazu stellt Microsoft den *visual EDM Designer* zur Verfügung.
- *code first*, beschreibt den Ansatz, das EDM aus Klassen erzeugen zu lassen.

Der *code-first-Ansatz* findet in dieser Arbeit Verwendung und wird daher näher erklärt. Folgende Klassen (Listing 2.4) sollen mittels EF in eine Datenbank abgebildet werden.

```
1  namespace CodeFirstBeispiel
2  {
3
4      public class Professor
5      {
6          public int Id { get; set; }
7          public string Name { get; set; }
8          public virtual List<Student> Studenten { get; set; }
9      }
10
11     public class Student
12     {
13         public int Id { get; set; }
14         public string Name { get; set; }
15         public int Fachsemester { get; set; }
16
17     }
18
19 }
```

**Listing 2.4:** Model-Klassen

Listing 2.4 zeigt die Klassen `Professor` und `Student`. Es besteht eine Relation zwischen Professor und Student, sodass einem Professor mehrere Studenten zugeordnet sind. Diese Klassen werden inklusive Relation als Entitäten in eine Datenbank abgebildet. Die Schnittstelle zwischen Code und Datenbank bildet die Klasse `System.Data.Entity.DbContex`. In der Regel wird eine von `DbContext` abgeleitete Klasse angelegt, dessen Felder, auch *Properties* genannt, eine Collection der Objekte enthalten, die als Entitäten in einer Datenbank abgebildet werden, wie in Listing 2.5. Properties sind durch Getter- und Setter-Methoden erweiterte Instanzvariablen.

```
1  namespace CodeFirstBeispiel
2  {
3
4          public class UniContext : DbContext
5          {
6                  public DbSet<Professor> Professoren { get; set; }
7                  public DbSet<Student> Studenten { get; set; }
8          }
9
10 }
```

**Listing 2.5:** Von DbContext abgeleitete Klasse

Das Hinzufügen von Daten zur Datenbank geschieht nun über die Klasse `UniContext`, wie Listing 2.6 zeigt.

```
1  namespace CodeFirstBeispiel
2  {
3       public class Uni
4       {
5            static void Main(string[] args)
6            {
7                 //Studenten werden erzeugt
8                 Student alice = new Student();
9                 alice.Name =  "Alice";
10                alice.Fachsemester = 3;
11
12                Student willi = new Student();
13                willi.Name =  "Willi";
14                willi.Fachsemester = 4;
15
16                //Professor wird erzeugt
17                Professor bob = new Professor();
18                bob.Name = "bob";
19                bob.Studenten = new List<Student>();
20                //Zuordnen der Studenten zum Professor
21                bod.Studenten.Add(alice);
22                bod.Studenten.Add(willi);
23
24                UniContext db = new UniContext();
25                db.Professoren.Add(bob);
26                db.SaveChanges();
27           }
28      }
29 }
```

**Listing 2.6:** Hinzufügen von Objekten zur Datenbank

Datenbankabfragen werden auch über UniContext mittels LINQ ausgeführt. Listing 2.7 sucht nach Studenten des Professors Bob, dessen Name mit dem Buchstaben **W** beginnt.

```
1  namespace CodeFirstBeispiel
2  {
3       public class Uni
4       {
5            static void Main(string[] args)
6            {
7
8            UniContext db = new UniContext();
9
10           Professor Bob = (from prof in db.Professoren
11                            where prof.Name == "Bob"
12                            select prof).FirstOrDefault();
13
14           List<Student> StudentenStartWithW =
15                    (from stud in Bob.Studenten
16                    where stud.Name.StartsWith("W")
17                    select stud).ToList();
18           }
19      }
20 }
```

**Listing 2.7:** Suche in der Datenbank über das EF

Die Liste `StudentenStartWithW` enthält nur den Studenten Willi. Die Beispiele zeigen auch, dass das EF die Interaktionen mit dem DBMS übernimmt und man sich bei der Implementierung des Models keine Gedanken über das zugrundeliegende DBMS machen muss.

### View

Die View stellt die Benutzeroberfläche für den Benutzer der Webanwendung dar. In MVC 4 sind Views erweiterte HTML-Dateien. Erweitert, da es sich nicht um reinen HTML-Code handelt, sondern um ein HTML-Gerüst, erweitert mit Platzhaltern, welches serverseitige Code-Fragmente, geschrieben in C# oder VB.NET, enthält. Eine sogenannte *View Engine* sorgt dafür, dass die richtige View ausgewählt wird, eingebettete Code-Fragmente ausgeführt und die Platzhalter mit HTML-Code gefüllt werden. Die Standard View Engine in MVC 4 heißt *Razor*.

```
1  namespace CodeFirstBeispiel
2  {
3
4  @{
5       var items = new string[] {"Willi", "Alice", "Bob"};
6  }
7  <html>
8       <head><title>Beispiel dynamische Liste von Studenten.</title></head>
9       <body>
10           <h1>Es sind @items.Length Studenten registriert.</h1>
11           <ul>
```

```
12                  @foreach(var item in items) {
13                       <li>Name des Studenten: @item.</li>
14                  }
15                  /*
16                  Ausgabe:
17                  <li>Name des Studenten: Willi.</li>
18                  <li>Name des Studenten: Alice.</li>
19                  <li>Name des Studenten: Bob.</li>
20                  */
21              </ul>
22          </body>
23  </html>
24
25  }
```

**Listing 2.8:** Erweiterte View

Listing 2.8 zeigt ein einfaches Beispiel für eine View, die mit C#-Code-Fragmenten erweitert ist. Code-Blöcke beginnen mit einem @. Diese Syntax gehört zur Razor-View-Engine. Razor parst die Code-Blöcke und erzeugt HTML-Code. In diesem Fall wird ein Array mit Namen erzeugt und anschließend mittels einer Schleife in HTML ausgegeben.

### Controller/ Routing

In MVC sind Controller für die Bearbeitung von HTTP-Request zuständig. Dabei greifen sie gegebenenfalls auf das Model zu, bearbeiten dieses und geben dem Browser eine View zurück. Listing 2.9 zeigt einen erweiterten Standard-Controller der beim Anlegen einer MVC 4 Webanwendung in Visual Studio 2012 erzeugt wird. Controller in MVC werden von der Klasse `System.Web.Mvc.Controller` abgeleitet. Sie enthalten bestimmte Methoden, sogenannte *Actions*, in denen die Controller-Logik enthalten ist. Das *Routing* bestimmt welcher Controller und welche Action innerhalb des Controllers aufgerufen wird.

```
1   namespace BA.Controllers
2   {
3       public class HomeController : Controller
4       {
5           //Action: Index
6           public ActionResult Index(string id)
7           {
8               if (id == "de")
9               {
10                  ViewBag.Message = "Dein erste ASP.NET MVC Anwendung!";
11              }
12              else
13              {
14                  ViewBag.Message = "Your first ASP.NET MVC application!";
15              }
16              return View();
17          }
18
19          public ActionResult About()
20          {
```

```
21              ViewBag.Message = "Your app description page.";
22              return View();
23          }
24
25          public ActionResult Contact()
26          {
27              ViewBag.Message = "Your contact page.";
28              return View();
29          }
30      }
31  }
```

**Listing 2.9:** HomeController

Das Routing in MVC sorgt dafür, dass eine URL einer Controller-Action zugeordnet wird. So bewirkt das Aufrufen der URL

`{DOMAIN}/home/index`

die Ausführung der Action `Index` im `HomeController` aus Listing 2.9. Es können auch Parameter an eine Action übergeben werden. So bewirkt folgende URL,

`{DOMAIN}/home/index/de`

dass das Feld `ViewBag.Message` deutschen, anstatt englischen Text zugewiesen bekommt. In diesem Fall wird der GET-Parameter *de* an die Action HomeController übergeben.

Wird eine Action eines Controllers aufgerufen, so gibt es verschiedene Möglichkeiten Parameter an die Action zu übergeben. Auch gibt es verschiedene Möglichkeiten an Rückgabewerten. Eine Möglichkeit Parameter an eine Action zu übergeben ist die Übergabe der Parameter über die URL (HTTP-Get-Methode), siehe oben. Eine weitere Möglichkeit ist die Parameter über die HTTP-Post-Methode zu übergeben, beispielsweise durch die Übermittlung einer HTML-From. Eine Technologie die das Arbeiten mit Parametern in Actions erleichtert ist das *Model-Binding*.

Das Model-Binding übernimmt das Parsen und zuordnen von HTTP-Request-Parametern. Listing 2.10 zeigt die Klasse `Produkt`. Diese Klasse ist ein Model, das nicht im Rahmen des Entity Framework zum Einsatz kommt.

```
1  namespace BA.Models.ViewModels
2  {
3      public class Produkt
4      {
5          public string Name { get; set; }
6          public string Beschreibung { get; set; }
7          public int Anzahl { get; set; }
8      }
9  }
```

**Listing 2.10:** ViewModel Produkt

Ein Produkt kann als Parameter einer Action übergeben werden. Das Model-Binding sorgt dafür, dass die Felder des Models den HTTP-Request-Parametern zugeordnet werden. Die Form in Listing 2.11 enthält die drei Felder Name, Beschreibung und Anzahl. Der Name der Input-Felder muss genau den Namen der Felder in der Klasse Produkt entsprechen.

```
<form action="/Home/NeuesProdukt" method="post" novalidate="novalidate">
    <fieldset>
        <legend>Produkt</legend>

        <div class="editor-label">
            <label for="Name">Name</label>
        </div>
        <div class="editor-field">
            <input id="Name" name="Name" type="text" value="">
        </div>
        <div class="editor-label">
            <label for="Beschreibung">Beschreibung</label>
        </div>
        <div class="editor-field">
            <input id="Beschreibung" name="Beschreibung" type="text" value="">
        </div>
        <div class="editor-label">
            <label for="Anzahl">Anzahl</label>
        </div>
        <div class="editor-field">
            <input id="Anzahl" name="Anzahl" type="number" value="">
        </div>
        <p>
            <input type="submit" value="Senden">
        </p>
    </fieldset>
</form>
```

**Listing 2.11:** HTML-Form

Wird diese Form an die Action *NeuesProdukt* (Listing 2.12) übermittelt, so sorgt das Modelbinding dafür, dass ein Objekt der Produkt-Klasse erstellt wird und dessen Felder, den Feldern der Form zugeordnet werden. Das erspart das Parsen des HTTP-Requests. Anschließend kann auf das Produkt zugreifen und es beispielsweise in einer Datenbank ablegen, oder dem Benutzer eine Nachricht zurückgeben, wie in Listing 2.12.

```
namespace BA.Controllers
{
    public class HomeController : Controller
    {

        //...

        public ActionResult NeuesProdukt(Produkt Produkt)
        {
            //Tue etwas mit dem Produkt.
            //Lege es beispielsweise in einer Datenbank ab,
            //oder greife auf Felder des Produktes zu...
            ViewBag.Titel = "Sie haben das Produkt "+ Produkt.Name +" erfolgreich angelegt.";
            return View();
```

```
15          }
16
17          //...
18
19      }
20  }
```

**Listing 2.12:** Model als Parameter

Rückgabetypen einer Action sind beliebig und können beispielsweise einfache Datentypen wie string, oder boolean sein. Häufig sind es jedoch Objekte, die von der Klasse `ActionResult` abgeleitet sind und den Controller veranlassen HTML-Code zurückzugeben. In Listing 2.12 wird durch den Aufruf der View-Methode ein ViewResult-Objekt zurückgegeben. Die Klasse `ViewResult` ist von `ActionResult` abgeleitet und enthält Funktionalität, die das Zurückgeben einer View an den Browser ermöglicht. Dabei folgt die Auswahl der zurückzugebenden View meistens einer Konvention: Standardmäßig haben Views den Namen der Action, in der sie zurückgegeben werden und befinden sich in Ordern die den Namen des Controllers tragen.

Bei der Rückgabe eines ViewResults wie in Listing 2.12 durchsucht Razor die Ordner

`Views/Home/`

und

`Views/Shared/`

nach Views. Der Ordner Shared enthält Views auf die mehrere Controller zugreifen. Je nachdem, mit welcher Programmiersprache die View erweitert wurde, C# oder VB.NET, haben Views Endungen wie *.cshtml oder *.vbhtml.

### 2.4.2 SignalR

ASP.NET SignalR ist eine Library für ASP.NET Entwickler, die es ermöglicht eine bidirektionale Verbindung zwischen Server und Klient aufzubauen. Dazu nutzt SignalR die HTML5 WebSockets API [4]. HTTP lässt zwischen Server und Client nur eine unidirektionale Verbindung zu. Der Client stellt Anfragen und der Server antwortet. HTML5 WebSockets hingegen, basieren auf dem *Transmission Control Protocol* (TCP), welches einen bidirektionalen Datentransport ermöglicht [37]. In Chat-Anwendungen beispielsweise, müssen Chat-Fenster über neue Nachrichten informiert werden. Dies geschieht häufig über *Polling*, indem der Client in bestimmten Zeitabständen, den Server nach neuen Nachrichten fragt. SignalR macht Polling überflüssig, da es dem Server ermöglicht, Nachrichten in Echtzeit an Chat-Fenster zu senden. Für weitere Informationen sei auf die Webseite von SignalR verwiesen [4].

### 2.4.3 ASP.NET Web API

ASP.NET Web API ist ein Framework zum Erstellen von *Webservices*. Zusammengefasst liegt der Unterschied zwischen *Webservices* und *Webanwendungen* darin, dass Webanwendungen hauptsächlich mit Menschen, beispielsweise Webseitenbesuchern interagieren. Webservices tun dies hingegen mit Software, wie zum Beispiel anderen Webservices. Darin liegt auch der Kern des Unterschiedes zwischen ASP.NET MVC und ASP.NET Web API. MVC Webanwendungen empfangen Daten wie, beispielsweise Felder einer HTML-Form und geben HTML-Code zurück, der einem Menschen eine visuelle Oberfläche bietet. Web-API-Webservices hingegen empfangen strukturierte Daten wie XML oder JSON, verarbeiten diese und geben wiederum strukturierte Daten zurück, sodass eine andere Anwendung sie verarbeiten kann [12].

Das Web-API-Framework bietet viele Funktionen, die das Erstellen von Webservices, welche auf dem *Representational State Transfer-* (REST) basieren, zu vereinfachen.

**Exkurs: Representational State Transfer (REST)**

REST ist ein Entwurfsmuster für Webservices, das auf einem zustandslosen Client-Server-Protokoll, meistens HTTP, basiert. Das von Roy Fielding entwickelte *REST Maturity Model* ermöglicht anhand folgender Konzepte eine Klassifizierung eines Webservices, bezüglich seiner Eigenschaft REST-ful zu sein [17]:

- Eindeutige Uniform Resource Identifier (URI) für Ressourcen.
- Nutzung von HTTP-Verben.
- Hypermedia as the engine of application state (HATEOAS)

Auf diesen Konzepten baut das REST-Entwurfsmuster, angelegt an HTTP, auf. In einem REST-ful Webservice wird jede Ressource über eine eindeutige URI angesprochen. Ressourcen werden mit den HTTP-Methoden *POST*, *GET*, *PUT*, *DELETE*, angelegt, abgefragt, geändert und gelöscht. Dabei muss die Methode GET *sicher* sein, das heißt, sie darf keine Seiteneffekte bei ihrem Aufruf auslösen. Die restlichen Methoden müssen *idempotent* sein, was bedeutet, dass ein mehrfaches Aufrufen der Methoden die selbe Auswirkung, wie ein einzelner Aufruf hat. Eine Anfrage enthält Informationen zum gewünschten Rückgabetyp. Jeder Aufruf einer REST-Ressource ist in sich geschlossen und unabhängig. Das heißt jede Nachricht die an einen REST-ful Webservice abgesendet wird, enthält genug Informationen, damit sie der Server versteht. Frühere Aufrufe haben keine Auswirkungen auf aktuelle Aufrufe und es werden keine Sitzungsinformationen gespeichert [36]. Diese Eigenschaft nennt man *Zustandslosigkeit*. Eine Anwendung die mit einem REST-ful Webservice kommuniziert, soll keine Information

brauchen, um durch die Ressourcen zu navigieren. Das HATEOAS-Entwurfsprinzip sieht vor, Navigationsinformationen in die Rückgaben eines REST-Aufrufs zu integrieren. Mit diesen Informationen, beispielsweise in Form von URIs, kann der Client durch den REST-ful Webservice navigieren [36].

Eine dieser Funktionen ist das Mapping von Controller-Actions, auf die oben genannten HTTP-Methoden. Listing 2.13 zeigt den Standard-Controller, der beim Anlegen eines Web API Projektes in Visual Studio 2012 erstellt wird. Auf den ersten Blick fällt auf, dass die Namen der Actions den Namen der HTTP-Methoden entsprechen. Dieses hat den Grund, dass die Routing-Mechanismen in Web API versuchen, HTTP-Requests entsprechend ihrer Aufruf-Methode, Actions zuzuordnen, um diese aufzurufen.

```
1  namespace WebApiTest.Controllers
2  {
3      public class ValuesController : ApiController
4      {
5          // GET api/values
6          public IEnumerable<string> Get()
7          {
8              return new string[] { "value1", "value2" };
9          }
10
11         // GET api/values/5
12         public string Get(int id)
13         {
14             return "value";
15         }
16
17         // POST api/values
18         public void Post([FromBody]string value)
19         {
20         }
21
22         // PUT api/values/5
23         public void Put(int id, [FromBody]string value)
24         {
25         }
26
27         // DELETE api/values/5
28         public void Delete(int id)
29         {
30         }
31     }
32 }
```

**Listing 2.13:** WebApi Standard-Controller

Beispielsweise liefert ein HTTP-GET-Request an

`api/values/5`

den Wert *value* zurück. Ein HTTP-POST-Request an

`api/values/5`,

mit leerem HTTP-Body macht jedoch keinen Sinn. Da die Action *Post* nach einem Inhalt im HTTP-Request verlangt. In den Actions können empfangene Daten verarbeitet und in einer Datenbank gespeichert werden. Dazu kann wie in MVC beispielsweise das Entity Framework und ein modellbasierter Ansatz genutzt werden. Weiter ist auffällig, dass Actions oft keinen Rückgabewert haben und wenn doch, dann ist dies kein HTML wie in MVC, sondern Datenstrukturen wie JSON, XML. Es gibt noch weitaus mehr Unterschiede und Gemeinsamkeiten zwischen Web API und MVC, die an dieser Stelle nicht aufgeführt werden, da sie für das Verständnis dieser Arbeit nicht bedeutend sind.

# Kapitel 3

# Systementwurf- und Umsetzung

Im letzten Kapitel wurden Grundlagen und Technologien vorgestellt, auf denen diese Arbeit basiert. Dieses Kapitel widmet sich nun der Umsetzung des Projektes. Dabei werden nicht alle Einzelheiten des Projektes behandelt, sondern ausgesuchte, grundlegende und besonders ausgewählte Details.

## 3.1 Anforderungsanalyse

Der Titel dieser Arbeit enthält den Ausschnitt *prototypische Implementierung*. Dass heißt, die Software die im Rahmen dieser Arbeit entsteht (CAMC), muss nicht den Anforderungen einer ausgereiften Software entsprechen, die der breiten Masse zugänglich gemacht wird. CAMC ist ein Prototyp der relativ schnell Feedback zur Machbarkeit und Eignung der Umsetzung einer Affilate-Marketing-Komponente in einen Live Chat liefern soll. Daher enthält diese Anforderungsanalyse ausschließlich funktionale Anforderungen.

CAMC ist eine Live Chat-Anwendung und soll funktionell, typische Live Chat Anforderungen erfüllen. Auf höchster Ebene soll CAMC einen Live Chat zwischen Klient und Agent ermöglichen. Dazu sollen sich Webseitenbetreiber und zugehörige Agenten bei CAMC registrieren können. Sowohl Agenten, als auch Webseitenbetreibern soll ein interner Bereich zur Verfügung stehen. Der Zugang für Webseitenbetreiber wird im folgenden *CAMC-Account* genannt, der Agenten-Zugang, *Agent-Account*. Webseitenbetreiber sollen über ihren CAMC-Account, Agenten und Affiliate Systeme verwalten können. Agenten über ihren Agent-Account, Chats verwalten und die Agenten-Konsole aufrufen.

Abbildung 3.1 verdeutlicht Akteure und deren Beziehungen im CAMC-Kontext. Einem Webseitenbetreiber sind beliebig viele Agenten zugeordnet. Ein Agent kann sich mit beliebig vielen Klienten im Live Chat befinden. Webseitenbetreiber und Agenten sind

bei CAMC registriert. Der Webseitenbetreiber ist zusätzlich bei einem, oder mehreren Affiliate-Systemen als Publisher registriert. Schließlich greift CAMC auf Affiliate-Systeme zu, um Werbemittel bzw. Produkte zu erlangen.

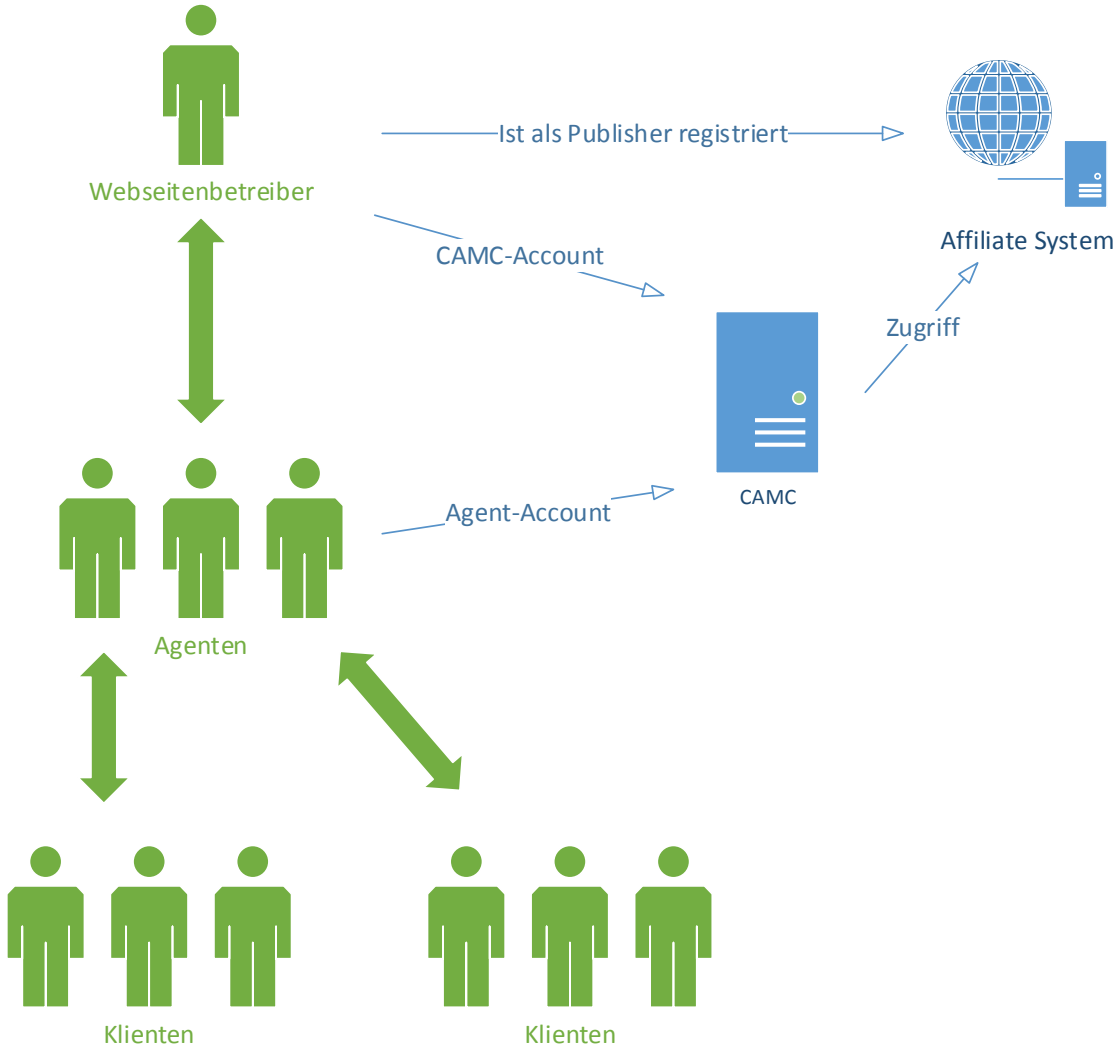


**Abbildung 3.1:** Akteure und deren Beziehungen

Klienten sollen durch Links oder Banner zum Chat aufgefordert werden können. Dazu muss CAMC von Haus aus eine Agenten-Konsole, sowie einen Klient-Chat bereitstellen. Darüber hinaus soll CAMC eine Schnittstelle (API) enthalten, die es ermöglicht eigene Klient-Chats und Agenten-Konsolen zu gestalten. Zum Beispiel eine mobilen Chat-Klienten für Smartphones. Die Agenten-Konsole soll möglichst viele Informationen über

den Klienten anzeigen. Der Klient-Chat kann dagegen ein einfaches Chatfenster sein.

Eine Affiliate-Marketing-Komponente soll die Integration von Affiliate Systemen ermöglichen. Das heißt, CAMC greift auf die Schnittstellen von Affiliate-Systemen zu, um Werbemittel von Produkten zu erlangen. Werbemittel sind in diesem Fall einfache Links zu Produkten. Diese Links sollen einem Agenten in der Agenten-Konsole angezeigt werden und stehen idealerweise in einem Zusammenhang zum Chat. Handelt der Chat beispielsweise vom Microsoft MVC 4 Framework und hat der Klient fragen dazu, so sollen dem Agenten passende Produkte angezeigt werden, beispielsweise Bücher zum MVC 4 Framwork. Außerdem soll der Agent die Möglichkeit haben nach Produkten zu suchen und Links in die Unterhaltung einzufügen.

## 3.2 Vorgehen und GUI Prototyping

In dieser Arbeit soll ein Prototyp einer Live Chat Anwendung implementiert werden. Unter den Prototyping-Vorgehensmodellen entspricht das *experimentelle Prototyping* am ehesten, dem Vorgehen in diesem Projekt. Denn das Ziel ist es, mittels eines Prototypen Rückmeldung zu erhalten und Erfahrungen zu sammeln, ob und wie sich Affilliate-Marketing in Live Chats integrieren lässt.

Insgesamt lässt sich diese Arbeit in zwei Phasen einteilen. In der ersten Phase wurden bestehende Live Chats und Affiliate-Marketing-Technologien recherchiert und untersucht. Das Ziel dieser Phase war es einerseits essentielle Funktionen eines Live Chats zu extrahieren, die später in CAMC übernommen werden sollten. Andererseits nach Möglichkeiten zu suchen, um Werbemittel in einen Live Chat einzubinden. Das Resultat waren GUI-Protoypen im Sinne von visuellen Designentwürfen, wie beispielsweise Abbildung 3.2.

Abbildung 3.2 zeigt den internen Bereich für Webseitenbetreiber. Die Tabs enthalten folgende Funtionen:

- Home: Informationen zu Chats und Agenten.
- Agente: Liste an Agenten die zum Webseitenbetreiber gehören. Funktionen um Agenten zu bearbeiten.
- Applications: Liste an Anwendungen, die die API nutzen. Erzeugung von API-Schlüsseln.
- Affiliates: Liste an Unterstützten Affiliate Systemen. Dort kann der Webseitenbetreiber seine Zugangsdaten zu Werbenetzwerken oder Affiliate-Programmen hinterlassen.

Abbildung 3.3 zeigt den internen Bereich für Agenten. Der Home-Tab enthält allgemeine

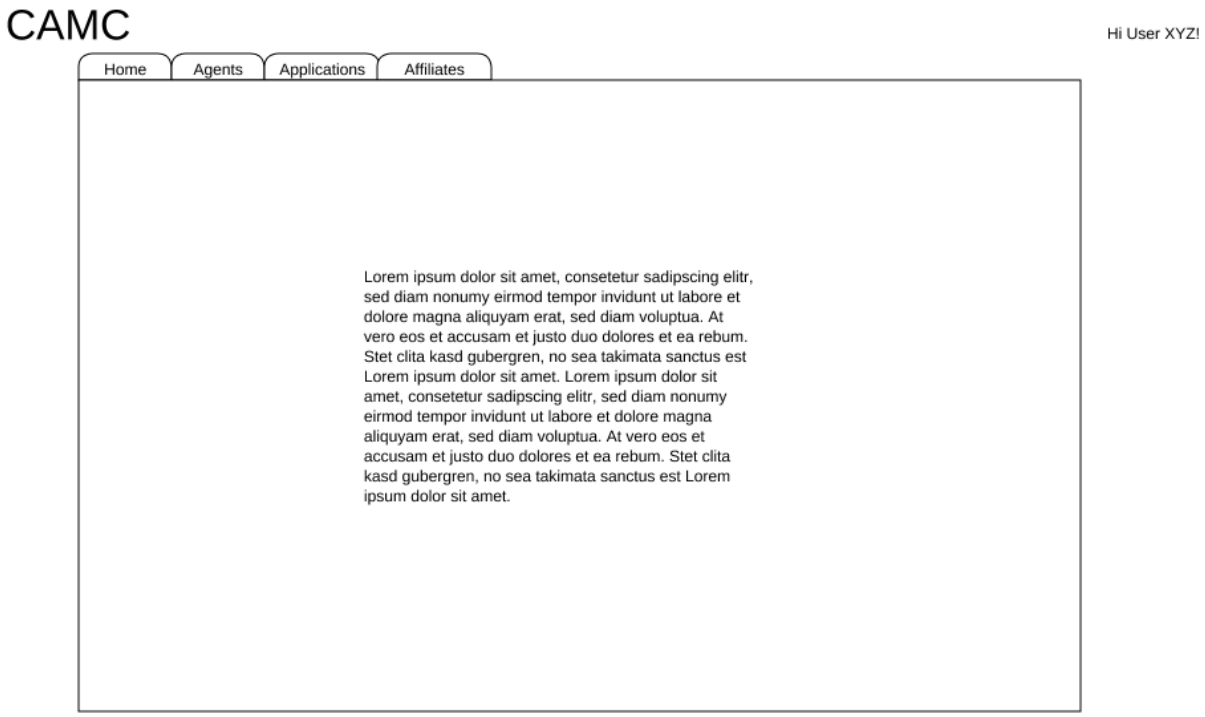


**Abbildung 3.2:** GUI-Prototyp: Interner Bereich für Webseitenbetreiber

Informationen zu Chats. Der LiveChats-Tab zeigt eine List an aktuellen Live Chats. Dort kann der Agent zu der Agenten-Konsole gelangen.

Abbildung 3.4 ist der bedeutendste GUI-Prototyp, da dort wichtige Konzepte dieser Arbeit abgebildet werden, wie beispielsweise die Affiliate-Marketing-Komponente. Die Linke Seite enthält eine Übersicht an Produkten aus den unterstützten Affiliate Systemen. Der Agent hat die Möglichkeit diese Produkte in die Unterhaltung einzubinden. Des Weiteren gibt es eine Möglichkeit Produkte zu suchen. Schließlich enthält die rechte Seite Informationen zum Klienten und das Chatfenster.

Viele Details stellten sich jedoch erst im Laufe der Implementation heraus. So lässt sich beispielsweise im Entwurf der Agenten-Konsole nicht erkennen, was für Informationen über den Klienten angezeigt werden. Oder wie sich der Chat so auswerten lässt, dass passend zur Unterhaltung Produkte angezeigt werden können.

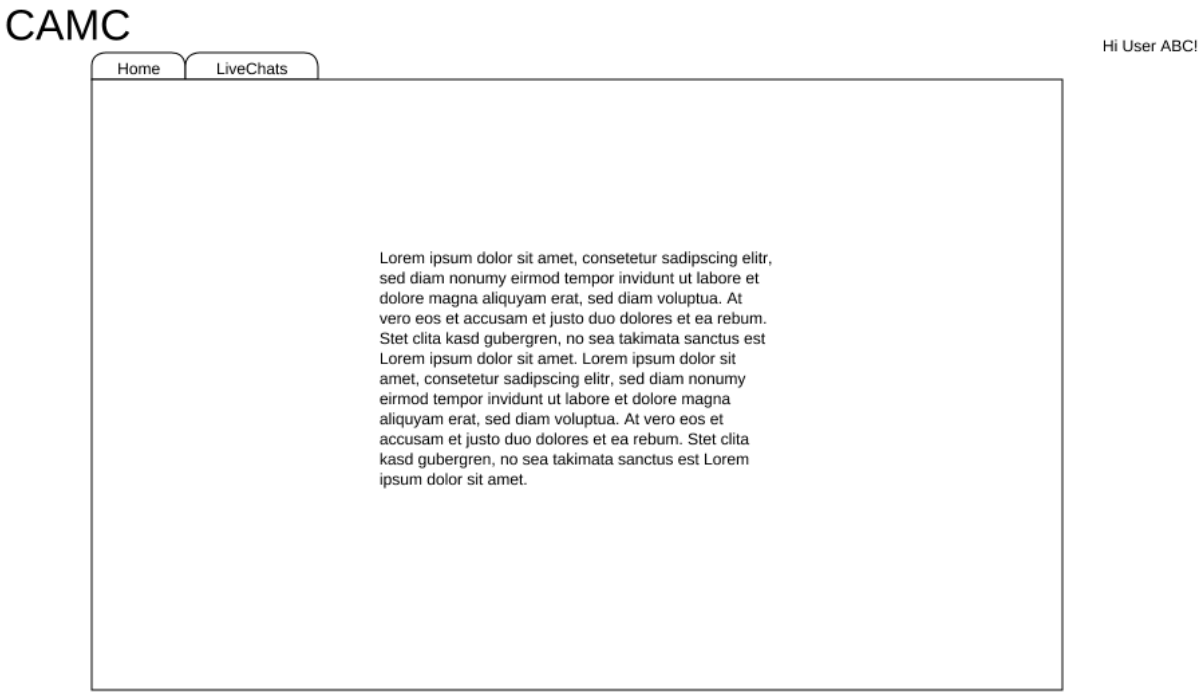


**Abbildung 3.3:** GUI-Prototyp: Interner Bereich für Agenten

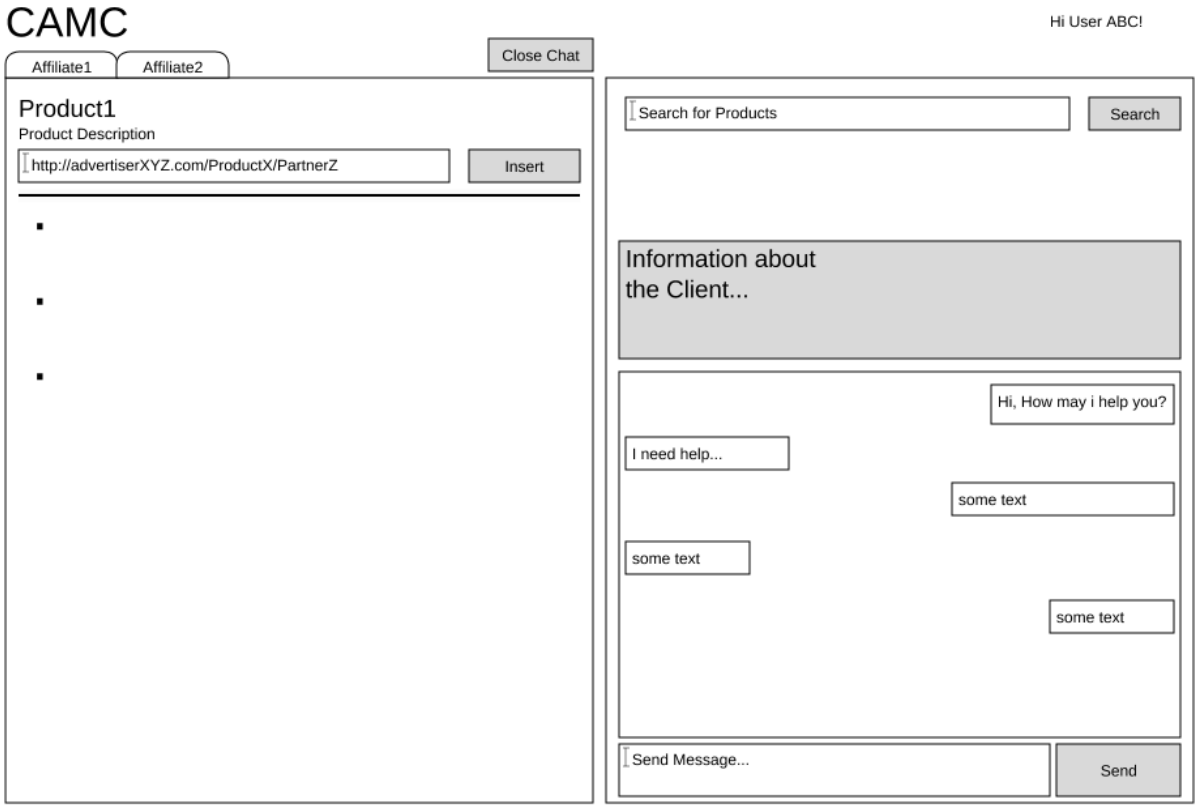


**Abbildung 3.4:** GUI-Prototyp: Agent-Konsole

## 3.3 Serverarchitektur

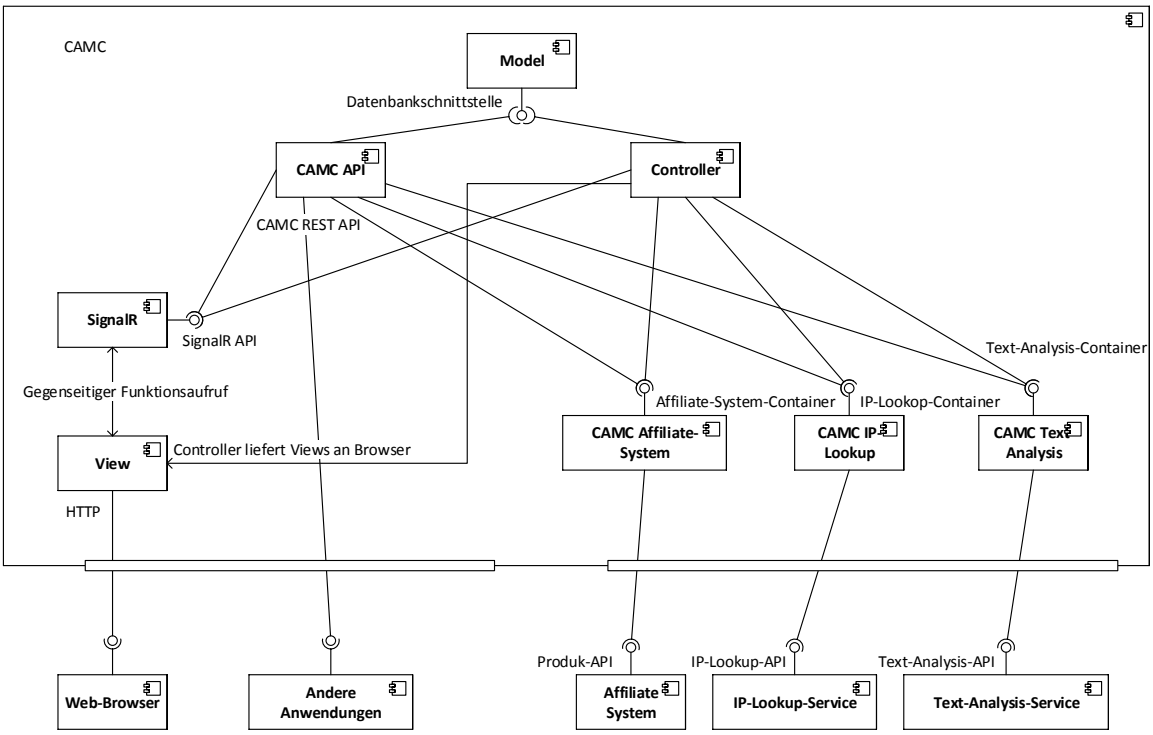


**Abbildung 3.5:** UML-Komponentendiagramm: Software-Komponenten in CAMC

Abbildung 3.5 zeigt ein UML-Komponentendiagramm, das die wichtigsten Softwarekomponenten in CAMC und deren Abhängigkeiten enthält. In diesem Abschnitt werden Model, View, Controller und die SignalR-Komponente erklärt. In den nächsten Abschnitten folgen die restlichen Komponenten.

Die oft erwähnte Affiliate-Marketing-Komponente ist nicht in dem Komponentendiagram von Abbildung 3.5 abgebildet, da es sich hierbei nicht um eine abgeschlossene Softwarekomponente handelt, sondern um ein Zusammenspiel von Funktionen aus den anderen Komponenten.

**Model** Die Datenbankanbindung in CAMC übernimmt das *Entity Framework*. Abbildung 3.6 zeigt ein *Entity Data Model* (EDM). Dieses Diagramm beschreibt das konzep-

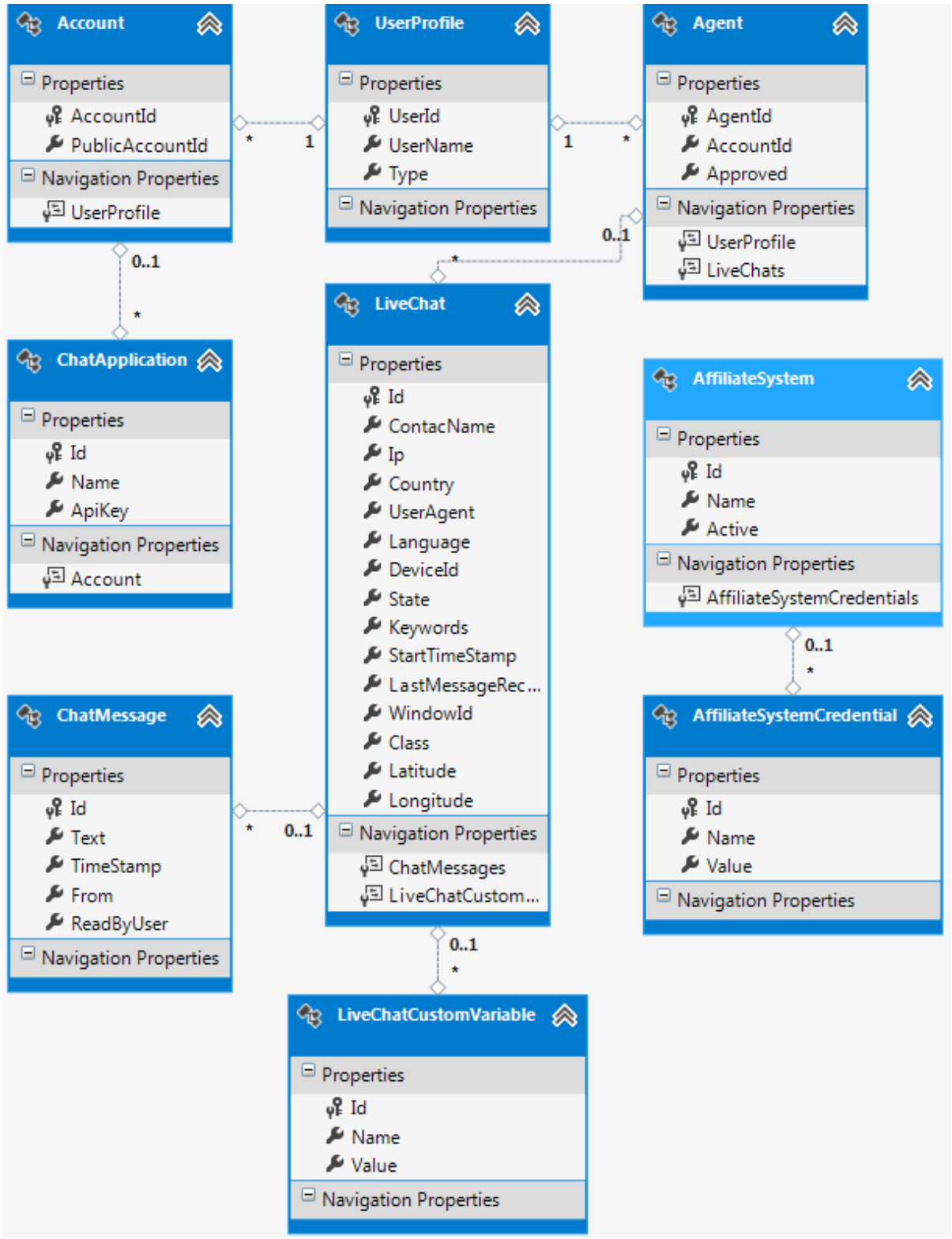


**Abbildung 3.6:** Entity Data Model in CAMC

tuelle Datenmodell in CAMC. Das EDM basiert auf dem *Entity-Relationship-Model* nach Peter Chen [34]. Das Vorgehen beim Erstellen des konzeptuellen Datenmodells folgt dem *Code-First-Ansatz*. Dabei wird das konzeptuelle Datenmodell durch Model-Klassen definiert, aus welchen das Entity Framework anschließend ein Datenbankschema generiert.

Der Zugriff auf die API des Entity Frameworks erfolgt über die Klasse `CamcDb.cs` (Listing 3.1). Diese Klasse ist von `System.Data.Entity.DbContex` abgeleitet, welche die Funktionalitäten für die Datenbankzugriffe enthält. Auffällig sind die Properties der Klasse `CamcDb`. Diese entsprechen Objekten, welche Entitäten aus dem EDM von Abbildung 3.6 enthalten. Die Abhängigkeiten von Entitäten werden durch Properties in den Model-Klassen abgebildet. So hat beispielsweise die Entität *Agent* ein Property *UserProfile*, welches einem Agenten genau ein UserProfile zuordnet.

```
1  namespace CAMC.Models
2  {
3      public class CamcDb : DbContext
4      {
5          public CamcDb()
6              : base("name=DefaultConnection")
7          {
8
9          }
10
11         public DbSet<LiveChat> LiveChats { get; set; }
12         public DbSet<ChatApplication> ChatApplications { get; set; }
13         public DbSet<ChatMessage> ChatMessages { get; set; }
14         public DbSet<LivChatCustomVariable> LiveChatCustomVariables { get; set; }
15         public DbSet<UserProfile> Users { get; set; }
16         public DbSet<Agent> Agents { get; set; }
17         public DbSet<Account> Accounts { get; set; }
18         public DbSet<AffiliateSystem> AffiliateSystems { get; set; }
19         public DbSet<AffiliateSystemCredential> AffiliateSystemCredentials { get; set; }
20     }
21 }
```

**Listing 3.1:** CamcDb.cs bildet die Schnittstelle zur Datenbank

**Views und SignalR** Views sind mit C# erweiterte HTML-Dateien, welche über die Controller an den Browser zurückgegeben werden. Eine Besonderheit an den Views in CAMC ist, dass mittels SignalR eine bidirektionale Verbindung zwischen Klient und Server aufgebaut wird. So ist es beispielsweise möglich, dass die Serveranwendung, in den Views definierte Javaskript-Funktionen aufruft. Anders herum sind auch die Views in der Lage mittels Javascript, Funktionen in CAMC aufzurufen. Mit dieser Technik ist der Live Chat in CAMC realisiert. Sendet ein Agent über die Agenten-Konsole eine Nachricht, wird diese mittels SignalR an CAMC gesendet. Anschließend wird die Nachricht, auch mittels SignalR, an den Klient-Chat gesendet und die entsprechende Javascript-Funktion aufgerufen. Listing 3.2 zeigt ein Code-Schnippsel aus der View der Agenten-Konsole.

```
1       //...
2           //Bei einem Klick auf den Senden-Button...
3           $('#send-message-btn').click(function () {
4               var text = $('#message-text').val();
5               //Sende die Nachricht mittels SignalR an CAMC
6               chat.server.sendMessage(text, WindowId, 1);
7               $('#message-text').val("");
8           });
9       //...
```

**Listing 3.2:** Ausschnitt aus der View der Agenten-Konsole

Dieses Schnippsel wird ausgeführt, wenn ein Agent eine Chat-Nachricht absendet. In Zeile 6 wird mittels SignalR auf dem Server die gleichnamige Funktion *sendMessage* aufgerufen. Diese Funktion verarbeitet die Nachricht und benachrichtigt den Klient-Chat.

```
1  //...
2  public void SendMessage(string message, int WindowId, int From)
3  {
4    using(CamcDb db = new CamcDb())
5    {
6      //...
7      if (db.SaveChanges() > 0)
8      {
9        context.Clients.Group(LiveChat.WindowId.ToString()).
10       newChatMessage(message, date, (MessageSender)From);
11      //...
12 }
13 //...
```

**Listing 3.3:** Die Funktion SendMessage wird durch eine View aufgerufen

Listing 3.3 zeigt einen Teil der Funktion `SendMessage`, die in Listing 3.2 aufgerufen wird. Nachdem die Nachricht verarbeitet wurde, werden in Zeile 9 und 10 alle betroffenen Chat-Fenster über die neue Nachricht informiert, indem die Javascript-Funktion `newChatMessage`, in den betroffenen Views aufgerufen wird.

**Controller** In CAMC sind mehrere Controller implementiert. So gibt es etwa für jeden Tab in den internen Bereichen einen eigenen Controller, der HTTP-Anfragen entgegennimmt. Auch für die Benutzer-Authentifizierung ist ein eigener Controller zuständig.

```
1  public class LiveChatController : Controller
2  {
3    private CamCDb _db = new CamCDb();
4    public ActionResult Index()
5    {
6      Agent curAgent = _db.Agents.Where(a => a.UserProfile.UserName ==
7         HttpContext.User.Identity.Name).FirstOrDefault();
8      List<LiveChat> LiveChats = curAgent.LiveChats.
9        .Where(ads => ads.State == ChatState.ACTIVE || ads.State == ChatState.ENDED)
10       .OrderByDescending(a => a.StartTimeStamp).ToList();
11
```

```
12      List<LiveChatViewModel> LiveChatVewModel = new List<LiveChatViewModel>();
13      foreach(LiveChat item in LiveChat)
14      {
15        LiveChatViewModel advm = new LiveChatViewModel
16        {
17          Country = item.Country,
18          Ip = item.Ip,
19          State = item.State,
20          LastMessageReceived = item.LastMessageReceived,
21          ChatSessId = item.Id,
22          WindowId = item.WindowId,
23          Class = item.Class,
24          ContactId = item.ContacName
25        };
26        LiveChatVewModel.Add(advm);
27      }
28      ViewBag.AgentId = curAgent.AgentId;
29      return PartialView("_Index", LiveChatViewModel);
30    }
31
32    protected override void Dispose(bool disposing)
33    {
34      _db.Dispose();
35      base.Dispose(disposing);
36    }
37  }
```

**Listing 3.4:** Live-Chat-Controller

Listing 3.4 zeigt beispielhaft den Controller des LiveChat-Tabs im internen Bereich eines Agenten. Wird der Tab aufgerufen sorgt das Routing dafür, dass die Action `Index` dieses Controllers ausgeführt wird. In der Action finden Controller-typische Vorgänge statt. Es werden Daten aus der Datenbank geholt. In diesem Fall handelt es sich um den eingeloggten Agenten und dessen Chats (Zeilen 6 bis 13). Diese Daten werden in ein ViewModel gepackt (Zeilen 15 bis 31) und die View mittels Razor mit dem ViewModel `LiveChatViewModel` erweitert. Schließlich wird die View an den Browser geschickt (Zeile 33). Analoges, geschieht auch in den meisten anderen Controllern.

## 3.4 Programmablauf und Datenverarbeitung

Das Abschicken einer Nachricht aus einem Klient-Chat löst in CAMC verschiedene Prozesse aus. So werden beispielsweise Informationen zum Absender gesammelt und die Nachricht semantisch analysiert. Anschließend werden gezielt Anfragen an ein Affiliate System gesendet um Links zu Produkten zu erhalten.

Abbildung 3.7 skizziert grob die Schritte, von dem ersten Eintreffen einer Nachricht in CAMC, beziehungsweise dem Start eines neuen Live Chats, bis zum Anzeigen von Produkten in der Agenten Konsole. Trifft eine Nachricht ein, so wird zuerst der Absender analysiert. Ein externer Dienst, ein sogenannter *IP-Lookup-Service*, soll anhand der IP-

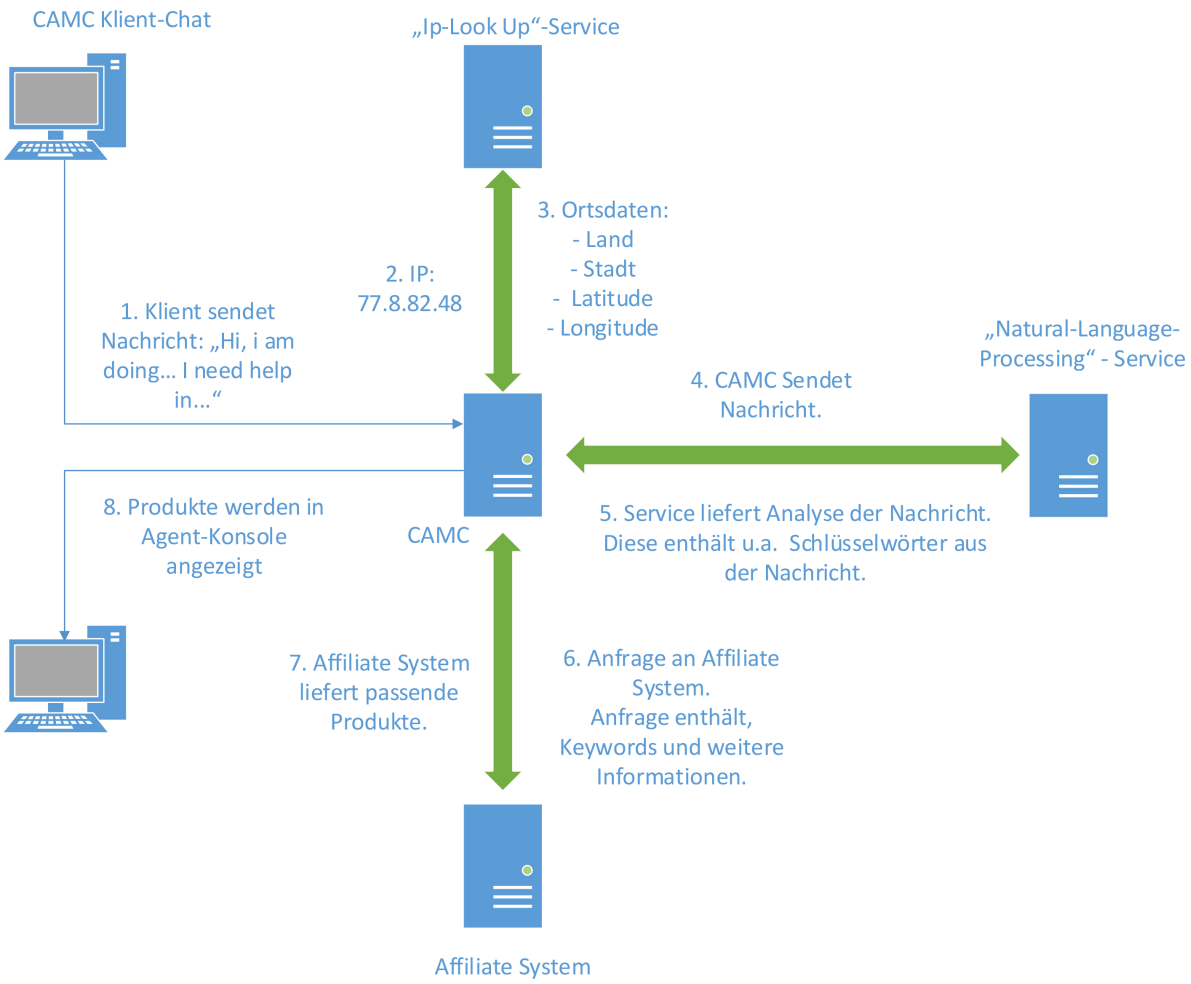


**Abbildung 3.7:** Verarbeitungsschritte einer Nachricht

Adresse (Schritt 2), Ortsdaten zum Absender liefern (Schritt 3). Diese Daten werden später dem Agenten angezeigt. Anschließend wird die Nachricht selbst analysiert. Auch diese Aufgabe übernimmt ein externer Dienst. Nachdem die Nachricht an einen *Natural-Language-Processing-Service* übermittelt wurde (Schritt 4), generiert dieser unter Anderem Schlüsselwörter, die den Inhalt der Nachricht beschreiben (Schritt 5). Diese Informationen werden an ein Affiliate System gesendet (Schritt 6), welches innerhalb seines Targeting, zur Nachricht passende Produkte liefert (Schritt 7). Schließlich werden Informationen zum Absender und Produkte von Affiliate Systemen, in der Agenten-Konsole angezeigt (Schritt 8). Details zu den einzelnen Prozessen folgen in den anschließenden Abschnitten.

### 3.4.1 Klienten-Profil

Dem Agenten soll es möglich sein ein genaues Klienten-Profil erstellen zu können, das ihm beispielsweise hilft den Klienten zu beraten und gezielt Produkte in den Chat einzubinden. Dazu sollen dem Agenten möglichst viele Informationen über einen Klienten angezeigt werden. Die Informationsbeschaffung kann über zwei Wege geschehen. Eine Möglichkeit ist es, dass ein Klient, bevor ein Chat beginnt, über ein Formular befragt wird. In dieser sogenannten *Pre Chat Survey* kann der Klient beispielsweise seinen Namen und seine E-Mail hinterlassen. Eine Pre Chat Survey ist in vielen Live Chats eine Bedingung, um eine Unterhaltung beginnen zu können. So auch in CAMC. In Live Chats, die über einen Link aufgerufen werden, also nicht über die Chat API zustande kommen, muss der Klient seinen Namen und seine E-Mail hinterlegen, um einen Live Chat beginnen zu können, siehe Abbildung 3.8. Optional kann der Klient, die Firma nennen für die er tätig ist, und den Namen eines Agenten angeben, mit dem er chatten möchte.

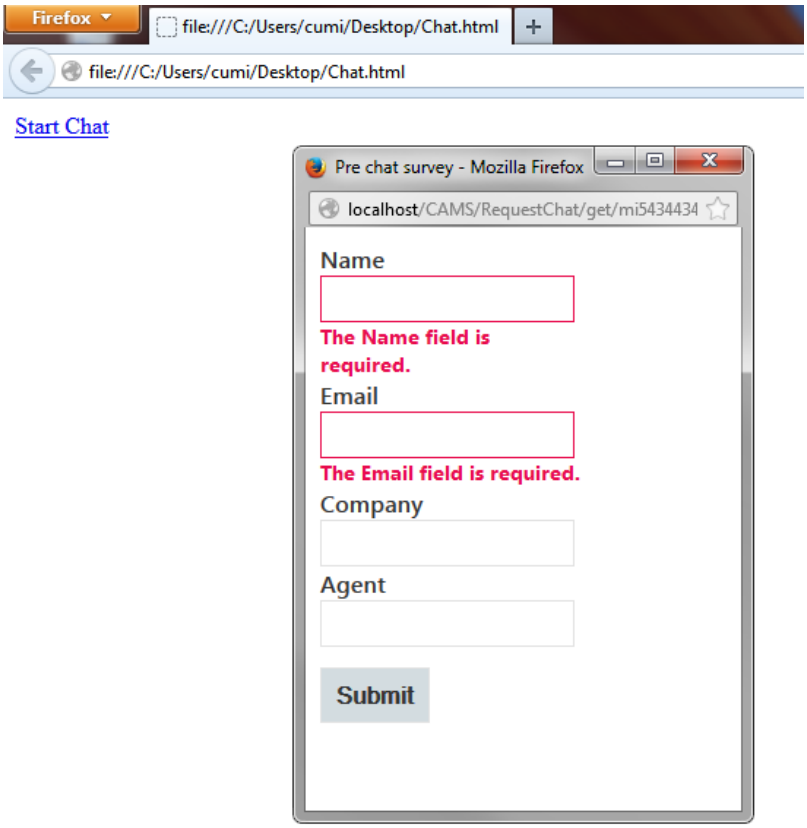


**Abbildung 3.8:** Pre Chat Survey im Klient-Chat

Eine weitere Möglichkeit ist die technische Informationsbeschaffung. Ein Live Chat wird mittels HTTP-Requests initialisiert, dass heißt, nimmt ein Klient eine Live-Chat-Aufforderung

an und klickt auf einen Link, so wird ein HTTP-Request an CAMC gesendet. Dieser HTTP-Request enthält einige Informationen über den Klienten. Er enthält zum Beispiel die IP-Adresse des Klienten und Informationen über den Browser, den der Klient benutzt. Besonders interessant ist jedoch die IP-Adresse, denn sie lässt Schlüsse über den Ort des Klienten zu. Sogenannte *Geo-Datenbanken* oder auch *IP-Lookup-Services* können bis auf die Postleitzahl genau den Ort des Klienten bestimmen und darüber hinaus auch Informationen zum Intertnet Service Provider (ISP) und zur Übertragungsgeschwindigkeit liefern. CAMC nutzt für diesen Zweck die Geo-Datenbank von MaxMind [26]. Dazu wird die IP-Adresse des Klienten an den GeoIp-Service von MaxMind geschickt. Dieser liefert Daten im CSV-Format, welche in CAMC verarbeitet und dem Agenten angezeigt werden. Oft sind solche IP-Lookup-Services jedoch ungenau und lassen eine genaue Ortsbestimmung nicht zu. Auch gibt es beispielsweise Möglichkeiten die IP-Adresse in einem HTTP-Request zu manipulieren (IP-Spoofing). Die MaxMind Geo-Datenbank hat im Laufe dieser Arbeit zuverlässige Standortdaten bezüglich der Stadt in der sich der Klient befindet, geliefert.

### 3.4.2 Chat-Analyse

Eine Aufgabe der Affiliate-Marketing-Komponente ist es, automatisch dem Agenten, zum Chat passende Produkte anzuzeigen. Dazu muss die Unterhaltung analysiert werden, Schlüsselwörter generiert und eine Anfrage, inklusive Schlüsselwörter, an ein Affiliate-System gesendet werden. Diese Prozedur soll dem Agenten das Suchen nach Produkten ersparen. Anstatt einen eigenen Algorithmus zu schreiben, übernimmt diese Aufgabe ein externer *Natural-Language-Processing-Service*. Nach längerer Suche hat sich [5] als passender Kandidat herausgestellt. Denn AlchemyAPI bietet im Gegensatz vergleichbaren Anbietern, die Analyse von reinem Text. Viele vergleichbare Anbieter konzentrieren sich auf die Analyse von Webseiten oder anderen Formaten. AlchemyAPI unterstützt je nach Funktion verschiedene Sprachen. Englisch jedoch wird von allen Funktionen unterstützt und dient hier als Ausgangssprache.

AlchemyAPI bietet mehrere Funktionen um Inhalte analysieren zu lassen. Folgende drei Funktionen wurden in CAMC verwendet, um Nachrichten zu analysieren:

- *Keyword Extraction API.* Es werden aus dem Text Schlüsselwörter extrahiert.
- *Text Categorization API.* Dem Text wird eine von zwölf vordefinierten Kategorien zugeordnet.
- *Concept Tagging API.* Es werden Konzepte im Text gesucht.

| **Schlüsselwort** | **Relevanz** | **Stimmung** |
|---|---|---|
| deep learning technology | 0.931797 | positive |
| Access Venture Partners | 0.908388 | neutral |
| artificial intelligence | 0.786308 | positive |
| web pages | 0.783459 | positive |
| text documents | 0.782035 | positive |
| new services | 0.750341 | positive |
| tweets | 0.629251 | positive |
| hires | 0.601697 | positive |
| emails | 0.588737 | positive |
| AlchemyAPI | 0.546467 | neutral |
| capabilities | 0.545792 | positive |
| forms | 0.543844 | neutral |
| content | 0.543694 | neutral |
| Series | 0.541011 | negative |
| company | 0.540584 | negative |
| sales | 0.540085 | negative |

**Tabelle 3.1:** Resultate der Keyword Extraction API

**Keyword Extraction API** Die Keyword Extraction API analysiert einen Text nach Schlüsselwörtern, die wichtige Themen im Text darstellen. Jedem Schlüsselwort wird ein *Relevanzfaktor* und eine *Stimmung* zugeordnet. Der Relevanzfaktor ist ein Wert zwischen 0 und 1 und beschreibt wie zutreffend das extrahierte Schlüsselwort ist. Ein Wert nahe null entspricht einem schwachen Schlüsselwort, das das Thema nur mangelhaft beschreibt. Ein Wert nahe 1 entspricht einem Schlüsselwort, das das Thema sehr gut beschreibt. Die Stimmung kann *postive*, *neutral*, oder *negative* sein.

Aus dem Text [38],

> AlchemyAPI has raised $2 million to extend the capabilities of its deep learning technology that applies artificial intelligence to read and understand web pages, text documents, emails, tweets, and other forms of content. Access Venture Partners led the Series A round, which the company will use to ramp up its sales and marketing, make hires and launch new services.

extrahiert die Keyword Extraction API folgende Schlüsselwörter:

**Text Categorization API** die Text Categoriaztion API ordnet einem Text eine von zwölf durch AlchemyAPI definierten Kategorien zu. Folgende Kategorien können zugeordnet werden: *arts and entertainment, business, computers and internet, culture and politics, gaming, health, law and crime, religion, recreation, science and technology, sports, and weather*. Auch hier wird der Kategorie ein Wert (Score) zwischen 0 und 1 zugeordnet, welcher aussagt wie zutreffend die Kategorie ist.

Dem Text [9],

> The Denver Broncos have signed middle linebacker Paris Lenon. He's expected to serve as veteran depth to a depleted defense. Lenon is a 12-year veteran who played for four teams. The 35-year-old has 809 tackles, 12 sacks, 10 forced fumbles and five interceptions in his NFL career. He most recently spent time with the Arizona Cardinals, starting 48 games in the past three seasons. He recorded 103 tackles, two sacks and one interception in the 2012 season.The Broncos are thin at linebacker after losing Stewart Bradley to a wrist injury.

ordnet die Text Categorization API die Kategorie *sport*, mit einem Score von *0.618912* zu.

**Concept Tagging API** Die Concept Tagging API generiert ähnlich wie die Keyword Extraction API Schlüsselwörter, hier Concepts genannt, welche Themen im Inhalt darstellen. Im Gegensatz zum Keyword Extraction API müssen diese Concepts nicht im Text enthalten sein, sondern beschreiben Themen des Textes auf einer höheren Abstraktionsstufe. Auch Concepts wird ein Relevanzfaktor und eine Stimmung zugeordnet.

Aus dem Text [16],

> The more things change... Yes, I'm inclined to agree, especially with regards to the historical relationship between stock prices and bond yields. The two have generally traded together, rising during periods of economic growth and falling during periods of contraction. Consider the period from 1998 through 2010, during which the U.S. economy experienced two expansions as well as two recessions: Then central banks came to the rescue. Fed Chairman Ben Bernanke led from Washington with the help of the bank's current $3.6T balance sheet. He's accompanied by Mario Draghi at the European Central Bank and an equally forthright Shinzo Abe in Japan. Their coordinated monetary expansion has provided all the sugar needed for an equities moonshot, while they vowed to hold global borrowing costs at record lows.

erzeugt die Concept Tagging API folgende Concepts:

| Concept | Relevanz |
|---|---|
| Monetary policy | 0.972084 |
| Inflation | 0.720149 |
| Central bank | 0.70608 |
| Federal Reserve System | 0.694171 |
| Money supply | 0.648812 |
| Economics | 0.635387 |
| Great Depression | 0.603431 |
| Chairman of the Federal Reserve | 0.555597 |

**Tabelle 3.2:** Resultate der Concept Tagging API

**AlchemyAPI in CAMC** Um auf die oben genannten Funktionen zuzugreifen stellt AlchemyAPI verschiedene *Software Development Kits* (SDKs) zur Verfügung, darunter auch eines für das .NET Framework. Dieses SDK ist in CAMC integriert und stellt die Verbindung zur AlchemyAPI her. Sowohl Schlüsselwörter, als auch Concepts und Kategorie werden pro Klienten-Nachricht generiert und einem Live Chat als *Keywords* zugeordnet, siehe Abbildung 3.6. Das heißt, nur die Nachrichten vom Klienten werden analysiert und das nicht als Ganzes, sondern separiert. Dieses Vorgehen hat den Vorteil, dass bei jeder neuen Klient-Nachricht, anstatt des ganzen Nachrichtenverlaufes, nur eine einzelne Nachricht an AlchemyAPI gesendet wird. Ein Nachteil ist jedoch, dass mehrere Nachrichten im Zusammenhang mehr Informationen enthalten, wie eine einzelne Nachricht. Letztendlich häuft jeder Live Chat im Laufe der Zeit eine Sammlung von Concepts, Kategorien und Schlüsselwörtern an, welche nachdem sie dem Live Chat zugeordnet sind, nicht weiter unterschieden und als *Keywords* gespeichert werden.

## 3.5 Affiliate Systeme

In der Agenten-Konsole sollen einem Agenten Produkte angezeigt werden, die er in die Unterhaltung einbaut, mit dem Ziel, dass der Klient die Produkte erwirbt. Diese Produkte stammen aus Affiliate Systemen. Affiliate Systeme sind internetbasierte Vertriebslösungen, wie beispielsweise die in Kapitel 2.2 vorgestellten Werbenetzwerke oder Partnerprogramme. Um Affiliate Systeme integrieren zu können, müssen diese aus technischer Sicht zwei Voraussetzungen erfüllen: Es muss eine API vorhanden sein, auf welche CAMC zugreifen kann und Links als Werbemittel müssen angeboten werden. Nach längerer Recherche hat sich nur das Amazon PartnerNet als passender Kandidat herausgestellt. Manche Affiliate Systeme sind auf bestimmte Produkte oder Regionen spezialisiert, folglich ist es durchaus

sinnvoll mehrere Affiliate Systeme in CAMC zu integrieren. Damit hat der Agent mehr Auswahlmöglichkeiten an Produkten, Vergütungsmodellen, etc. Daher sind im Rahmen dieser Arbeit zwei zusätzliche Mock-Affiliate-Systeme, names *ChatAd* und *AdLink* entwickelt und umgesetzt worden. Mock, da es keine realen Affiliate Systeme sind, sondern nur deren Funktion vortäuschen. Da das Amazon PartnerNet ein Partnerprogramm ist, sind die Mocks Werbenetzwerken nachgestellt. Es folgt eine genauere Betrachtung der in CAMC integrierten Affiliate Systeme. Für das Tracking und Targeting sind die Affiliate Systeme zuständig. Die nötigen Daten für das Targeting werden in CAMC generiert und an die Affiliate Systeme weitergeleitet.

**Mock-Affiliate-Systeme** ChatAd und AdLink sind aus technischer Sicht identisch. Beides sind Microsoft MVC 4 Webanwendungen und bestehen aus einer einfachen API mit Datenbankanbindung. Der Unterschied zwischen den Anwendungen liegt in den Produkten. AdLink enthält auch Produkte von deutschen Advertisern die in deutsch in der Datenbank vorliegen, ChatAd dagegen nur internationale Produkte, die in der englischen Sprachfassung in der Datenbank vorliegen. Listing 3.5 zeigt die Felder der Klasse `AdLink`. Diese Klasse ist ein Model und repräsentiert ein Produkt im Affiliate System AdLink und ist einem Produkt nachempfunden, wie es in vielen Shops in WWW angeboten wird. Das Produkt hat einen Namen und eine Beschreibung. Das Feld `Language` gibt an, auf welcher Sprache es angeboten wird. Das Feld `Url` enthält den Link zum Produkt und `Keywords` enthält Wörter, die das Produkt beschreiben. Die restlichen Felder sind in diesem Zusammenhang nicht von Bedeutung und werden nicht erläutert.

```
1  public class AdLink
2      {
3          public int Id { get; set; }
4          public string Description { get; set; }
5          public string Title { get; set; }
6          public string Language { get; set; }
7          public string Url { get; set; }
8          public virtual List<Keyword> Keywords { get; set; }
9          public virtual List<CustomField> CustomFields { get; set; }
10         public float Rate { get; set; }
11
12                 //...
13
14     }
```

**Listing 3.5:** Ausschnitt aus dem AdLink-Model

Das Produkt-Model in ChatAd ist gleichermaßen aufgebaut.

Die API in den Mocks ist eine ASP.NET Web API, wie sie im letzten Kapitel vorgestellt wurde. Es gibt nur einen Controller, den `AdController`. Dieser enthält die Action `Get`, die die Aufgabe hat, Produkt-Anfragen entgegenzunehmen und passende Produkte zurückzugeben. Listing 3.6 zeigt einen Ausschnitt aus dem AdController der AdLink-

Anwendung. Die Action `Get` nimmt als Parameter ein `AdRequestModel` entgegen, welches Informationen über die angeforderten Produkte enthält. So ist beispielsweise die angeforderte Sprachfassung und eine Liste an Suchbegriffen, die das Produkt beschreiben, im AdRequestModel enthalten. Diese Suchbegriffe wurden zuvor durch die AlchemyAPI aus den Nachrichten des Klienten generiert.

```
1  public class AdController : ApiController
2  {
3    [HttpPost]
4    Public List<AdLinkLight> Get(AdRequestModel model)
5    {
6      AdLinkDb _db = new AdLinkDb();
7      //...
8      var AdLinks = (from AdLink in _db.AdLinks
9      where AdLink.Language == model.Language
10       select AdLink).ToList();
11     foreach (var al in AdLinks)
12     {
13       //[0 <= Rate <= 1] keywords are weighted with 50 %
14       al.Rate = al.getKeywordStringList().Intersect(model.GetKeywordArray(),
15         StringComparer.OrdinalIgnoreCase).Count() / (float)al.Keywords.Count() * 0.5f;
16       Boolean titleMatch = false;
17       foreach (var kw in model.GetKeywordArray())
18       {
19         if (al.Title.IndexOf(kw, StringComparison.OrdinalIgnoreCase) >= 0 && !titleMatch)
20         {
21           //Keywords in title are weighted with 20%
22           al.Rate += 0.20f;
23           titleMatch = true;
24         }
25
26         if (al.Description.IndexOf(kw, StringComparison.OrdinalIgnoreCase) >= 0)
27         {
28           al.Rate += (1 / (float)model.GetKeywordArray().Count()) * 0.3f;
29         }
30       }
31     }
32     AdLinks = AdLinks.Where(p => p.Rate > 0.3).OrderBy(al => al.Rate).ToList();
33     List<AdLinkLight> AdLinksLight = new List<AdLinkLight>();
34     foreach (var al in AdLinks)
35     {
36       AdLinkLight all = al.getAdLinkLight();
37       all.Url += "&partnerid=" + partnerId;
38       AdLinksLight.Add(all);
39     }
40     return AdLinksLight;
41     //...
42   }
43 }
```

**Listing 3.6:** AdController

Zu erst werden alle Produkte in der passenden Sprache herausgesucht (Zeile 7). Anschließend werden unpassende Produkte durch einen einfachen Algorithmus aussortiert (Zeilen 14 bis 37).

Der Sortieralgorithmus durchsucht Title, Description und Keywords des Adlink-Models nach den Suchwörtern des AdRequestModels. Dabei wird zu jedem AdLink ein Wert berechnet, der angibt, wie zutreffend der AdLink bezüglich der Schlüsselwörter ist. Dieser Wert liegt zwischen 0 und 1 (inklusive) und berechnet sie wie folgt:

Sei $S$ die Menge der Suchbegriffe mit $n = |S|$ und $K$ die Menge der Keywords mit $k = |K|$. Sei $T$ die Menge Wörter im Titel und $D$ die Menge der Wörter in der Description des AdLinks.

Sei $\mathbf{1}_T$ definiert durch $\mathbf{1}_T : T \to \{0, 1\}$, $\mathbf{1}_T(x) = \begin{cases} 0 & \text{für } x \notin T \\ 1 & \text{für } x \in T \end{cases}$

Dann ist $s \in [0, 1]$ mit $s = 0.2 \cdot \mathbf{1}_T(S) + 0.5 \cdot \frac{|S \cap K|}{k} + 0.3 \cdot \frac{|S \cap D|}{n}$.

Im Idealfall $s = 1$, ist mindestens ein Suchbegriff im Titel, sind alle Keywords durch Suchbegriffe abgedeckt und alle Suchbegriffe kommen mindestens einmal in der Description vor. Alle AdLinks mit einem Wert $s \leq 0.3$ werden aussortiert.

In Zeile 46 von Listing 3.6 wird eine Liste von AdLinks zurückgegeben. Diese Liste wird in den angefragten Rückgabetyp umgewandelt. Enthält das Content-Type-Feld im HTTP-Request beispielsweise den Wert *application/json*, so wird die AdLinks-List in das JSON-Format konvertiert und zurückgegeben. In ChatAd ist der Algorithmus leicht abgewandelt, die Gewichtungen sind anders. Suchwörter im Titel werden mit einem Faktor von 0.3 gewichtet, Treffer in den Keywords mit 0.4 und Treffer in der Description mit 0.3.

**Amazon PartnerNet** Das Amazon PartnerNet ist ein Parnterprogramm das Webseitenbeitreibern ermöglicht Partnerlinks zu Produkten auf Amazon in die eigene Webseite einzubinden. Werden dadurch Käufe generiert, erhält der Webseitenbetreiber eine Vergütung. Dieses Modell lässt sich auf einen CAMC Live Chats übertragen. Anstatt, dass der Webseitenbetreiber die Links auf seiner Webseite einbindet, werden Links durch Agenten in Chats eingebunden und somit Produkte empfohlen. Idealerweise lässt sich über die Agenten-Konsole nach Produkten auf Amazon suchen. Dazu muss ein Webseitenbetreiber als Publisher im Amazon PartnerNet registriert sein und eine gültige Amazon *PartnerID* besitzen. Werden Käufe generiert, wird dem Webseitenbetreiber eine Vergütung ausgeschüttet, nicht den Agenten.

Um gezielt nach Produkten suchen zu können, stellt Amazon die *Product Advertising API* zur Verfügung. Diese wird in CAMC genutzt, um in der Agenten-Konsole Partnerlinks anzuzeigen. Die API unterstützt unter anderem REST-Anfragen. Zusammengefasst werden alle Schlüsselwörter eines Live Chats zusammen mit der Amazon PartnerID und anderen Informationen in einem GET-Request an die Product Advertisig API geschickt. Diese liefert Produktergebnisse in Form von XML zurück. In CAMC werden die Ergebnisse verarbeitet und einem Agenten in der Agentenkonsole angezeigt. Auch können über

das Suchfeld in der Agenten-Konsole neue Schlüsselwörter zum Live-Chat hinzugefügt werden. Für technische Details sei an dieser Stelle auf die Dokumentation verwiesen [7].

Abbildung 3.9 zeigt das Resultat der Integration des Amazon PartnerNet und der Amazon Product Advertising API, die Agenten-Konsole in CAMC. Auf der Linken Seite sind Produkte aus Amazon erkennbar. In diesem Fall handelt es sich um Bücher. Diese sind das Resultat der Anfrage an die Amazon Product API, Produkte zu liefern, die den Suchbegriffen *java pogramming*, *computer_internet* und *Programming language* entsprechen. Diese Suchbegriffe wiederum, sind Keywords, oben rechts zu erkennen, die aus der Nachricht von Alice extrahiert wurden. Durch das Suchfeld oben rechts, kann die Suche nach Produkten verfeinert werden. Es ist auch möglich Keywords mit einem Klick zu deaktivieren.

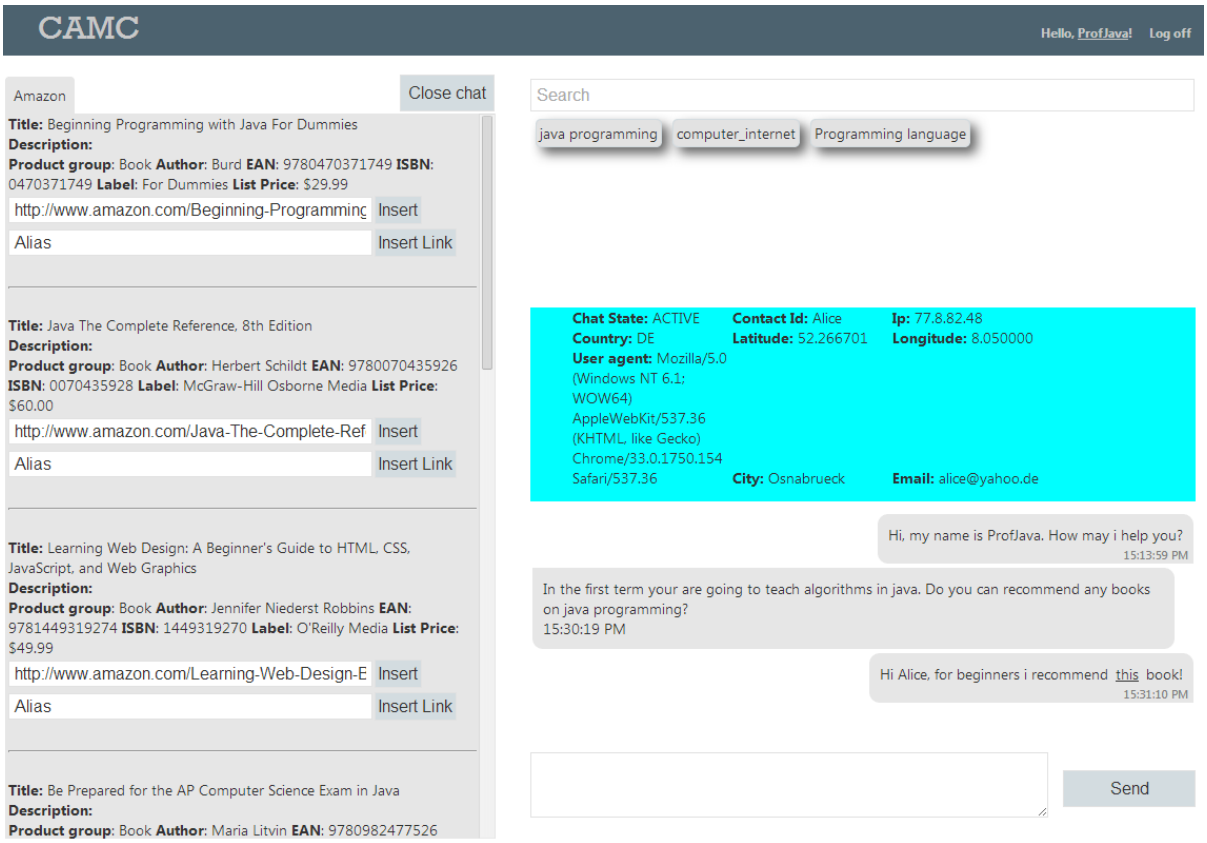


**Abbildung 3.9:** Screenshot: Agenten-Konsole mit Produkten von Amazon

## 3.6 Chat API

Um einen Live Chat in andere Anwendungen zu integrieren, oder um Klient-Chat und Agenten-Konsole umgestalten zu können gehört zu CAMC eine RESTful Chat API die es ermöglicht einen Live Chat zu starten. So wurde beispielsweise im Rahmen dieses Projektes ein Android Klient-Chat für Smartphones angefertigt. Diese App nutzt die Chat API um einen Live Chat über ein Smartphone zu betreiben.

Die Chat API ist eine ASP.NET Web API und enthält zwei Controller, den `ChatController` und den `ChatEventController`. Um die Chat API überhaupt nutzen zu können, muss im internen Bereich des Webseitenbetreibers ein API-Schlüssel erstellt werden und bei jeder Anfrage an die API im Authorization-Header des HTTP-Requests mitgegeben werden. Dieser API-Schlüssel ist ein *Global Unique Identifier* (GUID) und besteht aus einer 128Bit Zahl. Der API-Schlüssel dient zum Schutz vor unerlaubtem Zugriff auf die API und identifiziert den Webseitenbetreiber. Ein Live Chat kann drei Zustände annehmen, *CONNECTING*, *ACTIVE* und *ENDED*.

Tabelle 3.3 zeigt die Schnittstelle des ChatControllers. Ein GET-Request an diese Schnittstelle erzeugt in CAMC einen neuen Live Chat und liefert eine LiveChatId, auch eine GUID, welche den Live Chat eindeutig identifiziert. Zu diesem Zeitpunk befindet sich der Live Chat im Zustand CONNECTING.

| **URI** | **Funktion** |
|---|---|
| .../api/chat/get | Liefert eine eindeutige ChatId |
| .../api/chat/post | Initiert einen neuen Chat |

**Tabelle 3.3:** ChatController

Um in den Zustand ACTIVE zu wechseln und Nachrichten auszutauschen, muss ein POST-Request an den ChatController gesendet werden. Diese Anfrage enthält unter Anderen, die einen Schritt zuvor erzeugt LiveChatId, die Sprache des Chat-Klienten, den Namen eines gewünschten Agenten und weitere Informationen zum Klienten, wie etwa Ortsdaten, E-Mail, Geschlecht, Alter und viele weitere Informationen, welche beispielsweise zu Targeting-Zwecken genutzt werden können. Erforderlich ist jedoch nur die LiveChatId.

Der `ChatEventController` dient in erster Linie zum Nachrichtenaustausch. Ein `ChatEvent` kann vom Typ `ChatMessage` oder `ChatState` sein. Das ChatState enthält den Chat-Zustand. Ein ChatEvent vom Typ ChatMessage enthält eine Nachricht. Über die Get-Action können beispielsweise durch Polling neue Nachrichten abgefragt, oder der Chat-Zustand ausgelesen werden. Mit der Post-Action können neue Nachrichten versendet und

| **URI** | **Funktion** |
|---|---|
| ../api/chat/get | Liefert neuste Events. Beispielsweise neue Nachrichten. |
| ../api/chat/post | Erzeugt ein neues Chat-Event. |

**Tabelle 3.4:** ChatEventController

der Chat-Zustand geändert werden. Hier muss zusätzlich zum API-Schlüssel, bei jeder Anfrage die LiveChatId mitgesendet werden. Siehe Tabelle 3.4.

# Kapitel 4

# Zusammenfassung

Im Rahmen dieser Arbeit ist ein Prototyp einer Live-Chat-Webanwendung entstanden. Dieser Live Chat ist durch eine Affiliate-Marketing-Komponente erweitert, die es ermöglicht Links zu Produkten in einen Chat einzubinden. Dadurch wird der Live Chat zu einer Werbefläche. Dieses Kapitel enthält eine abschließende Betrachtung dieser Arbeit und erläutert kurz kritische Punkte.

## 4.1 Kritische Systempunkte

**Affiliate Systeme** Affiliate Systeme die in CAMC integriert sind, müssen bestimmte Voraussetzungen erfüllen: Erstens, sie müssen eine API enthalten auf die CAMC zugreifen kann, um Werbemittel zu beziehen. Zweitens, sie müssen als Werbemittel einfache Links anbieten. Eine intensive Recherche ergab jedoch nur das Amazon PartnerNet als passendes Affiliate System. Dies schließt zwar die Existens weiterer passender Anbieter nicht aus, verdeutlicht aber die Problematik. Es ist zum Beispiel durchaus denkbar, dass in einem Live Chat nicht nur materielle Güter empfohlen werden, wie sie Amazon anbietet, sondern auch immaterielle Güter. Damit das Affiliate Marketing nicht nur auf bestimmte Produkte oder Advertiser beschränkt ist, ist eine vielzahl von Affiliate Systemen erforderlich, die in CAMC integriert werden. Das Amazon PartnerNet ist ein Partnerprogramm, interessant ist jedoch auch die Integration von Werbenetzwerken, da diese eine vielzahl von Advertisern und Produkten anbieten. Aus diesem Grund sind zusätzlich die Mock-Affiliate-Systeme ChatAD und AdLink entstanden, die Werbenetzwerken nachempfunden sind.

**Chat-Analyse** Idealerweise soll einem Agenten automatisch, zur Unterhaltung passende Produkte angezeigt werden. Hat ein Klient beispielsweise Fragen zur Softwareentwick-

lung mit dem .NET Framework, so ist es eventuell sinnvoll, einem Agenten Links zu beliebten Büchern, zu diesem Thema anzuzeigen, die er dem Klienten empfehlen kann. Dadurch kann sich ein Agent die Zeit zum Suchen nach Produkten sparen. Das Vorgehen dazu, wie ist in Kapitel 3.4.2 beschrieben ist, liefert nur mangelhafte Resultate. Denn Keywords, die im Laufe eines Chats, durch die AlchemyAPI erzeugt werden, werden ohne weitere Verarbeitung, an Affiliate Systeme weitergeleitet. Oft sind jedoch Keywords enthalten, welche keinen Bezug zu Produkten haben.

Aus der Frage, *Hi, my name is Bob, what does MVC in Microsoft MVC 4 Framework mean?* erzeugt die AlchemyAPI die Keywords *Hi, Bob, Microsoft MVC* und *Framework mean*. Die ersten beiden Keywords stehen nicht mit der Problemstellung im Zusammenhang und würden das Ergebnis einer Produktsuche eventuell verfälschen. Daher ist an dieser Stelle eine Bearbeitung und Filterung der Keywords sinnvoll. Da diese Funktion jedoch nicht zwingend erforderlich für die Integration von Produkten in Live Chats ist, wurde auf solche Maßnahmen verzichtet. Eine Filterung von Keywords ist in der Suchfunktion der Agentenkonsole enthalten. Dort werden zusätzlich zu den Suchwörtern auch die automatisch generierten Keywords angezeigt. Diese kann man deaktivieren, das heißt sie werden im deaktivierten Zustand nicht an Affiliate Systeme geschickt.

## 4.2 Fazit und Ausblick

Diese Arbeit zeigt, dass es möglich ist Methoden aus dem Affiliate Marketing in Live Chats zu integrieren und einen Live Chat als eine Art Werbefläche zu nutzen. Es ist ein Prototyp eines Live Chat entstanden (CAMC), der durch eine Affiliate-Marketing-Komponente erweitert ist. Diese Komponente besteht aus verschiedenen Softwarekomponenten die im Zusammenspiel, einem Agenten ermöglichen Produkte aus Affiliate Systemen in einen Chat einzubauen. Ein nächster Schritt könnte beispielsweise sein, CAMC weiterzuentwickeln, um den Status eines Protoypen zu verlassen. So fehlen CAMC beispielsweise sicherheitstechnische Funktionen, wie die Verschlüsselung von Nachrichten. Nachdem CAMC um wichtige Funktionen erweitert wurde, müsste noch ausführliches Testen und Refactoring durchgeführt werden. Hat CAMC einen stabilen Zustand erreicht, könnte man es zum Beispiel nutzen, um die Effektivität eines Live Chats als Werbefläche zu analysieren und zu bewerten.

# Literaturverzeichnis


[1] *Yahoo.* http://www.yahoo.com/, 2013. – [Online; accessed 29-January-2014]

[2] *Yahoo Ad Manager.* https://admanager.yahoo.com/, 2013. – [Online; accessed 29-January-2014]

[3] *Alexa Top 500 Global Sites.* http://www.alexa.com/topsites, 2014. – [Online; accessed 29-January-2014]

[4] *ASP.NET SignalR Incredibly simple real-time web for .NET.* http://signalr.net/. Version: 214. – [Online; accessed 20-February-2014]

[5] AlchemyAPI: *AlchemyAPI — Transforming text into knowledge.* http://www.alchemyapi.com/. Version: 2014. – [Online; accessed 06-March-2014]

[6] Amazon: *AmazonPartnerNet.* https://partnernet.amazon.de/. Version: 2014. – [Online; accessed 19-February-2014]

[7] Amazon: *Product Advertising API Getting Started Guide (API Version 2011-08-01).* http://docs.aws.amazon.com/AWSECommerceService/2011-08-01/GSG/Welcome.html. Version: 2014. – [Online; accessed 06-March-2014]

[8] Bauer, Christoph; Greve, Goetz; Hopf, Gregor: *Online Targeting und Controlling* -. 2011. Aufl. Berlin : Springer DE, 2011. – ISBN 978–3–834–96742–8

[9] Benne, Jon: *Denver Broncos sign LB Paris Lenon.* http://www.sbnation.com/nfl/2013/8/20/4640600/paris-lenon-denver-broncos. Version: 2013. – [Online; accessed 06-March-2014]

[10] ECMA: *ECMA-334: C# Language Specification.* pub-ECMA:adr : ECMA (European Association for Standardizing Information and Communication Systems),

2001. – xiii + 479 S. http://www.ecma-international.org/publications/files/ecma-st/ECMA-334.pdf

[11] ECMA International: *Standard ECMA-335 - Common Language Infrastructure (CLI)*. 6. Geneva, Switzerland, 2012 http://www.ecma-international.org/publications/standards/Ecma-335.htm

[12] Galloway, Jon: *Professional ASP.NET MVC 4*. Hoboken, N.J. Chichester : Wiley John Wiley distributor, 2012. – ISBN 111834846X

[13] Google: *Google AdSense*. http://www.google.de/adsense. Version: 2014. – [Online; accessed 03-February-2014]

[14] Google: *Google AdWords*. https://adwords.google.de. Version: 2014. – [Online; accessed 19-February-2014]

[15] Imran Khan, Vasily Karasyov Lev Polinsky Joseph B. Bridget Weishaar W. Bridget Weishaar: The Rise of Ad Networks . (2007), Oktober

[16] Johnson, Adam: *Investors Start to See Post-Stimulus World Approaching*. http://www.bloomberg.com/news/2013-08-19/investors-start-to-see-post-stimulus-world-approaching.html. Version: 2013. – [Online; accessed 06-March-2014]

[17] Kurtz, Jamie: *ASP.NET MVC 4 and the Web API building a REST service from start to finish*. Berkeley, CA New York : Apress Distributed to the book trade worldwide by Springer, 2013. – ISBN 1430249773

[18] Lammenett, Dr. E.: *Praxiswissen Online-Marketing*. London : Springer Gabler, 2012

[19] Lerman, Julie: *Demystifying Entity Framework Strategies: Model Creation Workflow*. http://msdn.microsoft.com/en-us/magazine/hh148150.aspx. Version: Mai 2011. – [Online; accessed 13-February-2014]

[20] LiveChat, Inc.: *LiveChatInc*. http://www.livechatinc.com/, 2013. – [Online; accessed 18-December-2013]

[21] Liveninja: *Liveninja*. https://www.liveninja.com, 2013. – [Online; accessed 18-December-2013]

[22] LivePerson, Inc.: *LivePerson*. http://www.liveperson.com/, 2013. – [Online; accessed 18-December-2013]

[23] LivePerson, Inc.: *LivePerson Experts*. http://www.liveperson.com/experts, 2013. – [Online; accessed 18-December-2013]

[24] MacDonald, Matthew: *Beginning ASP.NET 4.5 in C#*. S.l : Apress, 2012. – 11 S. – ISBN 978–1–4302–4251–2

[25] Mason, Jeff: *5 REASONS TO CONVINCE YOU TO START*. http://www.velaro.com/blog/5-reasons-to-use-live-chat-service, 2013. – [Online; accessed 28-January-2014]

[26] Maxmind: *MaxMind - IP Geolocation and Online Fraud Prevention*. http://www.maxmind.com/de/home. Version: 2014. – [Online; accessed 06-March-2014]

[27] Microsoft: *LINQ to SQL [LINQ to SQL*. http://msdn.microsoft.com/de-de/library/bb386976(v=vs.110).aspx. Version: 2013. – [Online; accessed 13-February-2014]

[28] Microsoft: *Common Language Runtime (CLR)*. http://msdn.microsoft.com/en-us/library/8bs2ecf4(v=vs.110).aspx. Version: 2014. – [Online; accessed 04-February-2014]

[29] Microsoft: *Managed Execution Process*. http://msdn.microsoft.com/en-us/library/k5532s8a(v=vs.110).aspx. Version: 2014. – [Online; accessed 04-February-2014]

[30] Microsoft: *Overview of the .NET Framework*. http://msdn.microsoft.com/en-us/library/zw4w595w(v=vs.110).aspx. Version: 2014. – [Online; accessed 04-February-2014]

[31] Team, Microsoft A.: *ASP.NET MVC Overview*. http://www.asp.net/mvc/tutorials/older-versions/overview/asp-net-mvc-overview. Version: January 27, 2009. – [Online; accessed 25-February-2014]

[32] W3Techs: *Usage of server-side programming languages for websites*. http://w3techs.com/technologies/overview/programming_language/all. Version: 2013. – [Online; accessed 07-February-2014]

[33] Wikipedia: *Advertising network — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/wiki/Advertising_network. Version: 2014. – [Online; accessed 30-January-2014]

[34] Wikipedia: *Entity-Relationship-Modell — Wikipedia, The Free Encyclopedia*. http://de.wikipedia.org/wiki/Entity-Relationship-Modell. Version: 2014. – [Online; accessed 04-March-2014]

[35] Wikipedia: *HTTP-Cookie — Wikipedia, The Free Encyclopedia*. http://de.wikipedia.org/wiki/HTTP-Cookie. Version: 2014. – [Online; accessed 30-January-2014]

[36] Wikipedia: *Representational State Transfer — Wikipedia, The Free Encyclopedia*. http://de.wikipedia.org/w/index.php?title=Representational_State_Transfer. Version: 2014. – [Online; accessed 19-February-2014]

[37] Wikipedia: *WebSocket — Wikipedia, The Free Encyclopedia*. http://de.wikipedia.org/wiki/WebSocket. Version: 2014. – [Online; accessed 30-January-2014]

[38] Williams, Alex: *AlchemyAPI Raises 2 Million For Neural Net Analysis Tech, On Par With IBM Watson, Google*. http://tinyurl.com/nrvru3f. Version: 2013. – [Online; accessed 06-March-2014]

# Erklärung

Ich versichere, dass ich die eingereichte Bachelorarbeit selbstständig und ohne unerlaubte Hilfe verfasst habe. Anderer als der von mir angegebenen Hilfsmittel und Schriften habe ich mich nicht bedient. Alle wörtlich oder sinngemäß den Schriften anderer Autoren entnommenen Stellen habe ich kenntlich gemacht.

Osnabrück, März 2014