ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Bachelor-Thesis

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)
an der Hochschule für Technik und Wirtschaft des Saarlandes
im Studiengang Kommunikationsinformatik
der Fakultät für Ingenieurwissenschaften

Migration eines webbasierten Bestellungssystems in eine .Net-Umgebung mit Umbraco Content-Management Funktionalität

vorgelegt von
Bozhidar Aleksandrov

betreut und begutachtet von
Prof. Dr. Helmut Folz
Thomas Beckert, M.Sc.

Saarbrücken, 30. September 2018

Selbständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich erkläre hiermit weiterhin, dass die vorgelegte Arbeit zuvor weder von mir noch von einer anderen Person an dieser oder einer anderen Hochschule eingereicht wurde.

Darüber hinaus ist mir bekannt, dass die Unrichtigkeit dieser Erklärung eine Benotung der Arbeit mit der Note „nicht ausreichend“zur Folge hat und einen Ausschluss von der Erbringung weiterer Prüfungsleistungen zur Folge haben kann.

Saarbrücken, 30. September 2018

Bozhidar Aleksandrov

Zusammenfassung

Im Rahmen dieser Abschlussarbeit wird eine webbasierte Software, die die Internetseite "jungekueche"verwaltet, betrachtet. Diese Software wurde auf Basis von das Framework ASP erstellt. Der Grund dieser Analyse ist das nicht mehr zeitgemäße Design. Weiterhin ist das Ziel die Migration von der bereits erwähnten Software zu einem Content Management System - Umbraco. Die Software wird analysiert, bewertet und auf Basis deren wird ein neues Konzept entwickelt. In der folgenden Dokumentation werden die Vorteile von Umbraco, erläutert. Eine ausführliche Beschreibung der Methodiken der Entwicklung dieses Konzept veranschaulicht mithilfe der verwendeten Werkzeugen das Migrationsprozess. Zum Schluss wird das Konzept anhand eines Prototyps des Programms begreiflich gemacht.

Model-driven software development offers the method of choice when it comes to manage complex software production projects. However, these concepts face some obstacles when applied to maintenance of existing software systems. In order to ally such modern methods it is frequently assumedthat re-coding cannot be circumvent

— Dr.-Ing. Hans-Georg Pagendarm [16]

Danksagung

Zunächst möchte ich Prof. Dr. Helmut Folz für die kompetente Betreuung der Arbeit und die Begutachtung der Bachelorarbeit danken.

Herrn M. Sc. Thomas Beckert danke ich für das interessante Arbeitsthema, die Bereitschaft, diese Arbeit als Zweitgutachter zu lesen und zu bewerten.

Außerdem danke ich Herrn B. Sc. Matheo Zech für professionelle Korrektur und nützliche Hinweise, um dieser Arbeit besser zu werden.

Zu guter Letzt ein herzliches Dankeschön an meine Eltern, meinen Bruder und ebenso an meine Freunde, die für den nötigen Ausgleich gesorgt und mich auf diese Weise stets motiviert haben, nicht nur während der Bachelorarbeit sondern auch über die gesamte Studienzeit.

Inhaltsverzeichnis

1 Einleitung

Seit mehreren Jahrzehnten zeichnet sich eine immer weiter voranschreitende Digitalisierung der Gesellschaft ab. Manche Wirtschaftsbetriebe und Einrichtungen haben sich schon früh damit arrangiert und entsprechende Produkte angeschafft. Dies umfasst fertig käufliche Hard- und Softwareprodukte, sowie individuell entwickelte Lösungen. Diese sind, wie die meisten digitalen Güter, einem schnellen Alterungsprozess unterworfen. Das können Schnittstellen sein, welche vom Hersteller nicht mehr unterstützt werden, oder eine neue Betriebssystem Version die nicht mehr Unterstützt wird. Wenn die Kunden direkt mit einer Software interagieren, ist Userexperience und Design ein nicht zu unterschätzender Faktor.

Die meisten Softwareinfrastrukturen wachsen meist mit einem Unternehmen und seinem Bedarf nach digitalen Lösungen. Das bedeutet, dass eine verworrene Struktur aus Abhängigkeiten entstehen kann. Das ist dabei eher die Regel als die Ausnahme.

Elektronische Datenverarbeitung (EDV)-Systeme haben nicht die klassischen Verschleißerscheinungen, so wie man sie von klassischen Betriebsmitteln kennt. Allerdings entsteht auch so ein Interesse nach einem gewissen Nutzungszeitraum die bestehende Software zu ersetzen. Man bezeichnet diesen Alterungsprozess, welchen man eingehend in der Softwarequalitätsforschung untersucht, als Softwarealterung. Software ist „weich“ und man sollte annehmen sie sei leicht änderbar und wartbar. Dies kann allerdings mit fortschreitendem Alter teurer sein als die Migration zu einem neuen System. Firmen die nicht mit der Zeit gehen werden schnell als alt und uninnovativ wahrgenommen. Dies kann sich schnell auf den Umsatz eines Unternehmens auswirken. Deshalb sind Migrationen gerade im Bereich des Web-Developments besonders häufig, bei denen man von Grund auf ein neues System erstellt.[16]

Es gibt mehrere Fälle von Softwaremigrationen. In manchen Fällen kann Software Hardware ersetzen. In andere Fällen ersetzt wiederum Software Hardware. Wesentlich häufiger ist jedoch das alte Software durch neue ersetzt wird, sowie alte Hardware durch ihre neueren Pendants.

1.1 Vorstellung des Unternehmens

Der Aufgabestellung wurde von der Firma SSitePoint"festgelegt. Das Unternehmen spezialisiert in den Bereichen Content Management System, Mobile Web Applications und e-commerce. Die Mitarbeiter sind .Net-Experten, die hochwertige Software auf Microsoft-Technologie setzen. SSitePointïst das einzige Unternehmen im Saarland, das Umbraco CMS verwendet. M. Sc. Thomas Beckert ist den Geschäftsführer der Firma und auch den Aufgabegeber der gegenwärtigen Arbeit. [9]

1.2 Motivation

In dieser Abschlussarbeit wird die Umstellung eines Bestellsystems eines Catering Services abgebildet werden. Dies bedeutet, das es sich bei dem Thema der Arbeit um eine Software zu Software Migration handelt. Genauer um eine modellgetriebene Softwaremigration.

Dabei wird das aus dem beginn dieses Jahrtausend stammende Bestellsystem, welches auf einem Windows Imaging Components (WIC)-Plugin basiert und einem Active Server Pages (ASP)-Backend. Da die damals verwendeten Technologien vom Hersteller Microsoft seit geraumer Zeit End-of-Life gesetzt wurden, ist eine Migration zu einem aktuelleren Technologiestack zwingend erforderlich. Das Ziel des Migrationsprojekt ist es, das aktuelle Bestellsystem, welches immernoch auf ASP basiert, durch einen modernen Technologiestack zu ersetzen.

Dazu zählt die neue Kozeptionierung des Frontend. Dies soll zur Verbesserung und Erleichterung der Bedienbarkeit führen. Dazu wird eine neue Seite, welche auf Umbraco und ASP.NET basiert, erstellt.

1.3 Zielsetzung

Dieses Projekt wird in zwei Paketen aufgeteilt. Das Ziel des ersten Pakets A ist dem Auftraggeber verschiedene Möglichkeiten zu bieten, die Inhalte seiner Seite zu editieren und zu ändern.

Im Paket B werden angefordert, dass ein Bestellsystem die Kommunikation zwischen dem Auftraggeber und seinem Kunde verwaltet. Weiterhin soll das System Buchhaltung und Auftragsspeicherung steuern. Zusätzlich wird die Erarbeitung eines Konzepts angefordert, das die Daten von der veralteten, webbasierten Software zu den obengenannten Anforderungen übertragen müssen.

1.4 Aufgabenstellung

Im nachfolgenden Kapitel werden alle verwendeten Technologien erörtert und kurz erklärt. Danach wird im darauf folgenden Kapitel der technische und optische Stand der aktuellen Internetpräsenz analysiert.

Im darauf folgenden vierten Kapitel erfolgt die Erfassung und Anforderungsanalyse der Problemstellung deren theoretischen Lösungen. Im nachfolgenden Kapitel wird darauf aufgebaut und es erfolgt die praktische Umsetzung der Lösungssätze. Im letzten Kapitel wird die Qualität der Umsetzung erörtert und persönliche Designentscheidungen begründet. Danach folgt ein Ausblick auf weitere Verbesserungsmöglichkeiten.

2 Genutzte Technologie und Services

In diesem Kapitel beschäftigt man sich mit den verwendeten Technologien und Webservices. Diese sind in der Webentwicklung sehr entscheidend, da davon die Userexperience und Wartbarkeit abhängt. Besonderer Augenmerk wurde dabei auf die möglichst weite Verbreitung der verwendeten Technologien gelegt. Dies ist ein Vorteil, da daraus resultiert das es eine große Community gibt, die die Projekte aktuell hält.

2.1 HTML

Die Grundlegende Sprache für das erstellen und Rendern von Internetseiten ist Hypertext Markup Language (HTML). Es ist im Grunde DIE Schlüsseltechnologie um Internetseiten aus dem World Wide Web (WWW) anzuzeigen. Natürlich kann man HTML auch für das erstellen und webbasierten lokalen Graphical User Interface (GUI) genutzt werden. Weitere Informationen findet man unter [3].

2.2 CSS

Cascading Style Sheets (CSS) ist eine moderne Technologie zur Gestaltung von Internetseiten. Dies dient der Gestaltung von HTML-Texten und liefert auch teilweise dynamische Funktionalität. Weitere Informationen findet man unter [18].

2.3 JavaScript

JavaScript ist eine objektorientierte Interpretersprache, die Webseiten erst dynamisch macht. Sie wird in diesem Kontext im Webbrowser des Anwenders ausgeführt und stellt die Client-Seite einer Applikation dar. [4]

2.4 JQuery

JQuery ist ein weit vebreitetes JavaScript-Framework. Es stellt Funktionen zur Verfügung welche ein leichtes manipulieren des Seiteninhalts ermöglicht. So kann die Entwicklung von komplexen Webprojekten bedeutend beschleunigt werden.[2]

2.5 AngularJS

Hierbei handelt es sich um ein JavaScript clientseitiges Framework, welches hochdynamische WebAnwendungen oder lokale Anwendungen mit Webtechnologie ermöglicht. Mithilfe von Angular ist Möglich neue Architekturkonzepte auf den Client zu bringen und komplexe Anwendungen zu entwickeln. [5]

2.6 Umbraco

Umbraco ist ein Content Management System (CMS). Es dient zum Erstellen, Bearbeiten und zur Verwaltung dynamischer Webseiten. Umbraco basiert auf C# und auf der ASP.Net-Technologie. Heutzutage werden Microsoft SQL Server, My SQL, VistaDB, Peta Poco und weitere Datenbanken verwendet. Dieses CMS ist Open Source und die erste Version ist vom dänischen Software-Entwickler Niels Hartving im Jahr 2000 veröffentlicht worden.

Aus folgenden Gründen hat man sich bei der Umsetzung für Umbraco entschieden [15]:

- Umbraco ist ein flexibles CMS. Es gibt keine unnötigen Optionen und Schaltflächen. Alles ist einfach zu benutzen und zu verstehen.
- Der intuitive Editor ermöglicht es jede Art von Content einfach einzupflegen. Seiten sind einfach zu bearbeiten oder zu aktualisieren und wird nach dem selben eingängigen Paradigma dargestellt. Es ist ohne Bedeutung mit welchem Gerät auf die Webseite zugegriffen wird - Umbraco ist immer responsiv.
- Es gibt keine Einschränkung welche Webentwicklungsprogrammiersprache man nutzen muss. Umbraco ist sehr anpassungsfähig.
- Sehr gut angepasst für agile Prozesse: „Im Vergleich zu anderen Systemen geht der Livegang Deines Projekts damit sehr schnell. Umbraco unterstützt die agilen Prozesse der modernen Digitalbranche, bei denen es essenziell ist, dass Editoren immer und überall Content publizieren können, ohne damit "Content Freezeßu verursachen. Gleichzeitig sollen auch Entwickler Bugfixes und Features schnell einbauen können. Umbraco sorgt dafür, dass der Flow nie endet“. [wieder da] [15]
- Umbraco CMS ist integrierbar. Man kann E-Comerce-Platform, CRM (Custom Relationship Management) System oder 3rd-Party Personalisation Engine verwenden. Ohne Probleme können individuelle Systeme integriert werden. Wegen Application-Programm-Interface (API) werden alle Daten mit sichtbarem Content mit dem Umbraco Front- und Backend vernetzt.
- Die Community vom Umbraco ist groß. Freundliche und aktive Umbraco Nutzer helfen jeder Zeit gegenseitig bei der Verbesserung des Codes. Viele aktive Tester sorgen dafür, dass Umbraco ständig verbessert wird.
- Das Modell der Lastverteilung ist in ASP.NET integriert. Jede ASP.NET-Webseite besitzt eigene Session-Verwaltung, die so konfiguriert werden kann, dass sie die Daten auf den SQL Server verlegen kann. So lassen sich Daten in einem gemeinsamen Datenspeicher ablegen. So ist es möglich, dass jeder Server auf den Datenstand eines Nutzers zugreifen kann.
- Volle Versionskontrolle, volle Integration in vorhandene Strukturen, zeitgesteuertes Veröffentlichen, Workflow-Orientierte Seitenverwaltung, schnelle Seitenvorschau vor der Veröffentlichung, Mehrsprachigkeit, Papierkorb zum einfachen Wiederstellen von gelöschten Elementen und seine Lizenzkostenfreiheit sind weitere Eigenschaften, weshalb man sich für Umbraco als CMS entschieden hat.

2.7 1&1 Website Check

Diese Anwendung überprüft, wie gut die betrachtete Webseite ist und was noch optimiert werden kann. Vier Aspekten werden geprüft [1]:

- Darstellung der Webseite
- Auffindbarkeit in Suchmaschinen
- Darstellung der Webseite
- Sicherheit der Webseite
- Geschwindigkeit der Webseite

Wenn die Internetadresse eingegeben wurde, wird Webseite aufgerufen. Danach wird der Quellcode analysiert.

2.8 HTTP Obeservatory

Mozilla HTTP Observatory ist ein Set von Tools, die zur Analyse und als Informationsquelle für Verbesserungen der Website dienen.[6]

3 IST-Analyse

In dieser Arbeit wird das Bestellsystem von der Party-Service-Website "jungekueche"betrachtet. Dazu zählen auch das Verwaltungssystem des Auftraggebers und die Verwaltungsseite des Kunden. Zuerst wird das Userinterface aus der Sicht von allen drei Akteuren beschrieben. Danach wird die Funktion des Systems evaluiert.

3.1 Beschreibung des User Interface (UI) des Bestellsystems

3.1.1 Anmeldeformular

Die Anmeldung und die Registrierung für neue Nutzer befindet sich auf einer gemeinsamen Seite. Das Anmeldeformular hat zwei Felder, in denen der Benutzer seinen PIN und seine E-Mail eingeben muss. Neben den Feldern steht ein Hinweis, welcher den Kunden mit den nötigen Informationen versorgt, wie er sich anmelden oder registrieren kann. In der Abbildung 3.1 wird Anmeldeformular dargestellt

Abbildung 3.1: Die bisherige Eingabemaske für das Kundenlogin

3.1.2 Registerformular

Falls der Kunde keine PIN hat, muss er sich registrieren. Dieses Formular besteht aus fünf Abschnitten. Im ersten muss der Kunde seine persönlichen Daten angeben. In dem zweiten steht die Informationen des Auftrags. Menü Auswahl ist der nächste Abschnitt. Im vorletzten wird die Zubehörauswahl und Serviceauswahl angezeigt. Der letzte Abschnitt ist die Erklärung zu dem Registrierungsprozess.

3.1.3 Auftraggeber - Verwaltung

Im Fall einer Registeranfrage kann der Auftraggeber sie bestätigen oder löschen. Dem Auftraggeber stehen drei Optionen zur Verfügung; Auftragsverwaltung, E-Mail Verwaltung und Umsatzverwaltung. Zuerst betrachtet man die Auftragsverwaltung. Dort befinden sich 11 Optionen, 9 davon leiten zu eigenen Unteroptionen weiter. Die anderen Zwei sind Speichern und Äbmelden". Durch diese Optionen kann der Auftraggeber die Kundedaten einsehen, editieren, bearbeiten und löschen. Außerdem kann er die dazugehörigen Nachrichten lesen und löschen. Auch kann er neue und alte Aufträge einsehen und bearbeiten oder löschen.

Abbildung 3.2: Die bisherige Eingabemaske für das Registrierungsformular

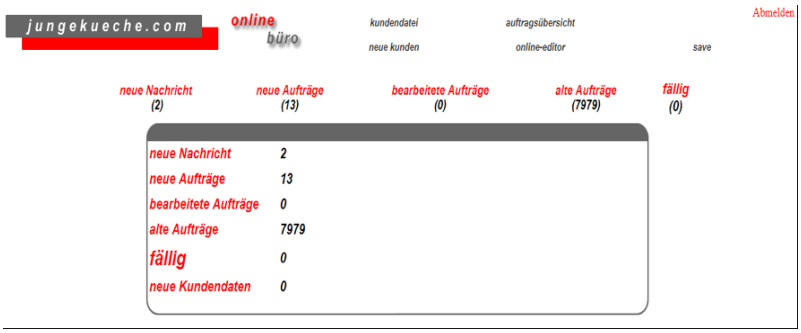

Abbildung 3.3: Die bisherige Eingabemaske für die Auftragsverwaltung

3.1.4 Kundenansicht

Wenn die Anfrage des Kunden bestätigt wird, bekommt der Kunde einen PIN per E-Mail. Mit diesem PIN und seiner E-Mail kann der Kunde sich bei "Jungekuecheänmelden. Nach dem Login-Prozess, befindet er sich auf einer persönlichen Nutzerseite wieder. Dort hat er eine Übersicht über seine aktuellen und alten Bestellungen, ein Kontaktfenster, in dem er die Nachrichten einsehen oder schreiben kann, sowie eine Art Newsletter. Außerdem gibt es eine Option, durch die der Kunde neue Bestellungen aufgeben kann.

3.2 Beschreibung der Funktionalität

Einer Anmeldung muss immer eine Registrierung vorausgehen. Diese Funktionalität wird mit ASP und ASP.NET realisiert. Die verwendeten Hilfsprogramme sind JQuery, JavaScript, AJAX, HTML und CSS.

Zuerst wird betrachtet, wie die Kommunikation zwischen den verschiedenen Technologien funktioniert. Das ganze Programm besteht aus Front- und Backend. Das Frontend bezeichnet den Teil des Programms, welches der Nutzer verwendet. Im Backend werden die Daten verarbeitet. Die Verbindung zwischen den beiden wird über JavaScript und JQuery realisiert. Wenn der Benutzer eine Tätigkeit ausführt, werden die eingegebenen Daten über JavaScript Methoden zu dem Backend weitergeleitet. Dort werden sie verarbeitet und über JavaScript Methode aufgerufen. Dort werden sie bearbeitet und wieder zu dem Frontend zugeschickt. Das Backend entsteht aus ASP-Funktionen, in denen sich verschiedenen Methoden befinden, sowie Datenbanken. In den Datenbanken werden die

Abbildung 3.4: Die bisherige Eingabemaske für die Auftragsverwaltung

eingegebenen Daten gespeichert.

Abbildung (neue Diagramm.. die Kommunikation zwischen Front- und Backend bzw. Client -Server)

Nach der grundlegenden Kommunikation zwischen Front- und Backend, wird erläutert wie die einzelnen Kommunikationsabläufe zwischen Front- und Backend koordiniert werden.

3.2.1 Anmelde- und Registerformular

Hier wurde noch nicht korrigiert

Nach einem Klick auf dem Feld "Bestellformular & Loginërscheint das Anmeldeformular sowie die Bestellseite. Wenn der Kunde auf dem Button Änmeldung"klickt, wird eine HTML Post-Methode aktiviert. So werden JavaScript- und Methoden von ASP-Datei aufgerufen. Die JavaScript-Methoden werden benutzt, damit die Eingabe auf Korrektheit geprüft wird. Im Folge werden Methoden von ASP-Datei werden aufgerufen.

Der Registerformular funktioniert auf demselben Prinzip. Nach Eingabe der benötigen Information, werden diese geprüft. Bei einem korrekten Input, werden die Eingaben zu dem zugeordneten Daten, bzw. Methoden zugeschickt.

weitere Informationen im Anhang - Funktionen

3.2.2 Auftraggeber-Verwaltung

Hier wird man mit dem Auftragsverwaltung konfrontiert. Die Auftragsverwaltung, wie es bereits beschrieben wurde, setzt sich aus verschiedenen Optionen zusammen. Die Funktionsweise der jeweiligen Option lässt sich separat betrachten.

3.2.2.1 Nachricht-Sektion

ermöglicht dem Auftraggeber die Information zu lesen oder löschen. Nach zugeordnetem Wahl wird die Information in der Datenbank "komentarëntweder aufgerufen oder entfernt.

3.2.2.2 Aufträge: Neue, alte und bearbeitende

Bei allen drei Arten von Aufträge stehen identische Verwaltungsmöglichkeiten zu Verfügung. Der Auftraggeber kann den Auftrag sehen, editieren oder löschen. Jede von diesen

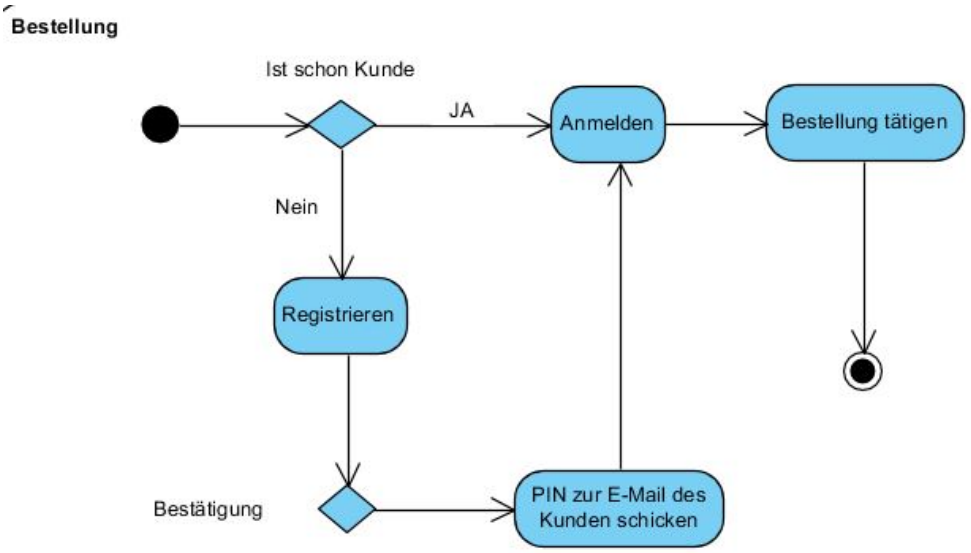

Abbildung 3.5: Die bisherige Eingabemaske für den Vorgang der Bestellung/Registrierung

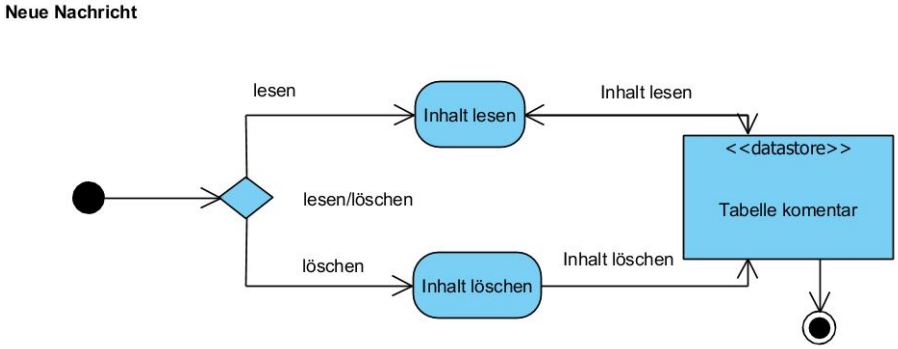

Abbildung 3.6: Die bisherige Eingabemaske für den Nachrichten lesen oder löschen

Aktivitäten ist mit mehrere Datenbanken verbunden, die verschiedenen Informationen über die gewählten Artikel beinhalten. In den Abbildungen 3.7 und 3.8 betrachtet man die jeweilige Funktionalität und welche Datenbanken werden zugegriffen. Es spielt keine Rolle welche Art von Auftrag gewählt wird, die Funktionalität umfasst dieselben Datenbanken.

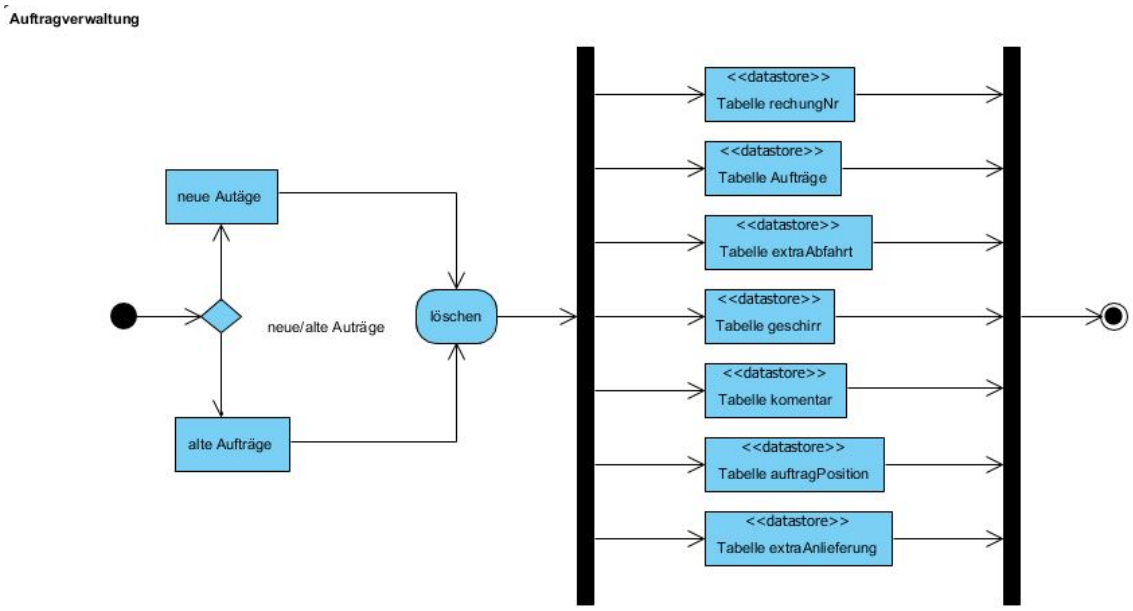

Abbildung 3.7: Die bisherige Eingabemaske für den Auftrag löschen

3.2.2.3 Kundendatei

Hier befindet sich die Information über die Kunden. Man hat die Möglichkeit die Information zu editieren oder zu löschen. Um der bestimmten Kunde schneller zu finden, steht eine Suchmaschine zur Verfügung. Eine bessere Übersicht wird in der Abbildung 3.9 dargestellt.

Wenn man editieren oder löschen möchte, geschieht dies über JavaScript Funktion. Diese Funktion ruft die Methoden auf, die sich in der zugeordneten ASP-Datei befindet. Im Ëditor-Fenster"kann die Daten des Kunden bearbeitet werden, Login-Daten wie Email an Kunden übermitteln, Wichtige Information für alle Kunden senden, V.I.P-Status des Kunden geben, spezifische Warnungen jeweiligem Kunde aufgeben, Information schreiben, sowie Unterschrift editieren und etc. Besser Verständnis zum diesen Fenster ergibt sich aus Abbildung 3.10

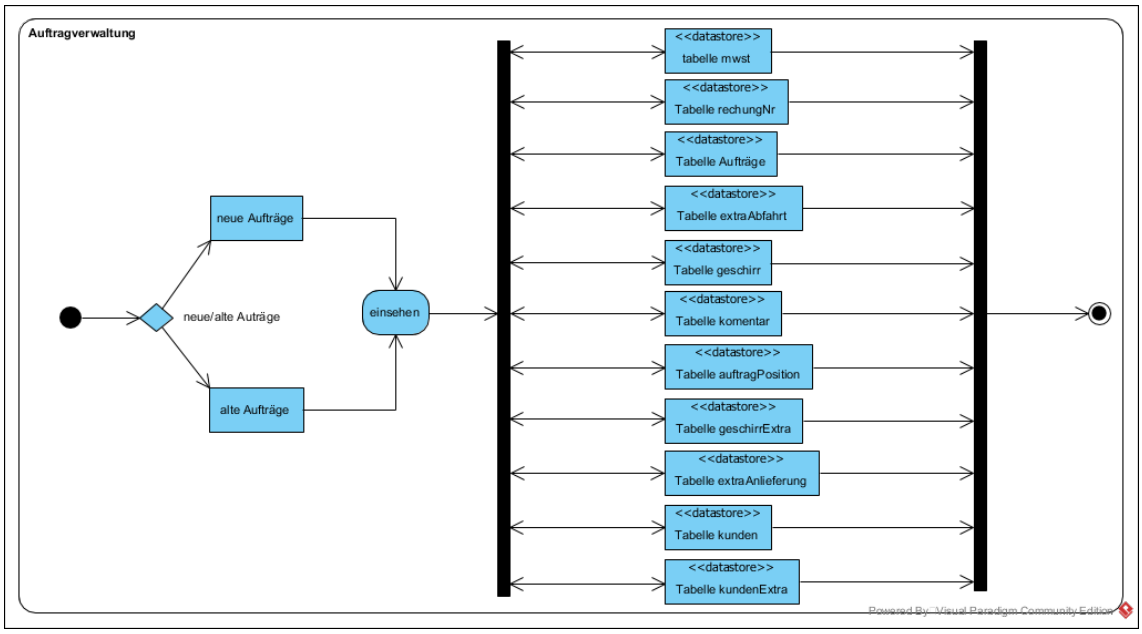

Abbildung 3.8: Die bisherige Eingabemaske für den Auftrag lesen

Abbildung 3.9: Die bisherige Eingabemaske für die Kundendatei

Durch JavaScript - Methode wird eine zusätzlichen Möglichkeit generiert, neue Nachrichten an dem Kunden zu schrieben. Sie wird aktiviert, wenn mit dem Cursor auf dem bestimmten Kunden von der Liste des Adressbuch gefahren wird. Die Kommunikation zwischen dem Auftraggeber und dem Kunde wird auch im Datenbank "komentar"gespeichert.

Anhand der vorherigen Beschreibungen und der folgenden Abbildungen kann ein vertieftes Verständnis für die Struktur eines Auftrages aufbauen. Von den Abbildungen 3.11 und 3.12 wird ersichtlich, wie die Aktivitäten des Auftraggeber geschehen und wie die Nachricht-Kommunikation abläuft und welche Datenbanken werden verwendet.

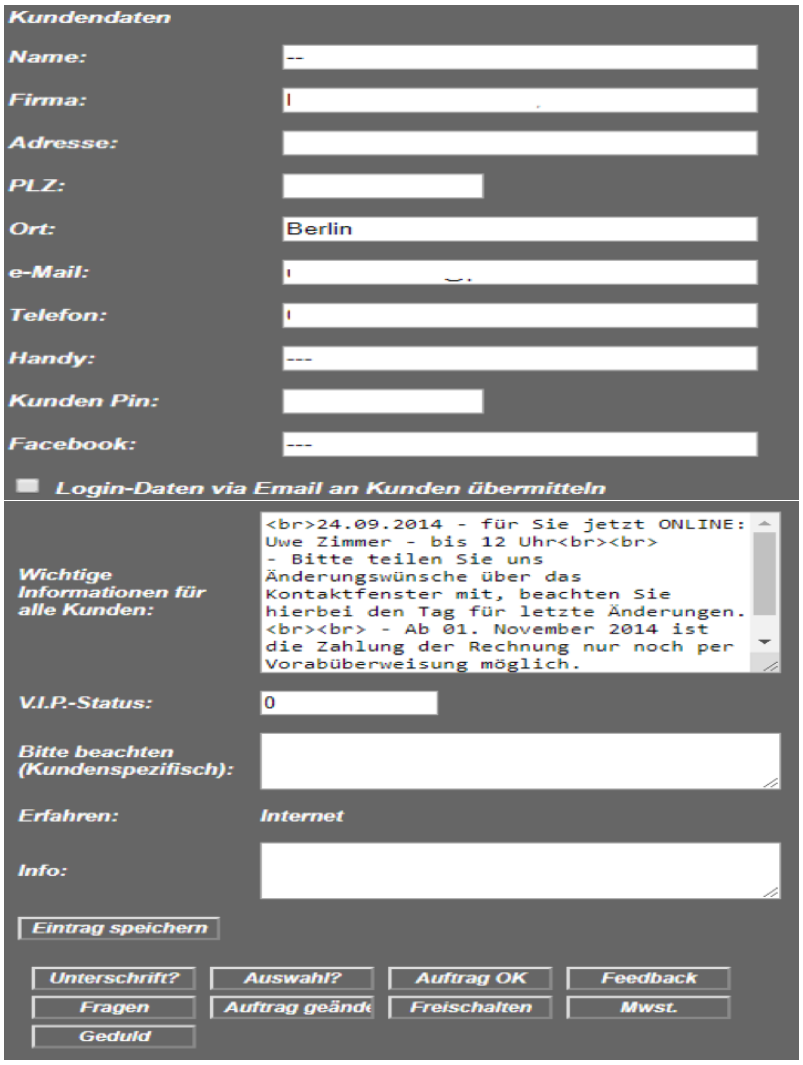

Abbildung 3.10: Die bisherige Eingabemaske für das Kunden-Editor

Die Kommunikationsverlauf geschieht von Frontend zum Backend über Datenbank.

3.2.2.4 Online-Editor

Durch diese Option können die neuen Artikel erstellt, editieren und löschen werden. Es gibt „Artikel – Arrangements“ und „Artikel – Standard“. Wenn ein Artikel bereits erstellt wurde, kann dieser entweder gelöscht, editiert oder zugeordnet werden. JavaScript Funktionen werden verwendet, um die Eingaben zu prüfen. Wenn die Eingaben treffend ausgefüllt worden sind, werden die obengenannten Methoden aus den jeweiligen Dateien aufgerufen. In der folgenden Abbildung 3.13 ist die Übersicht der Seite zu sehen.

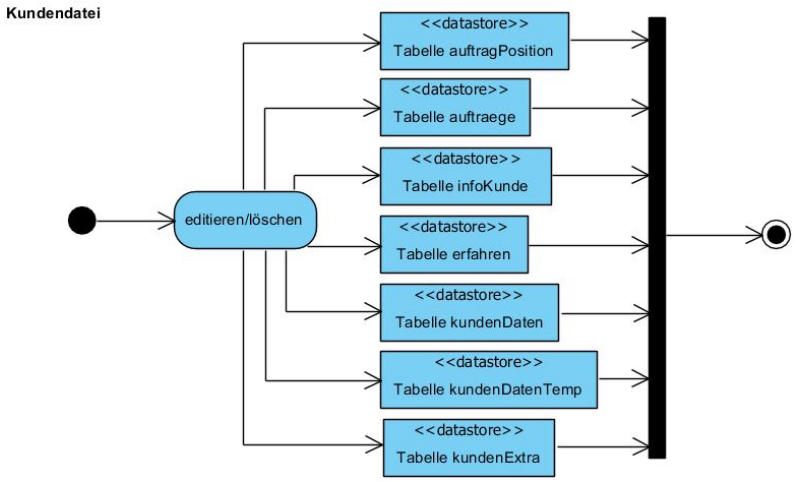

Abbildung 3.11: Die bisherige Eingabemaske für das Editieren/Löschen Funktion

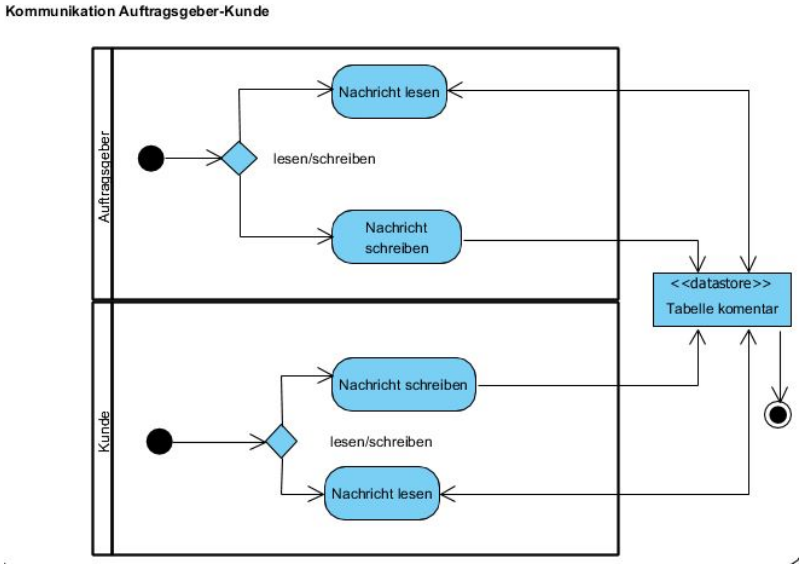

Abbildung 3.12: Die bisherige Eingabemaske für die Kommunikation zwischen dem Auftragsgeber und dem Kunde

Wenn die Option „Editieren“ zu dem „Artikel-Arrangements“ ausgewählt wird, wird eine neue Seite geöffnet, in der es vielfältige Möglichkeiten gibt, die Inhalte und Daten zu bearbeiten. Von der Abbildung 3.14 wird ersichtlich, wie diese Seite aussieht.

Artikel-Standard“ hat nur eine Möglichkeit. Sie wird in der Abbildung 3.15 dargestellt.

3.3 Testen

Für diese Arbeit wird ein webbasierten Sicherheit-Checker von 1&1. Nach eine vertiefte Recherche, zeigte sich, dass die Webseite auf eine mittlere Sicherheitsposition steht. Die Webseite ist nach vier Kriterien ohne Beanstandung abgesichert.

- Secure Sockets Layer (SSL) Verschlüsselung – Über SSL wird sichergestellt, dass zwischen User und Server die übertragenen Daten nicht gelesen werden können.
- Cookies sind auch sicher und so ist der Browser von Dritten über JavaScript unlesbar.

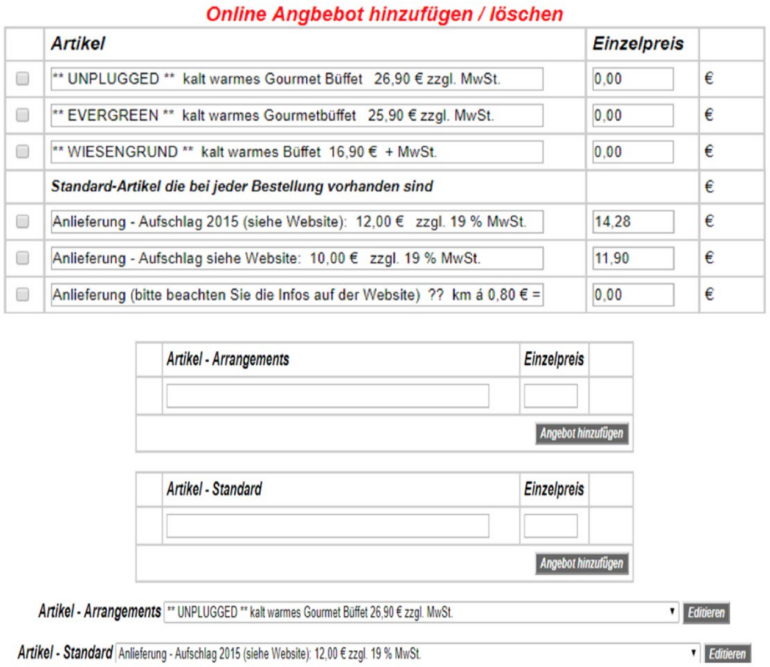

Abbildung 3.13: Die bisherige Eingabemaske für die Übersicht das Menü

- Apache-Status ist verboten. Die Status-Seite ist öffentlich nicht erreichbar. So wird den Schutz von potenziellen Angreifern erhöht.
- Server Version ist nicht sichtbar. Die Server Version ist öffentlich nicht einsehbar. Damit kann ein Angreifer nicht einfach bekannte Schwachstellen ausnutzen.

Wie man von der Abbildung **??** sehen kann, ist die Webseite gesichert, aber es gibt einige Bereiche, die eine Optimierung benötigen würde. Sie lädt sich langsam, wird im Browser relativ oft gefunden, aber ist sie gesichert.

Über noch ein Web-Checker werden die witere Funktionalitäten der Wrbseite analysiert. Observatory[**bentley:1999**] von Mozilla analysiert der Browser im vier Schritten.

- Hypertext Transfer Protocol (HTTP)
- Transport Layer Security (TLS)
- Secure Shell (SSH)
- **Third-Party Tests!** (**Third-Party Tests!**)

Erste zwei werden betrachtet, weil nur sie wichtig zu dieser Arbeit sind.

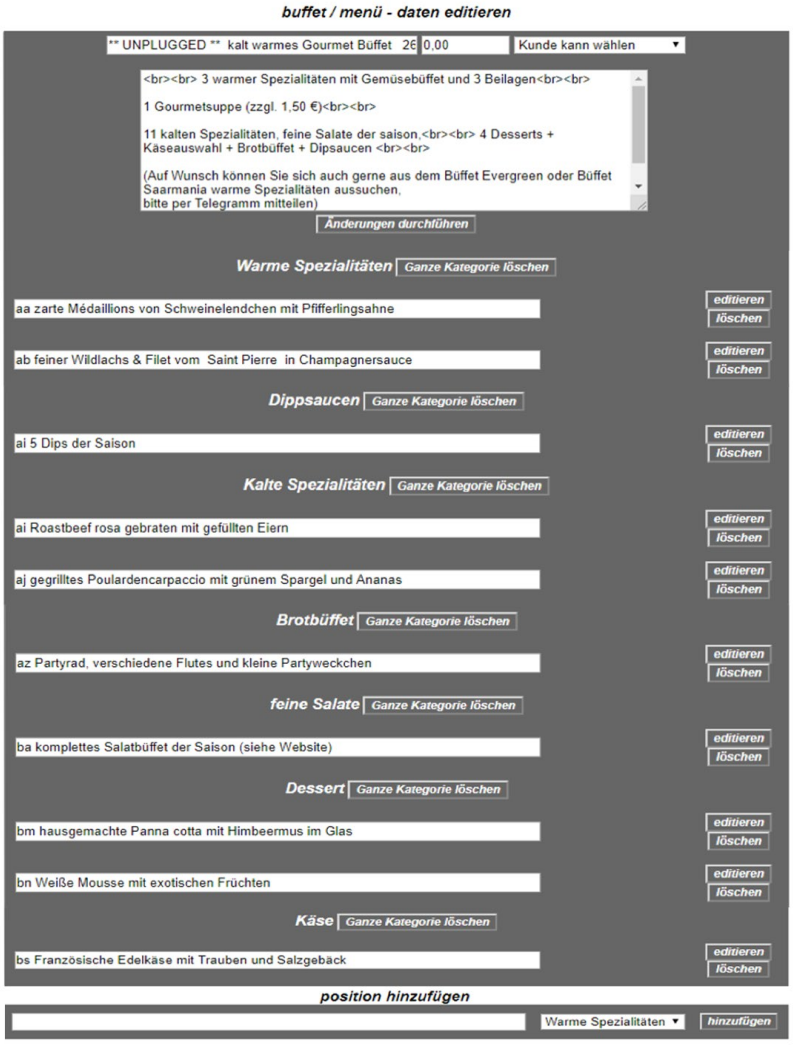

Abbildung 3.14: Die bisherige Eingabemaske für die Übersicht des Menü-Editor-Arrangements

HTTP Observatory

Hier werden aktuelle Sicherheit der Webseite im öffentlichen Internet beobachtet. Alle Anfragen sind über POST- oder GET-Methoden durchgeführt und die Rückmeldungen sind im JSON-Format. Nach dieser Analyse wird gezeigt, dass **CPS!** (**CPS!**) nicht implementiert wurde. CSP verhindert Weitbereich von Cross-Site-Scripting (XSS)- und „clickjacking"- Attacken. Cookes sind teilweise gesichert. Hier fehlt es „Secure"-Attribut, sowie „SameSite" und „Prefix". Durch „Secure" werden die Cookies davon abgehalten, dass sie über nicht gesicherten HTTP versendet werden. „SameSite" hält die Cookies vom Cross-Site-Request-Forgery (CSRF) Attacken und Cross-Site-Übersendung ab, d. h. es ist möglich, dass es einen Hacker-Angriff auf Computersystem des Benutzers durchgeführt werden [**bentley:1999**] . Es wird keine _Host- und _Secure-Prefixes auf den Namen der Cookies verwendet. Das bedeutet, dass sie nicht gegen Überschreiben geschützt sind. Man kann in der Abbildung 3.17 die genaue Bewertung von der Webseite. stellt das Resultat von der durchgeführten Analyse dar.

TLS

Die betrachtenden von uns Webseite verwendet Signaturverfahren SHA-256-With-RSA. Das bedeutet, dass eine Kombination zwischen SHA256 (kryptologi-schen Hashfunktion, die Hashwerte mit einer Länge zwischen 256 und 512 Bits erzeugt) und Rivest–Shamir–Adleman

	Artikel - Standard	Einzelpreis	
	Anlieferung - Aufschlag 2015 (siehe Website): 12,00 € zzgl. 19 % MwSt.	14,28	
			Angebot editieren

Abbildung 3.15: Die bisherige Eingabemaske für die Übersicht des Menü-Editor-Standard

Abbildung 3.16: Ergebnis vom 1zu1

(RSA) (ermöglicht symmetrische Schlüssel jeder üblichen Länge verschlüsselt zu werden). In der Abbildung 3.18 stellt dar

3.4 Warum ist eine Migration notwendig?

ASP ist eine nicht Objekt-Orientierte Skripte Sprache. Es ist anfällig zu Unter-brechung der Applikationen, z. B. wenn Internet Information Services (IIS) gestoppt und neu gestartet wird. Wenn eine ASP-Seite aufgerufen wird, wird den Text linear analysiert. Die Frontend Verwendung des Bestellsystems ist verschachtelt und es gibt redundanten Optionen. Auf einer Seite befindet sich Login- und Registrierung Menü, sowie das Bestellformular. Das verlangsamt die Eröffnung der Seite. Es gibt unnötige Quellcode, das entfernt werden kann. Ein anderer Nachteil des Systems ist, dass es nicht dynamisch orientiert ist, d.h. die Umgebung ist nicht Benutzerfreundlich. Der Auftragsgeber muss sich für jede nötige Änderung bei dem Entwickler melden. Das macht den Benutzer abhängig von dem Hersteller des Programms. Heutzutage gibt es vielfältige Auswahl von CMS, durch das der Auftragsgeber mit verbreiteten Möglichkeiten zu Änderun-gen seiner Website verfügt. Im folgenden Kapitel wird einer diesen CMS erläutert.

Test Scores

Test	Pass	Score	Explanation
Content Security Policy	✘	-25	Content Security Policy (CSP) header not implemented
Cookies	✘	-40	Session cookie set without using the `Secure` flag or set over HTTP
Cross-origin Resource Sharing	✔	0	Content is not visible via cross-origin resource sharing (CORS) files or headers
HTTP Public Key Pinning	–	0	HTTP Public Key Pinning (HPKP) header not implemented (optional)
HTTP Strict Transport Security	✘	-20	HTTP Strict Transport Security (HSTS) header not implemented
Redirection	✔	0	Initial redirection is to HTTPS on same host, final destination is HTTPS
Referrer Policy	–	0	Referrer-Policy header not implemented (optional)
Subresource Integrity	✘	-5	Subresource Integrity (SRI) not implemented, but all external scripts are loaded over HTTPS
X-Content-Type-Options	✘	-5	X-Content-Type-Options header not implemented
X-Frame-Options	✘	-20	X-Frame-Options (XFO) header not implemented
X-XSS-Protection	✘	-10	X-XSS-Protection header not implemented

Abbildung 3.17: Ergebnis vom HTTP Observatory

Certificate Information

Common name:	jungekueche.com
Alternative Names:	
First Observed:	2018-08-06 (certificate #187995580)
Valid From:	2018-07-27
Valid To:	2018-10-25
Key:	RSA 2048 bits
Issuer:	Let's Encrypt Authority X3
Signature Algorithm:	SHA256WithRSA

Abbildung 3.18: Ergebnis vom TLS Observatory

4 Anforderungsanalyse

In der durchführenden Analyse wird ein neues Bestell- und Verwaltungssystem Konzept erläutern. Auf der Basis der IST-Analyse wird eine Verbesserung mit aktuellen Webtechnologie und Verfahren durchgeführt. In diesem Kapitel wird man sich die entstehenden Probleme und Optimierungspotenzial näher anschauen.

Das neue Bestellsystem soll nicht kompliziert sein, sondern Benutzerfreundlich. Die Optionen sollen klar gezeigt werden und einfach sein. Sicherheit ist einen wichtigen Teil der guten webbasierte Entwicklung, aus diesem Grund werden zusätzliche Prüfungen in dieser Richtung entwickelt. Die Auftragsverwaltung und Kundenübersicht werden mit denselben Funktionen bleiben. Das ganze Konzept wird aus einer Umbraco-Instanz heraus verwaltet. In den folgenden Unterkapitel werden die oben beschriebene Anforderungen erweitert und erläutert.

4.1 Paket A

Aktuelle Software ergibt sich dem Auftraggeber wenige Möglichkeiten Frontend zu ändern. Es wurde immer ein Entwickler benötigt, um etwas geändert zu werden. Ein Beispiel dafür ist, dass der Auftraggeber nicht den Schrift oder die Farbe des Textes editieren konnte.

In diesem Unterkapitel wird eine volle Freiheit des Auftraggeber zur Verwendung der Tätigkeiten angefordert. Es wird besonders darauf geachtet, dass die Möglichkeiten zu ändern und editieren, erhalten bleiben. Als Beispiele wurden Schriftart und Farbe vorgegeben.

Es wird die Verwendung vom Grids, Macros und Formulare empfohlen. Diese und weitere Begriffen werden im nächsten Kapitel erläutert.

4.2 Paket B

4.2.1 Kundenverwaltung

In diesem Unterkapitel wird es mit den Anforderungen beschäftigt, die der Auftraggeber gestellt hat, um die Kundenaktivitäten besser kontrollieren zu können. Eine eigene Seite wird auch erstellt, in der Kunden Bestellungen erstellen und einsehen können oder Nachrichten verschicken oder lesen können.

4.2.1.1 Kundenerfassung

Hier wird es die Anforderungen zu den Onlinebestell-System und Kunden Übersicht beschrieben.

1. Der Kunde kann sich bei der Erstbestellung registrieren und bekommt eine PIN zugeschickt. Bei jede weitere Bestellung kann er sich mit seiner E-Mail und der PIN einloggen. Das, was beachtet werden muss ist, dass beim Bestellen die Mail geprüft werden muss und ein Hinweis ausgegeben werden soll, wenn die Mail

schon in der Datenbank bereits existiert. Die Neukunden kommen in der Bestellübersicht des Auftraggebers in einer gesonderten Übersicht und sie müssen von ihm bestätigt oder übernommen werden. Danach kann der Kunde sich einloggen. In den Abbildungen 3.2 und 3.1 sind die Register- und Anmeldeformular zu sehen. In den vorgegebenen Anforderungen ist zu diesem Punkt, nichts zu ändern. Die Funktionalitäten und Aktivitäten müssen dieselben bleiben.

4.2.1.2 Kundenansicht

1. Wenn der Kunde vom Auftraggeber verifiziert wird, kann sich mit seinen E-Mail und PIN einloggen. Er kann seine aktuelle und vergangenen Bestellungen einsehen. Ein Kontaktfenster wird verwendet, über das die Kommunikation und Absprache läuft. Der Auftraggeber kann entweder allgemeine Information seinem Kunde schreiben oder Kundenspezifisch. Die Abbildung 3.4 stellt die bisherigen Kundenansicht dar Das sind die alte Funktionalitäten. Es wird angefordert, sie nicht geändert zu werden, aber die Kunden sollen über Member-Section im Umbraco abgebildet werden.

4.2.1.3 Auftraggeber-Ansicht

1. Der Auftraggeber hat eine Kundenübersicht, die er filtern kann. In der Detailansicht sieht er die Infos zum Kunden wie Bestellverlauf und kann von der Ansicht heraus E-Mails an den Kunden senden (Mail-Vorlage oder frei gestaltbar). Anschaulicher wird im Abbildungen 3.9 und 3.10 dargestellt. In dem neuen Konzept wird angefordert, die Funktionalitäten von der alten Software gleich zu bleiben.

4.2.1.4 Kommunikation

1. Über die Option „neue Nachrichten“ kann der Auftraggeber mit seinen Kunden kommunizieren und Absprachen zu den Aufträgen treffen.

2. Für das neuen Konzept soll eine neue Kommunikationsmethode für die alte Funktionalitäten mit Filtern (gelesen, nach Kunden suchen usw.) integriert werden. Der „Chat“ sollte auch aus der Kundenkartei-Ansicht funktionieren und die Kommunikation soll an einen Auftrag gebunden sein. Die Kommunikationsübersicht wird in Abbildung 4.1 illustriert.

4.2.2 Artikelverwaltung

4.2.2.1 Artikel erfassen, ändern und löschen

1. Der Auftraggeber kann in einer recht einfachen Maske Artikel verwalten (onlineeditor). Es gibt nur zwei Kategorien: Arrangements und Artikel-Standard. Im Moment sind die Arrangements noch nicht mit den Webseiten gekoppelt.

2. Ein Arrangement besteht aus Kategorien (fest vorgegeben) und Positionen und kann sich darin unterscheiden, ob Kunden Positionen auswählen dürfen oder nicht.

3. Es wird angefordert, der Artikel einfacher verwaltet werden zu können können und die Artikelseite der Webseite soll direkt mit dem Artikel gekoppelt sein.

   Bisherige Übersicht für den Artikel ist in der Abbildungen 3.13 und 3.14 zu sehen.

Abbildung 4.1: Die bisherige Eingabemaske für die Nachricht

4.2.3 Auftragsverwaltung

4.2.3.1 Übersicht

1. Der Auftraggeber hat eine Übersicht über seine aktuellen Aufträge. Für jede Art von Auftrag (neu, bearbeitet, alte und fällig) gibt es ein Select-Befehl.

2. Diese Ansicht soll in einer eigenen Umbraco-subsection umgesetzt werden. Die Artikeldatenbank soll von Access nach SQL transportiert werden. Es muss eine neue Zuordnung Kunde zu Umbraco-Member geben. In den Abbildung 4.2 ist die Übersicht zu sehen.

Abbildung 4.2: Die bisherige Eingabemaske für die Übersicht

4.2.3.2 Detailansicht

1. In der Detailansicht kann der Auftraggeber den Auftrag bearbeiten, den Status ändern, Positionen editieren / hinzufügen und löschen, dem Kunden Freigaben erteilen (z.B. Aussuchen der Positionen) und eine Rechnungsnummer vergeben.

2. Der Auftrag muss auf vier Seiten ausdruckbar (genau vor-gegeben) sein.

3. Die Begrifflichkeiten von zweitem Blatt müssen für Auftraggeber zu Verfügung gelassen werden, damit er sie selbst ändern kann.

4. Quick-Icons erleichtern die Kommunikation mit dem Kunden und die Anpassung an dessen Auswahl zu Auftragsbeginn (z.B. Änderung Serviceauswahl).

5. Diese Ansicht soll in einer eigenen Umbraco-subsection umgesetzt werden.

   Detailansicht wird in der Abbildung 4.3 dargestellt.

Abbildung 4.3: Die bisherige Eingabemaske für die Detailansicht

4.2.4 E-Mail-Verwaltung

1. Hier kann der Auftraggeber E-Mail Vorlagen mit Platzhaltern definieren, die er dann in der Kundenkommunikation auswählen kann.

2. Es gibt ein Form für Terminanfragen über die Webseite. Diese erzeugen eine Mail an den Auftraggeber. Dieser muss in der Mail nur einen Link betätigen, um die Anfrage zu bestätigen. Dies hängt auch mit dem E-Mail Verwaltungssystem zusammen.

3. Diese Ansicht soll in einer eigenen Umbraco-subsection umgesetzt werden. Anschaulicher wird im Abbildung 4.4 dargestellt.

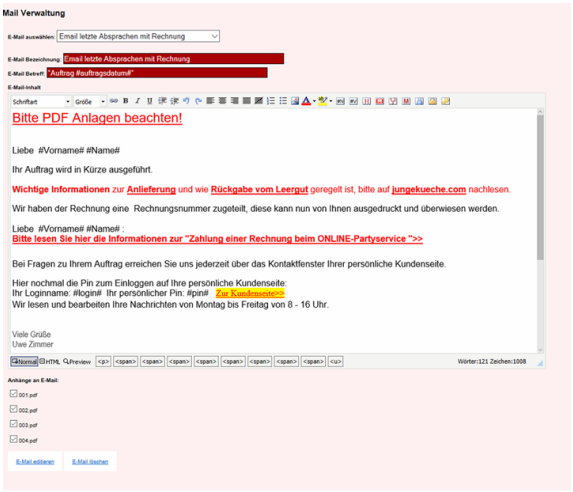

Abbildung 4.4: Die bisherige Eingabemaske für die E-Mail-Verwaltung

4.2.5 Umsatzverfassung

1. Hier kann der Auftraggeber sich die Umsätze der letzten Monate / Jahre, die über die Aufträge zustande gekommen sind, an-schauen. Dabei ist es wichtig, dass diese Datensätze nicht direkt an die Aufträge gekoppelt sind, sondern aus einer extra Tabelle kommen, die der Auftraggeber auch selbst noch editieren kann.

2. Diese Ansicht soll in einer eigenen Umbraco-subsection umgesetzt werden. In der Abbildung 4.5 wird die Umsatzerfassung dargestellt.

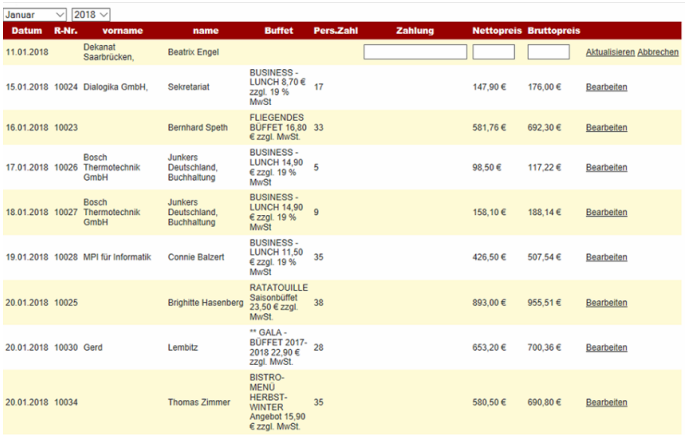

Abbildung 4.5: Die bisherige Eingabemaske für die Umsatzerfassung

5 Konzeption und Implementierung

Das neue Konzept, das in diesem Kapitel erklärt wird, beinhaltet die Migration von Datenbanken von der alten webbasierten Software zur neuen Software, die auf Umbraco CMS basiert. Die Funktionalitäten müssen gleich bleiben. Das Projekt unterteilt sich in zwei großen Pakete – Paket A und Paket B. Im ersten Paket wird die Verwaltung des Frontends aufgebaut. Das Ziel ist maximale Flexibilität für den Website-Besitzers und durch einfache Tätigkeiten, bestimmte Zwecke zu erfüllen. Man soll in diesem Bereich den Inhalt der Webseite editieren, ändern und erweitern können. Umbraco verfügt über sogenannte „Grids“. Sie dienen dazu, dass man das Design der Seite manipulieren und auch die schon besprochenen Möglichkeiten ausnutzen kann. Es werden auch eigene Makros benutzt, in den ein Quellcode der SHOP – Komponenten hingeschrieben wird. Somit kann der Auftraggeber Makros in beliebigen Teilen der Seiten einfügen. Umbraco – Forms werden als Formulare benutzt, damit man selber die Felder einordnen kann. Beide Pakete werden als Startknoten aus einer Umbraco-Instanz heraus verwaltet. Paket B ist als Kern der Seite aufgebaut. Das ist eigentlich das Online-Bestellsystem. Hier ist es wichtig, dass eine unkompliziert bedienbare, reibungslose und flexible Umgebung aufgebaut wird. Das System besteht aus drei Hauptkernen:

- Bestellsystem

  Kundenverwaltung (Anmeldung, Kundenbereich, Kommunikation)

  Artikelverwaltung

  Auftragsverwaltung

- E-Mail-Verwaltung

  E-Mail-Vorlagen anlegen, editieren, löschen

- Umsatzerfassung

  Umsatzübersicht nach Monat und Jahr

5.1 Aufbau vom Umbraco

Wie oben schon erklärt wurde, ist Umbraco [17] ein Content Management System, das flexibel und Benutzerfreundlich ist. Für ein besseres Verständnis, worum es geht, wird es in diesem Unterkapitel erläutert.

Das User Interface von Umbraco ist auf drei Teile unterteilt. Erste Teil ist Hauptfunktion bzw. Setcion. Dort befinden sich die Hauptoptionen: Content, Media. Settings, Developer, Users, Members, Forms. Damit ein klarer Unterschied zwischen „Member“ und „User“ gemacht wird, werden die beiden Begriffe erklärt. „User“ ist jemand, der Zugriff zu „Umbraco-Backoffice“ hat und dort bestimmte Rechte hat.

Ein Member wird von Umbraco für die Registrierung und Authentifizierung eines externen Besuchers benutzt. Das sind Leute, die nur Front-End benutzen dürfen. Man kann auch eine „Custom Setcion“ erstellen, womit wir uns im weiteren Kapitel beschäftigen.

Der nächste Teil ist die Unternavigation oder auch Tree genannt. Alle Unteroptionen stehen dort. Jede Hauptfunktion hat ihre eigenen Unteroptionen. Zugehörige Funktionalitäten der Unteroptionen (Trees) werden betrachtet. Die Abbildung 5.1 stellt das Funktionsprinzip von Umbraco [10].

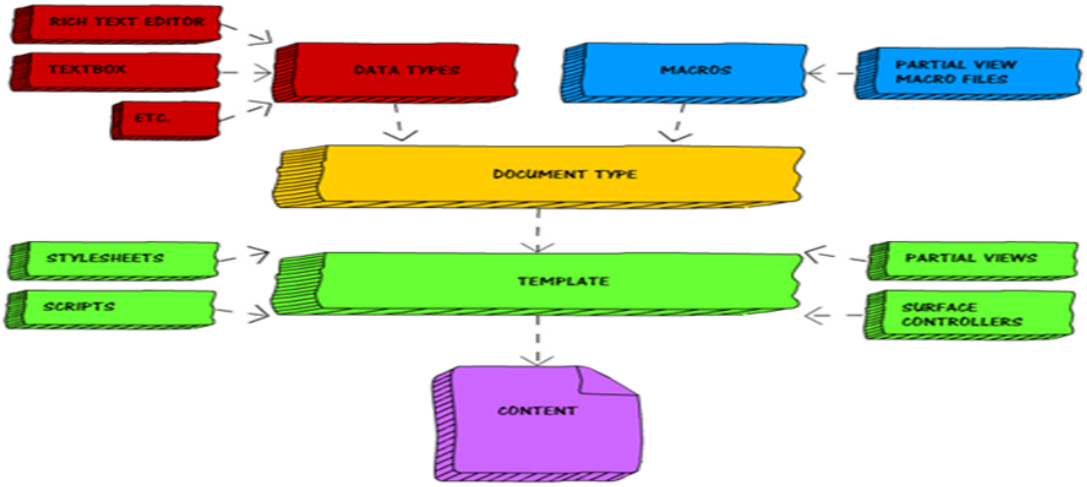

Abbildung 5.1: Funktionalität vom Umbraco-Backoffice
Quelle: http://droom.biz/umbraco-architecture-diagram/e-member-to-rule-them-all-umbraco-architecture-diagram/

- **Trees vom Content-Section:** In diesem Tree befinden sich alle Seiten, die im Website Front-End erschienen werden können. Dort steht auch Recycle Bin, oder auch Papierkorb genannt. So kann man die gelöschte Seite zurücksetzen.
- **Trees von der Media-Section:** Hier stehen alle Videos und Bilder zur Verfügung.
- **Trees vom Settings-Section:** In diesem Tree befindet sich Moglichkeiten CSS, JavaStript, Document Types und zugehörige Templates. PatialView ist auch dort. Es hat eine Übertragungsfunktion von Backend zum Frontend.
- **Trees vom Developer-Section:** Hier stehen zur Verfügung Trees, die den Entwickler ermöglichen, bereits erstellte Seiten weiter zu entwickeln. Das wird durch Data Typ, Macros, Packages, Relation Types, XSLT Files und Partial View Macro Files ermöglicht.
- **Trees vom User:**In diesem Bereich stehen die Benutzer, die mit bestimmtem Rechten Umbraco-Backend benutzen dürfen. Zu einem bestimmten User können verschiedene Rechte abgegeben werden.
- **Trees vom Member-Section:** Hier sind Benutzer, die Frontend benutzen dürfen.
- **Trees vom Forms-Section:** Hier kann man leicht verschiedene Arten von Formularen erstellen.

Der dritte Teil vom Umbraco-Backoffice ist der Editierbereich. Dort werden alle Eigenschaften von jeweiligen Tree-Optionen dargestellt.
In der Abbildung 5.2 kann man sehen, wie Umbraco-Backoffice aussieht.

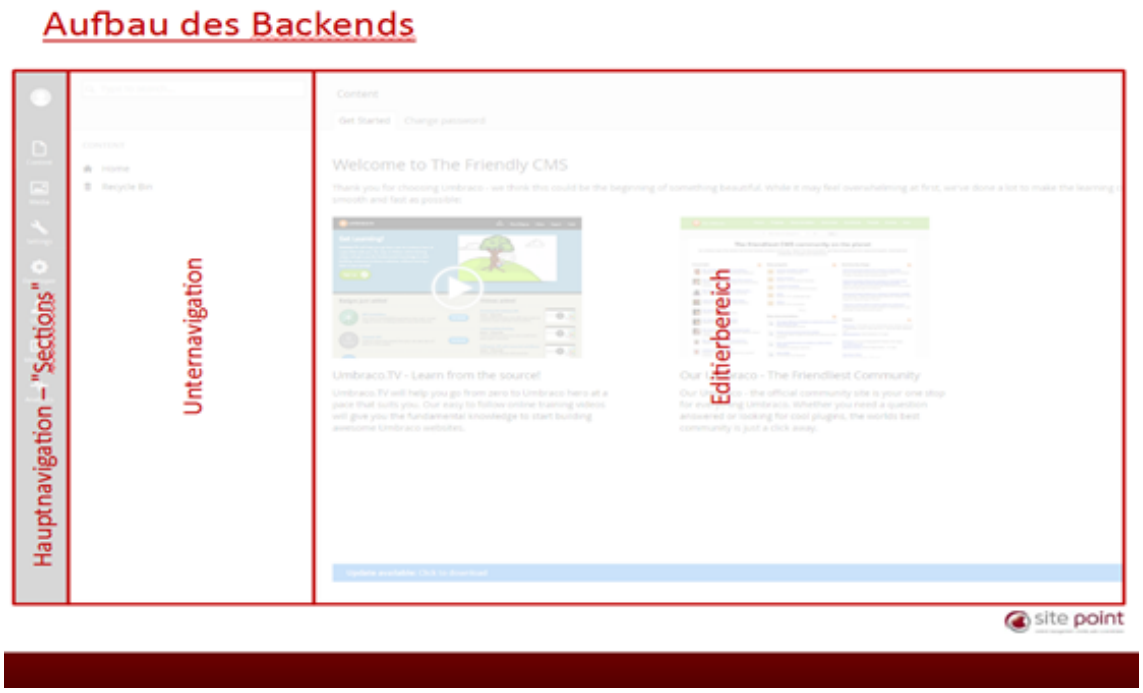

Abbildung 5.2: Übersicht vom Umbraco-Backoffice
Quelle: Script 11 von .NET Webkonzepte und Werkzeuge - Thomas Beckert

5.2 Paket A

Um der Auftraggeber mehr Flexibilität zu haben, die Frontend-Seite zu editieren, werden "Grids Rahmen verwendet. Grid enthält zwölf Spalten. Man kann diese Spalten zusammen binden. So ist Möglich, die Seite auf beliebige Teilen verteilt zu werden, wie im Abbildung 5.3 gezeigt wird.

Im Grid kann man eigene Einstellungen machen. In dieser Arbeit werden nur zwei Beispielen gezeigt, wie Grids verwendet könnten: Die Farbe und die Größe vom Schrift ändern. "Controls"wie "Rich text editor", Ïmage", "Quote", "Macro"... Für angefordertes Ziel werden Rich text editor und Macros verwendet. Man kann Macros auch im Rich text editor benutzen. Das wird im späteren Kapitel gemacht. Jetzt erden nur die schon besprochene Beispiele. Sie werden im Stylesheets unter "rteëingegeben. Im Abbildung 5.4 kann man sehen wie CSS-Befehle im Umbraco geschrieben werden können.

Über Rich text editor kann man den Text auch editieren, aber nur mit festen Optionen. Diese Stylesheet sind im Richtext edior integrierbar, somit kann Auftraggeber die Schriftart und Farbe ändern. In der Abbildung 5.5 ist der Schrift auf rote Farbe und 18pt geändert.

5.3 Paket B

5.3.1 Kundenverwaltung

Die Kundenverwaltung fasst den ganzen Bereich um, der sich um die Kunden bezieht. Mehr wird es in nächster Unterkapitel erfahren.

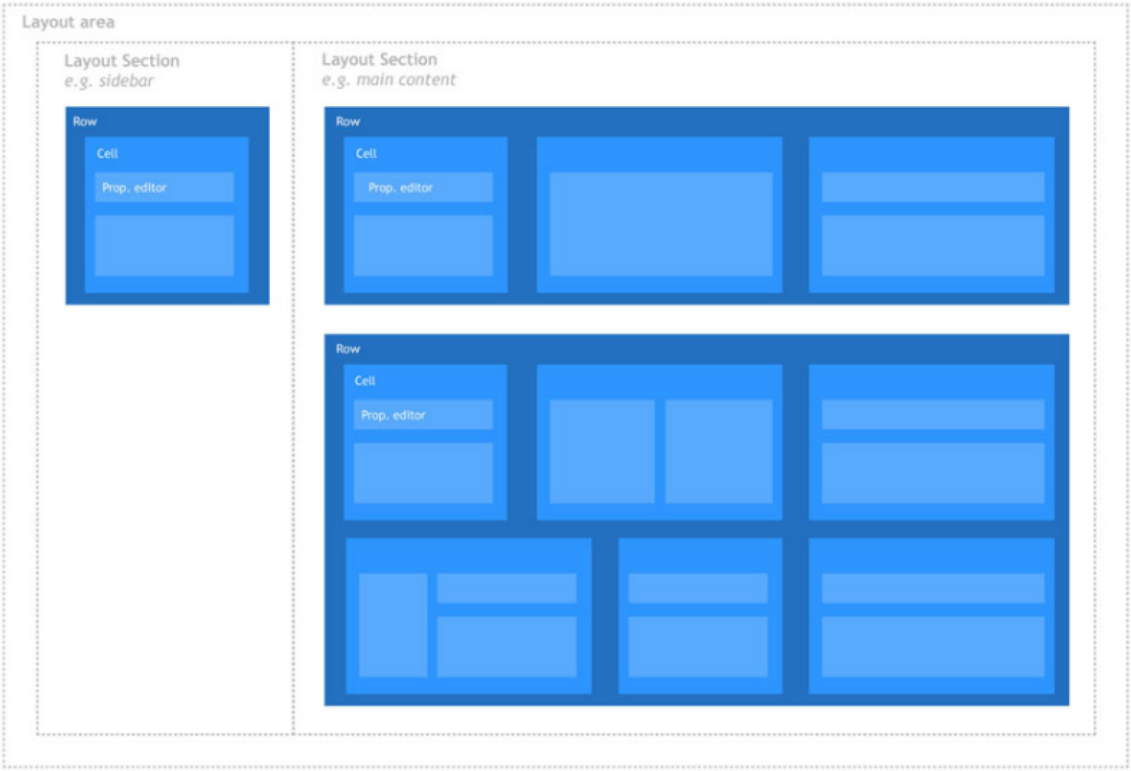

Abbildung 5.3: Übersicht vom Umbraco-Grids

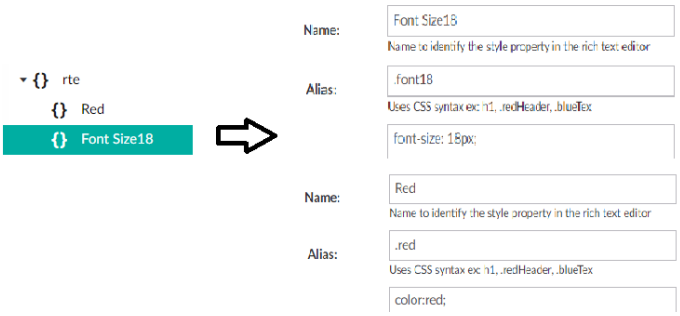

Abbildung 5.4: Styling vom Umbraco-Grids

5.3.1.1 Kundenerfassung

1. Das angestrebte Ziel ist, dass der Kunde sich registrieren kann. Die Registrierung muss gesichert und individuell sein. Damit die Registrierung zuverlässig ist, muss nach den folgenden Regeln gehalten werden: Vertraulichkeit, Integrität, Verfügbarkeit und Authentizität. Ein Personal-Identification-Number (PIN) lässt sich in E-Mail des Kunde zusenden. So wird sichergestellt, dass die Kunde eine gültige E-Mail hat. Nach der Registrierung wird überprüft, ob die E-Mail schon existiert. Wenn ja, das Prozess wird storniert und eine Nachricht erscheint, in der geschrieben ist, dass der Kunde diese E-Mail schon verwendet. Der Auftraggeber entscheidet, ob der sich registrierten Kunde bestellen darf oder nicht. Danach kann der Benutzer sich mit dem PIN und die E-Mail anmelden. Die Kundenverfassung wird durch Member Application Programming Inter-face (API) von Umbraco realisiert. Wie schon erläutert wurde, ist Umbraco auf ASP.NET basiert. Damit kann man vom CMS unterschiedliche Services benutzen. Ein davon ist „MemberService“. Das ist direkte Kanal zu dem MemberAPI. Es ist in den Services[**UmbracoHQ2018Services**] pro-

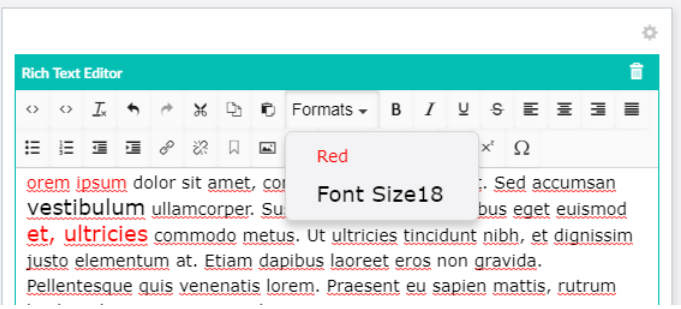

Abbildung 5.5: Styling in Umbraco-Grids - Rich text editor

perty von dem SurfaceController [13] zu Verfügung gestellt. Die Registrierung wird via ASP.Net Code und Media API programmatisch erstellt. Das ist eine komplexe Methode, in der vielfältige Dateien benötigt werden: (SurfaceController, Model[12], PartialView und Source Datei, in der View-Methoden über eine Aufruf-Funktion aufgerufen wird). Die benötige Information, die wir zu der Registrierung brauchen, wird im Model- Datei geschrieben. In der Datei Model stehen Model Properties. Das sind Parametern, mit denen man arbeitet. Einfach erklärt, über das Model werden die Properties von Partial-View zu dem SurfaceController oder umgekehrt übertragen. SurfaceController ist der „Autobahn“ zu der Umbraco-Datei. Das ist ein Model View Controller (MVC), das mit dem Umbraco interagiert wird. Es wird von der Bibliothek Umbraco.Web.Mvc.SurfaceController geerbt. Über PartialView werden die Verbindungen zwischen Kontakt Formular und Model Properties geschehen. Das ist eigentlich eine Teilansicht, die von Umbraco Frontend benutzt wird. Dort ist Umbraco User-Interface (UI). Folgende Abbildung 5.6 ergibt bessere Verständnis wie die obengenannten Begriffe zu einander stehen.

Diese Möglichkeit vom Umbraco erlaubt den Benutzer sehr leicht und bequem in dem Server anzumelden oder sich zu registrieren. In Anhang - Kundenerfassung kann man sich übersichtlicher genau anschauen, was es genau gemacht wird, damit diese Anforderung erfüllt wird.

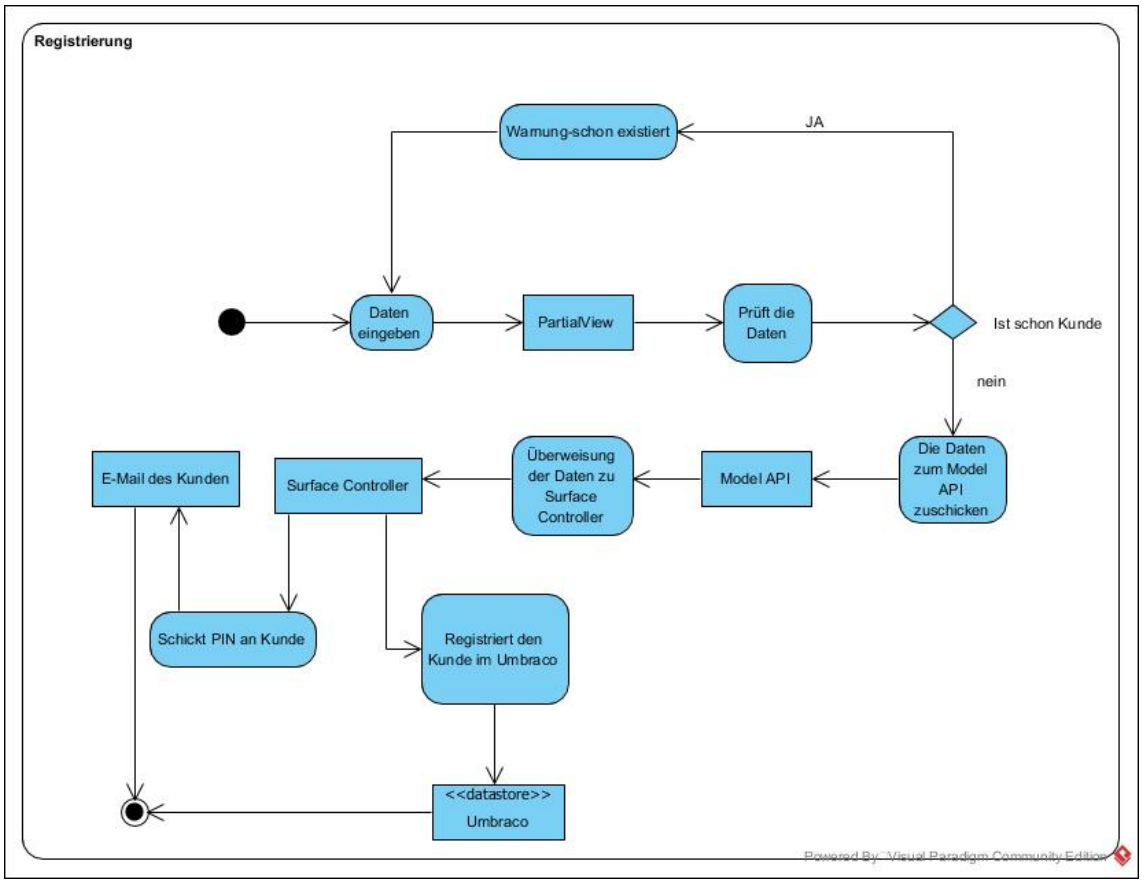

Abbildung 5.6: Neues Konzept zum Registrieren

5.3.1.2 Kundeansicht

Für Erleichterung des Kunde stehen auf einer Seite alle Möglichkeiten, die der Kunde hat:

- Neue Bestellungen abgeben, aktuelle und vorherige Bestellungen ansehen
- Neue Nachricht schreiben und alte Nachrichten ansehen.
- Wichtige Information zu beachten
- Individuelle Information vom Auftraggeber.

1. Der Kunde bekommt einen PIN an seinem E-Mail. Die Listings 5.1 und A.5 ermöglichen den PIN, obwohl er vom Kunde nicht eingegeben wurde, direkt zur Kundes E-Mail zugechiskt zu werden.

Listing 5.1: JavaScript PIN Generator

```
function myFunction() {
document.getElementById('newInput').setAttribute('Value', Math.
    floor((Math.random() * 9000) + 1000));
}window.onload = myFunction;
```

2. Nach der Registrierung sieht der Kunde eine neue Seite. Dort kann er die obengenannten Optionen verwenden. Wenn der Kunde neue Bestellung tätigen will, wird ein

neues Fenster geöffnet, in dem er erwünschten Artikeln wählen und bestellen kann. Die gewählten Produkte werden in den Datenbanken gespeichert. Von dort werden sie als vergangene Bestellungen verwendet. Das selben Prinzip steht auch für die Kommunikation zwischen den Auftraggeber und den Kunden. Wenn eine Nachricht geschrieben wird, wird sie in den anderen Datenbanken gespeichert. Vom Umbraco wird die Information direkt zu den Kunden gesendet. Der Auftraggeber, so wie der Kunde, können von der Datenbank die Bestellungen und die Nachfragen ansehen. In den weiteren Kapitel wird detailliert erklärt wie die Kommunikation und die Aufträge funktioniert. 3. Nach angefordertes Ziel wird den Kunde Profile-Seite über Member abgebildet. Das Bedeutet, dass diese Seite ist nur zum jeweiligen Kunde personalisiert. Die Abbildung wird über memberId geschehen. Es wird ein Filter erstellt, durch das diese Personalisierung erreichen wird. memberId befindet sich im umbraco-Member Datenbank. Das, was im Kundenansicht gemacht wurde, ist Verwendung von Macros und Grids. Damit man besser versteht, worum es geht, wird folgenden Abbildung 5.7 erstellt.

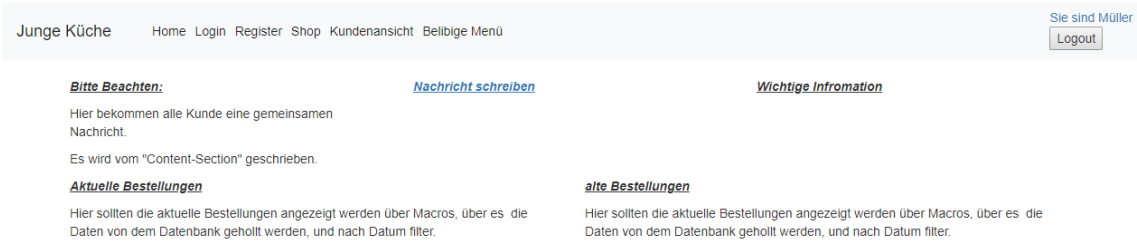

Abbildung 5.7: Kundeansicht

Über ein Macros, das man im Listing 5.2 anschauen kann, ist es Möglich die Meldungen vom Auftraggeber unter "Wichtige Informationßu sehen. Im Macro ist eine Methode schrieben, über sie wird die Übertagung vom Member-Section zum Content-Section ermöglicht wird. Abbildung 5.7 zeigt, wie der Auftraggeber zu dem Kunde eine Informationsmeldung von Member-Bereich schickt.

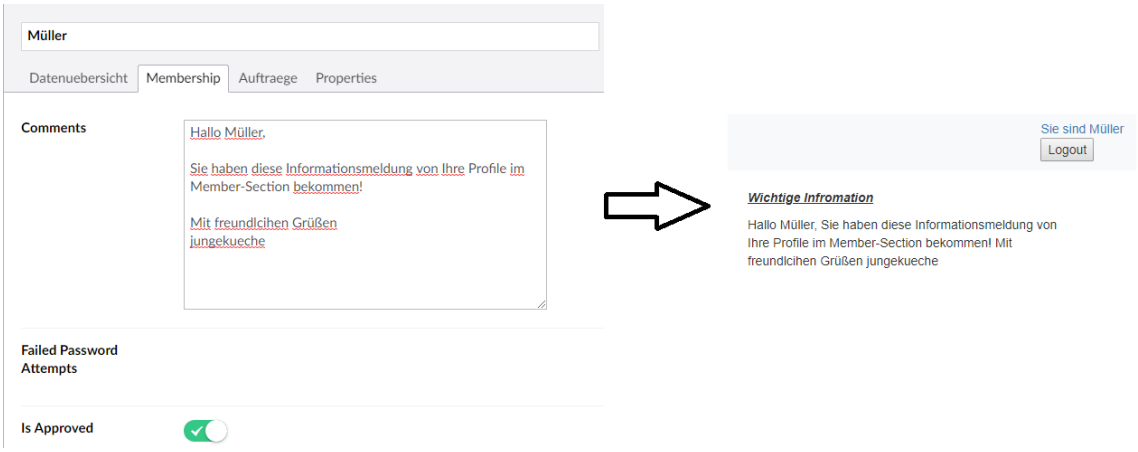

Abbildung 5.8: Übertragung der Informationsmeldung von Member-Section zu Kundenansicht

Listing 5.2: Macro zum Kundenansicht

```
@inherits Umbraco.Web.Macros.PartialViewMacroPage


@{
        var memberID = ApplicationContext.Current.Services.
           MemberService.GetByUsername(Membership.GetUser().
           UserName);
        var info = memberID.Comments;
}
@info
```

Das Kontaktfenster und allgemein die Bestellungen werden im weiteren Unterkapitel erläutert. Das gesamte Bild wird im Abbildung 5.9 dargestellt.

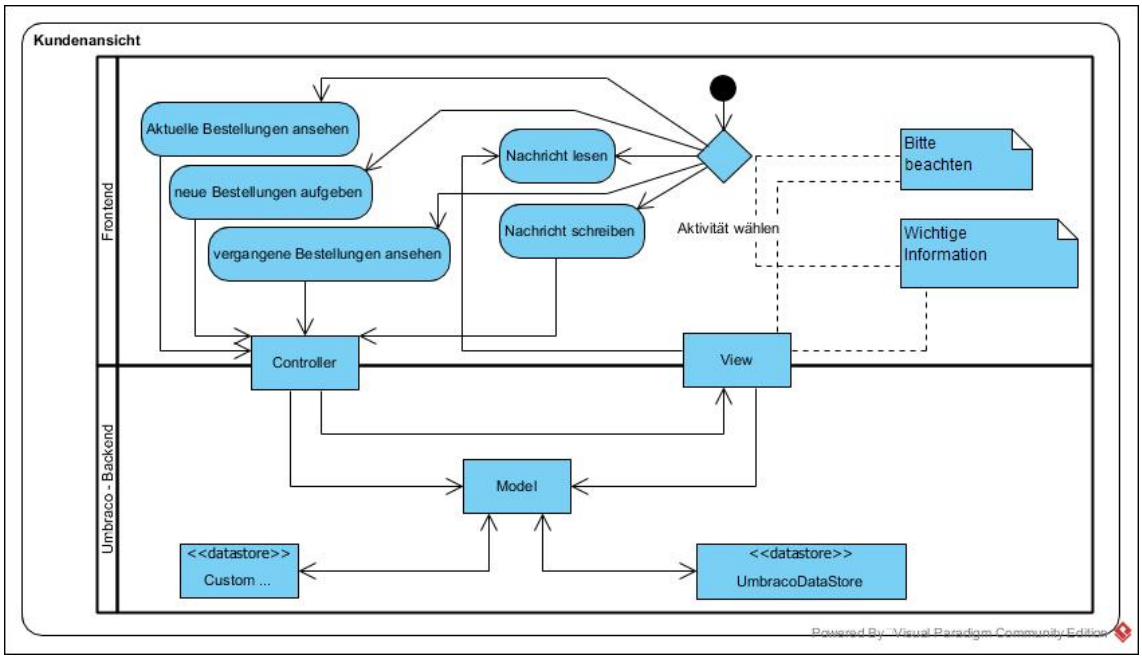

Abbildung 5.9: Funktionalität von Kundenansicht

5.3.1.3 Auftraggeber-Ansicht

1. Mithilfe vom Umbraco-Member-Section kann der Auftraggeber seine Kunde filtern. Es gibt eingebaute Einstellungen, wie ListView [14]. Durch diese Möglichkeit ist man kann die Kunden im Member-Bereich filtern, nach dem Name suchen oder löschen. Es kann auch zusätzliche Properties erstellt werden. Diese Properties können über extra Attribut im Model-Datei mit dem Frontend verbunden werden. Hier wurden neuen Tabs und Properties srtellt. Als Beispiele wird Tab-"Datenuebersicht"gegeben. Dort befindet sich nur ein Property Ört". Im Tab-Membership ist "Passwordäuch ein Beispiel, aber man kann mehrere einbauen. Abbildung 5.10 stellt dieses Tab und zugehöriges Property dar.

Wie schon erläutert wurde, ist ein Macro erstellt, durch das die Information zum jeweiligen Kunde Zugeordnet ist.

Um die Migration problemlos durchgeführt zu werden, muss man eine neue Zuordnung im Member-Bereich realisieren. Die neue Properties müssen zu den alte übertragene

Abbildung 5.10: Beispiel zur Erstellung eines extra Tab und Property

Attributen von der Member-Datenbank zustimmen. Mehr dafür wird im Unterkapitel "Übersichtërläutert.

5.3.1.4 Kommunikation

In diesem Kapitel wird beschrieben, die Kommunikation zwischen dem Auftraggeber und dem Kunde. Als Anforderung wird eine Übersichtliche Kommunikationsmethode mit Filtern (gelesen, nach Kunden suche...) festgestellt.

Es wird ContentAPI [11] verwendet. In Kundenansicht wird über ein View Nachricht-Form, damit der Kunde Nachrichten schicken kann. Mithilfe der Model und SurfaceController die Nachricht wird in der Umbraco Conten-Section. Dort sie wird als neue Untertreenode im vorgemachte Tree "Nachrichten"Dieses Tree wird mithilfe von Umbraco-ListView modelliert. Umbraco-ListView ermöglicht beliebig viele Untertreenodes, in einem einzigen Tree gelagert zu werden. Um die Nachrichten zu dem zugehörigen Kunde zu stimmen, wird es in jeweilige Nachricht das IDnummer des Kunden integriert. Mithilfe diesem Nummer wird es möglich auch ein Filter aufgebaut. Somit kann die Kunde nur seine private Nachrichten lesen. Listings 5.3 und Abbildungen 5.11 und 5.12 zeigen den ganzen Verlauf der Kommunikation.

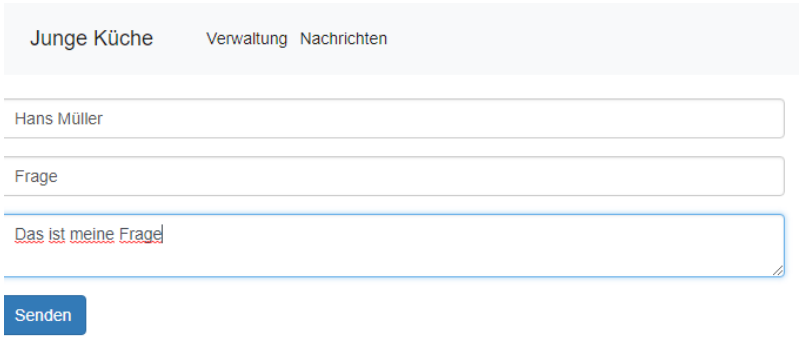

Abbildung 5.11: Nachricht Formular

Listing 5.3: NachrichController

```
namespace newKonzept.Controllers.NachrichtController
{
public class NachrichtController : SurfaceController
{
// GET: Nachricht
public ActionResult Index()
{
return PartialView("NachrichtPartial/NachrichtPartial", new
    NachrichtModel());
}
[HttpPost]
public ActionResult HandleFormSubmit(NachrichtModel model)
{
if (!ModelState.IsValid)
{
return CurrentUmbracoPage();
}

var memberID = ApplicationContext.Current.Services.MemberService.
    GetByUsername(Membership.GetUser().UserName);
model.SenderId = memberID.Id;
var comment = Services.ContentService.CreateContentWithIdentity(
    model.Sender, CurrentPage.Id, "nachricht");

comment.SetValue("Betreff", model.Betreff);
comment.SetValue("Sender", model.Sender);
comment.SetValue("Message", model.Message);
comment.SetValue("SenderId", model.SenderId);

Services.ContentService.Save(comment);

Services.ContentService.Save(comment);
TempData["success"] = true;

return RedirectToCurrentUmbracoPage();
}
}
}
```

Wenn die Nachricht nicht gelesen ist, ist durchsichtig. In dem geöffneten Fenster, steht die gesendete Nachricht. Neben dran ist ÄntwortTab. Somit kann der Auftraggeber die Frage beantworten. Die Nachrichten werden nach memberID gefiltert. In Abbildung 5.13 und 5.14 wird die Rückfrage dargestellt.

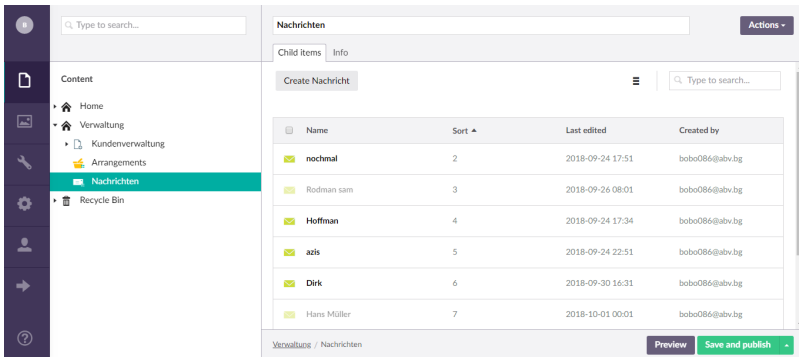

Abbildung 5.12: Nachricht in Umbraco-Backoffice

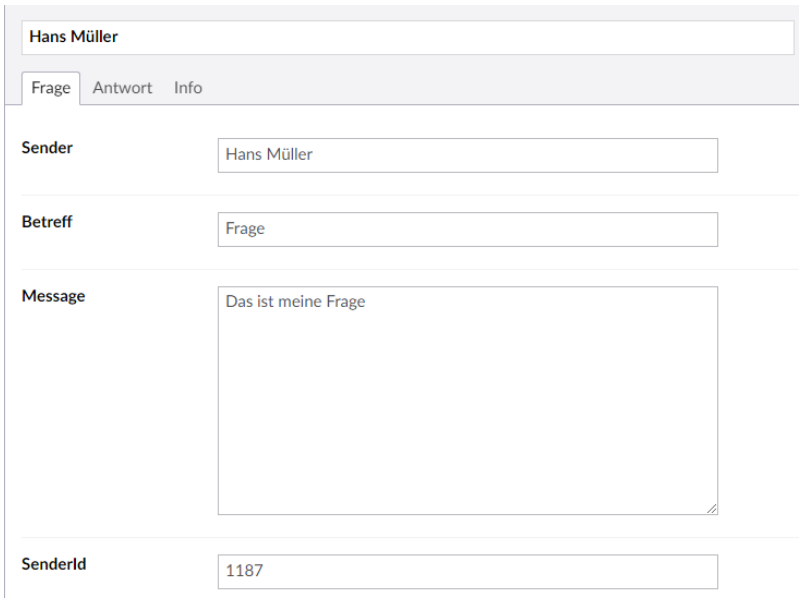

Abbildung 5.13: Die Nachricht wurde bekommen

In Listing ist das Filter, das in einem Macro integriert wird, dargestellt.

Listing 5.4: NachrichFilter

```
<ul>

@foreach(var item in selection){

var pageId = item.GetPropertyValue
<string>
("senderId");
if(pageId.ToString() == memberID.Id.ToString()){
<li>
<a href="@item.Url">@item.Name</a>
</li>
}
}
</ul>
```

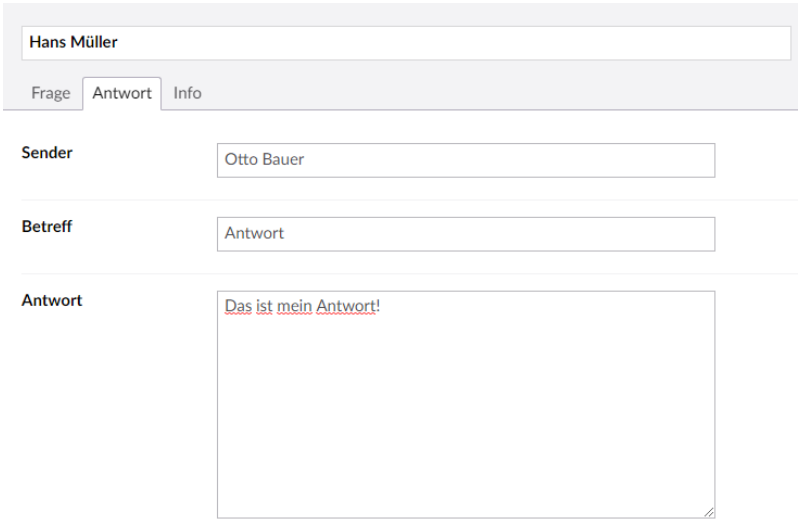

Abbildung 5.14: Antwort Schreiben

Der Kunde sieht die Antwort von dem Auftraggeber, wie in der Abbildungen 5.15 und 5.16 gezeigt wird.

Abbildung 5.15: Antwort bekommen

Abbildung 5.16: Alle Nachrichten

5.3.2 Artikelverwaltung

5.3.2.1 Erfassen, editieren und löschen

In der vorgegebenen Aufgabe muss den Online-Editor mit zwei Kategorien sein - Arrangements und Artikel-Standard. Diese müssen zu den zugehörigen Webseiten gekoppelt werden. Als Anforderung steht, dass Artikel-Verwaltung einfacher verwaltet werden kann und die Artikelseite der Webseite soll direkt mit dem Artikel gekoppelt sein. In der alten Seite ist die Artikel-Verwaltung nicht mit den zugehörigen Seite gekoppelt. Migration ist nicht angefordert.

Das betrachtete Ziel wird über Macros, PetaPoco-Datenbank [8] und Umbraco vorgegebene Funktionen. Zuerst wird Umraco ListView verwendet, damit die erstellte Artikel sich im Umbraco-Content-Section unter einer gemeinsamen Unteroption befinden. Dazu ist auch vorgegeben, dass den neuen erstellten Artikel direkt editiert werden kann. Abbildung 5.17 zeigt übersichtlicher wie die Artikel-Verwaltung aussieht.

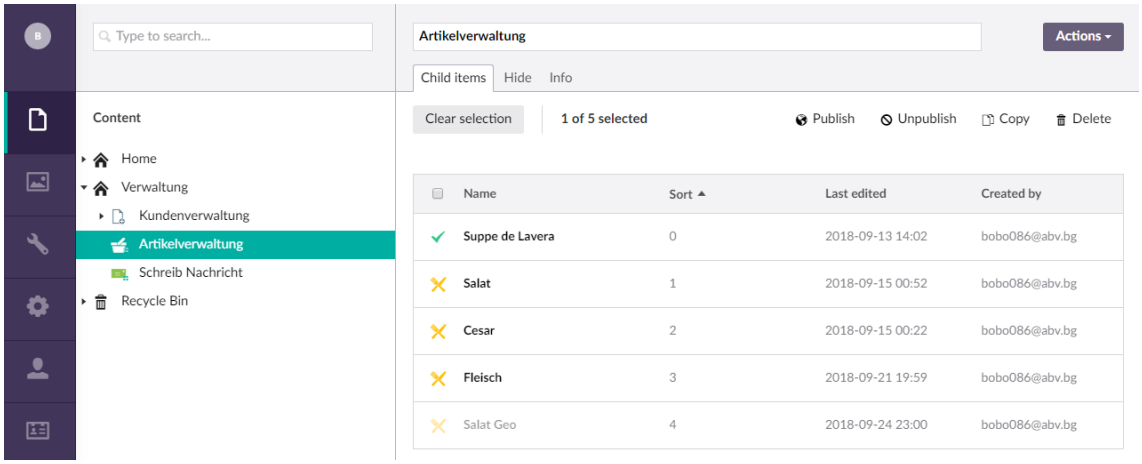

Abbildung 5.17: Übersicht der Artikel-Verwaltung

Wenn der Artikel ünpublishïst, kann er nicht im Frontend verwendet wird. Wenn die Artikel schon erstellt sind, wird ein Quellcode im Macros programmiert. Diese Macros werden im schon besprochenen Grids integriert, über die die Inhalt der Artikel zum Frontend übertragen wird. Die Inhalt wird nicht nur mit den Webseiten, sonder auch mit dem Shop-Menü gekoppelt. So wird Selbständigkeit des Auftraggeber verbessert. In der nächsten Abbildung wird gezeigt, wie mit der Erstellung von einem Artikel, werden seinen Daten sowohl im zugehörige Seite, als auch in der Shop-Menü gekoppelt.

Wenn der Kunde registriert ist, kann er die von dem Auftraggeber eingegebene Artikel aus dem Shop wählen. Nach der Bestellung werden die Daten in der Datenbank Ärtikel"gespeichert. Gleichzeitig werden diese Daten zusammen mit dem Kunden Daten auch im Datenbank Äuftraege"gespeichert. Aus Abbildung 5.18 wird gesehen, dass die Daten von der Artikelverwaltung wurden erfolgreich zu dem Shop versendet.

5.3.3 Auftragsverwaltung

5.3.3.1 Übersicht

Das vorgegebene Ziel ist eine Übersicht über die Aufträge und Nachrichten. Diese Ansicht soll in einer eigenen Umbraco-Section umgesetzt werden. Die Artikeldatenbank soll von Access nach SQL transportiert werden. Es muss eine neue Zuordnung zu Umbraco-Member geben.

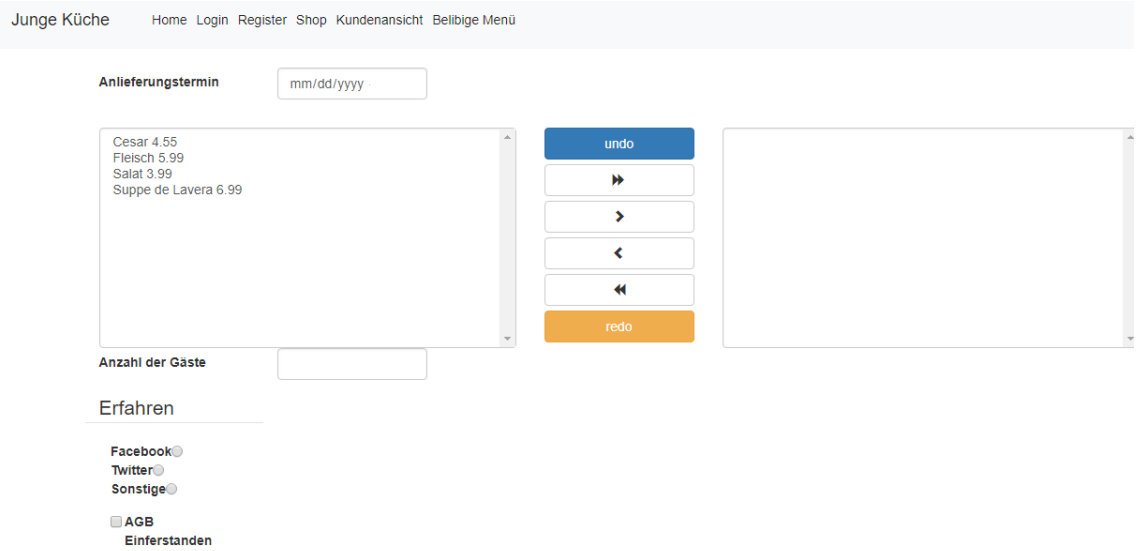

Abbildung 5.18: Übersicht des Shops

Das vorgeschlagene Konzept umfasst die Migration von Access Datenbankdatei und Erstellung von neue Custom-Section.

1. Um ein neues Section aufgebaut zu werden, braucht man zuerst in einer Klasse-Datei, Methoden von den Bibliotheken ümbraco.buisnesslogicünd ümbraco.interfacesäufzurufen. Über ÄpplicationMethode und ÏApplicationInterface. Somit wird die Custom-Section erstellt. Wenn das fertig ist, wird die Übersicht über AngularJS-Controller und HTML erstellt. Diese Dateien werden zu dem API-Controller über Paket-Manifest referenziert. In diesem Controller werden die eigentliche Funktionalitäten von Custom-Section realisiert. Mithilfe dem API-UmbracoAuthorizedJsonController werden die Daten über Model-Attributen von schon erstellter Datenbank Äuftragеßur Verfügung gestanden, die schon im vorherigen Unterkapitel erwähnt wurde. Die Datenbanken werden über PetaPoco erstellt. In der Abbildung 5.19 stellt die Kommunikation dar.

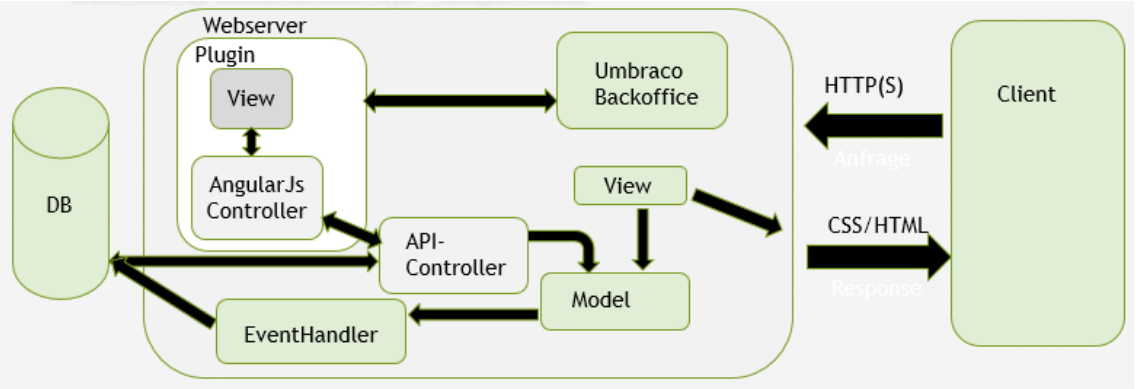

Abbildung 5.19: Eine Übersicht von Kommunikation zwischen verschiedene Elemente

2. Wie es angefordert wurde, soll eine Migration von Access-Datenbank zu Umbraco durchgeführt werden. Das angestrebte Ziel ist, dass die alten Daten in der neuen zugehörigen Datenbank gespeichert werden. Die neue betrachtete Datenbanken sind Custom-Ärtikelünd Ümbraco-Member". Nach tiefer Recherche wird ein Konzept erstellt. Grundsätzlich stehen im Umbraco Member-Section [7] drei Unteroptionen - Members, Member Types und Member Groups. Im Member Type wird die Einstellungen (Properties)

des Members aufgebaut. Dort wird eine neue Member Type mit extra Properties erstellt. Folgendes Konzept ist basiert auf der Erwartung, dass die Datenbanken von Access auf CSV-Format konvertiert sind. Das Bedeutet, dass die Accsess-Datenbankformat Datei auf einer Text-Datei umgewandelt wird. Somit kann man die Methode StreamReader nutzen. Um das vorgegebene Ziel realisiert zu werden, braucht man, ein SurficeController zu erstellen, das die Übertragung leitet. Umbraco verfügt mit spezielle Äutobannen", durch die möglich ist die Manipulation von festgelegte Sections im Umbraco. Man nennt diese Äutobannen SServices". Sie werden über ApplicationContext aufgerufen. In diesem Fall wird "MemberService"genutzt. Damit die Datei gelesen wird, wird die Funktion StreamReader verwendet. Somit wird die gelesene Datei in einer Variablen gespeichert. Von dieser Variablen wird mithilfe von dem Befehl-split in einem Array gespeichert. Jeder Teil von dem Array wird über MemberService Befehle und Methoden im Umbraco Members gespeichert. Im Anhang A.6 steht das Quellcode, um man besseres Vorbild zu haben, sowie in der Abbildung 5.20 für eine klare Übersicht der Migration dargestellt wird.

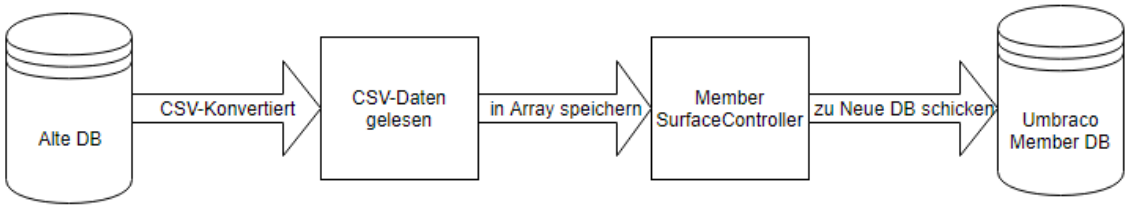

Abbildung 5.20: Eine Übersicht von Kommunikation zwischen verschiedene Elemente

Wenn die Kundendaten schon übertragen wurden, kann man das selben Prinzip der Migration für die Artikel benutzen. Hier wird als Ziel statt MemberServices - Datenbank Artikel-Datenbank verwendet. Nach der Umwandlung von Access-Datenbankdatei auf CSV-Format, wird diese Datei über die Methode StreamReader, danach ist die Information im ein Array gespeichert. Über API-Controller werden die Daten im ÄrtikelDatenbank übertragen. In Listing 5.5 wird anschaulicher gesehen, wie läuft die Übertragung der Daten durch.

Listing 5.5: Datenbank Ärtikel Transfer

```
var setArtikel = new Artikel();
var db = new PetaPoco.Database("umbracoDbDSN");
setArtikel.kundeID =_split[0];
setArtikel.bezeichnung = _split[1];
setArtikel.beschreibung = _split[2];
setArtikel.preis = _split[3];
setArtikel.art = _split[4];
setArtikel.kannWaehelen = m_split[5];

db.Insert(setArtikel);
```

5.3.3.2 Detailansicht

Hier wird angefordert, dass der Auftraggeber die Aufträge bearbeiten kann, den Status ändern, Positionen editieren, hinzufügen oder löschen, dem Kunden Freigaben erteilen (zum Beispiel Aussuchen der Positionen) und ein Rechnungsnummer vergeben. Diese

Aktivitäten sind gleich mit der alten Webseite und müssen in eigenem Umbraco-Section umbesetzt werden.

Die Erstellung von dem Custom-Section ist bereits bekannt. Hier wird es mit den Auftraege- und Member-Datenbank über AngularJS-Controller anhand des Package-Manifest zu dem API-UmbracoAuthorizedJsonController verbunden. In Listings 5.6 und 5.7 wird gezeigt wie die Daten von der Datenbank aufgerufen werden.

Listing 5.6: AngularJS-Controller ruft die zugeordnete Methoden im API-Controller

```
angular.module("umbraco").controller("Detailansicht.editController
   ", function ($scope, $routeParams, auftragResource,
   navigationService, notificationsService) {

        $scope.loaded = false;

        // Inhalt wird geladen
        if ($routeParams.id == -1) {
                $scope.auftrag = {};
                $scope.loaded = true;
        }
        else {
        // Artikel wird geladen
                auftragResource.getById($routeParams.id).then(
                   function (response) {
                $scope.auftrag = response.data;
                $scope.loaded = true;
                });
        }

        $scope.save = function (auftrag) {
                auftragResource.save(auftrag).then(function (
                   response) {
                // $scope.auftrag.$isdirty = false;
                        $scope.auftrag = response.data;
                        notificationsService.success("Success",
                           auftrag.auftragId + " " + " has been
                           saved.");
                        navigationService.syncTree({ tree: '
                           auftragTree', path: [-1, $scope.id],
                           forceReload: true }).then(function (
                           syncArgs) {
                                navigationService.reloadNode(
                                   syncArgs.node);
                                });
                        });
        }
});
```

Listing 5.7: API-UmbracoAuthorizedJsonController

```
namespace newKonzept.Controllers.DB
{
        [PluginController("Detailansicht")]
        public class AuftragApiController :
           UmbracoAuthorizedJsonController
        {
                private Database db = new Database("umbracoDbDSN");
                public IEnumerable<Auftraege> GetAll()
                {
                        List<Auftraege> liste = new List<Auftraege
                           >();
                        foreach (Auftraege auftrag in db.Query<
                           Auftraege>("SELECT * FROM Auftraege"))
                        {
                                liste.Add(auftrag);
                        }
                        return liste;
                }
        public Auftraege GetById(int id)
        {
                return db.SingleOrDefault<Auftraege>("SELECT * FROM
                     Auftraege WHERE auftragId=@0", id);
        }

        public Auftraege PostSave(Auftraege auftrag)
        {
                if (ModelState.IsValid)
                {
                        if (auftrag.auftragId > 0)
                        {
                                db.Update(auftrag);
                        }
                        else
                        {
                                db.Insert(auftrag);
                        }
                }
                return auftrag;
        }

        public int DeleteById(int id)
        {
                var db = new PetaPoco.Database("umbracoDbDSN");
                return db.Delete<Auftraege>("WHERE auftragId=@0",
                   id);
        }
}
```

Die Daten von Member-Datenbank werden als String über MemberService aufgerufen und in ein Array gespeichert. Dieses Array wird von der AngularJS aufgerufen und zu HTML-Plugin-Editor zugeschickt. Die Daten des Members können in der Detailansicht nicht geändert werden, deswegen werden sie als String verwendet.

5.3.4 E-Mail Verwaltung

5.3.5 Umsatzverwaltung

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde eine veraltete webbasierte Bestellsystem. Nach der Analyse wurde festgestellt, dass eine neues Konzept aufgebaut werden muss, damit eine Migration der Software geschehen werden kann. Die Anforderungen wurden festgelegt und wurde Ubraco CMS als Migrationsziel entschieden. Nach tiefe Recherche, Forschungen und Tests wurden die Konzeption und Implementierung erstellt. Während Arbeitsverlauf wurden viele Quellen genutzt. Das Konzept wurde Teilweise beendet. Zwei Unterkapitel wurden nicht erläutert, wegen weniger Information und Zeitaufwand. Vorteile nach der Migration wurde erreicht. Der Auftraggeber hat verbreitete Möglichkeit und Flexibilität für die Verwaltung des Frontends. Kundenerfassung wurde erfolgreich bestanden. Es wurde ein Missverständnis zu Kommunikation erschient. Deswegen wurde die Kommunikationskonzept zurückgewiesen. Die Daten des Kunden wurden erfolgreich im Member-Section von Umbraco gespeichert. Dort kann der Auftraggeber die Kundendaten editieren oder löschen. Die Möglichkeit, die dem Auftraggeber ergibt, private Information zum Kunde zu senden, wurde eben bestanden. Aufgrund nicht verbundenen Datenbanken funktioniert die Artikelverwaltung, Übersicht und Detailansicht nicht. Das Konzept zur Migration wurde theoretisch erstellt. Das Prozess wird durch Konvertierung der Datenbankdateien auf CSV-Datei realisiert und anhand SurfaceControllers in der neuen Datenbank übertragen. Es wird empfohlen weitere Testen der Software und Verbesserungen für Kommunikation, Datenbankverbindungen und Artikelverwaltung.

Literatur

[1] 1and1. *Web-Cheker*. 1and1. Juli 2018. URL: https://www.1and1.com/domain-check.

[2] Mary Delamater und Zak Ruvalcaba. *murach's Javascript and jQuery*. Mike Murach und Associates, 2017.

[3] Jon Duckett. *HTML and CSS Desigt and build websites*. John Wiley und Sons, 2011.

[4] David Flanagan. *JavaScript*. OReilly, 2011.

[5] Minko Gechev. *Switching to Angular*. Packt Publishing, 2017.

[6] April King. *HTTP Observatory Scoring Methodology*. Aug. 2018. URL: https://github.com/mozilla/http-observatory/blob/master/httpobs/docs/scoring.md.

[7] OurUmbraco. *Members*. UmbracoHQ. Sep. 2018. URL: https://our.umbraco.com/documentation/getting-started/data/members/.

[8] Brad Robinson. *PetaPoco*. Sep. 2018. URL: https://github.com/CollaboratingPlatypus/PetaPoco.

[9] SitePoint. *SitePoint*. Juli 2018. URL: https://www.sitepoint.de/.

[10] UmbracoHQ. *Backoffice overview*. Sep. 2018. URL: https://our.umbraco.com/documentation/Getting-Started/Backoffice/.

[11] UmbracoHQ. *Content*. Sep. 2018. URL: https://our.umbraco.com/Documentation/Reference/Management/Models/Content.

[12] UmbracoHQ. *Models*. Sep. 2018. URL: https://our.umbraco.com/documentation/Reference/Management/Models/.

[13] UmbracoHQ. *SurfaceController*. Sep. 2018. URL: https://our.umbraco.com/Documentation/Reference/Routing/surface-controllers.

[14] UmbracoTV. *ListView*. Sep. 2018. URL: https://umbraco.tv/videos/umbraco-v7/implementor/fundamentals/document-types/list-view-and-templates-menu/.

[15] UmracoHQ. *Umbraco*. UmbracoHQ. Juli 2018. URL: https://umbraco.com/.

[16] Cristian Wagner. *Model-Driven Software Mogration: A Methodology*. Springer Viewweg, 2014.

[17] Nik Wahlberg und Paul Sterling. *Umbraco User's Guide*. Wiley Publishing, 2011.

[18] Charles Wyke-Smith. *Stylin with CSS A Disigner's Guide*. New Rider, 2013.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

EDV Elektronische Datenverarbeitung

WIC Windows Imaging Components

ASP Active Server Pages

HTML Hypertext Markup Language

WWW World Wide Web

GUI Graphical User Interface

CSS Cascading Style Sheets

CMS Content Management System

HTTP Hypertext Transfer Protocol

TLS Transport Layer Security

SSH Secure Shell

SSL Secure Sockets Layer

XSS Cross-Site-Scripting

CSRF Cross-Site-Request-Forgery

RSA Rivest–Shamir–Adleman

IIS Internet Information Services

PIN Personal-Identification-Number

API Application Programming Inter-face

MVC Model View Controller

UI User-Interface

Anhang

A Anhang -Kundenverwaltung

A.1 Kundenerfassung

Hier kann man besser sehen wie die Umsetzungsvorgabe erfüllt wurde.

Im Model werden die Attributen erstellt, die von den Cotroller und PartialView benutzt werden.

Listing A.1: RegisterModel

```
using System.ComponentModel.DataAnnotations;

namespace newKonzept.Models.Register
{
        public class RegisterModell
        {
                [Required]
                public string Name { get; set; }
                [Required]
                public string Vornane { get; set; }
                [Required]
                public string Ort { get; set; }
                [Required]
                [EmailAddress]
                public string Email { get; set; }
                public string Password { get; set; }
        }
}
```

Im "PartialView"werden vom Model und PIN JavaScript-Funktion-Generator angegeben, die Attribute von Model.

Listing A.2: RegisterView

```
@model newKonzept.Models.Register.RegisterModell
@using newKonzept.Controllers.Register;


@if (TempData["success"] == null)
{
        using (Html.BeginUmbracoForm<CheckIsApprovedController>("
            HandleFormSubmit"))
        {
                using (Html.BeginUmbracoForm<RegisterController>("
                    Register"))
                {

                        @Html.AntiForgeryToken()
```

```
@Html.ValidationSummary(true)

<fieldset>
<div class="row form-group">
<div class="col-md-6">
<!-- <label for="lname">Last Name</label>
    -->
@Html.TextBoxFor(m => m.Name, new { @class
    = "form-control", placeholder = "Name"
    })
@Html.ValidationMessageFor(m => m.Name)

</div>
</div>

<div class="row form-group">
<div class="col-md-6">
<!-- <label for="lname">Last Name</label>
    -->
@Html.TextBoxFor(m => m.Vornane, new {
    @class = "form-control", placeholder = "
    Vorname" })
@Html.ValidationMessageFor(m => m.Vornane)

</div>
</div>

<div class="row form-group">
<div class="col-md-6">
<!-- <label for="email">Email</label> -->
@Html.TextBoxFor(m => m.Email, new { @class
     = "form-control", placeholder = "Email"
     })
@Html.ValidationMessageFor(m => m.Email)
</div>
</div>
<div class="row form-group">
<div class="col-md-6">
<!-- <label for="lname">Last Name</label>
    -->
@Html.TextBoxFor(m => m.Ort, new { @class =
     "form-control", placeholder = "Ort" })
@Html.ValidationMessageFor(m => m.Ort)

</div>
</div>
@Html.HiddenFor(m => m.Password, new { id =
     "newInput", Value = "123" })
@Html.ValidationMessageFor(m => m.Password)
@section Scripts{

        <script type="text/javascript" src
            ="~/Scripts/randomNr.js"></
            script>
}

<div class="form-group">
```

```
                        <button type="submit" class="btn btn-
                            primary">Register</button>
                        </div>


                        </fieldset>

                    }

            }
}
else
{
            <p>Sie haben sich erfolreich registriert!</p>
}
```

PIN JavaScript-Generator. der PIN wird werden zwischen den Zahlen 1000 und 9999 generiert.

Listing A.3: PIN-Generator

```
function myFunction() {
        document.getElementById('newInput').setAttribute('Value',
            Math.floor((Math.random() * 9000) + 1000));
}window.onload = myFunction;
```

Der Kunde gibt seine Daten ein.

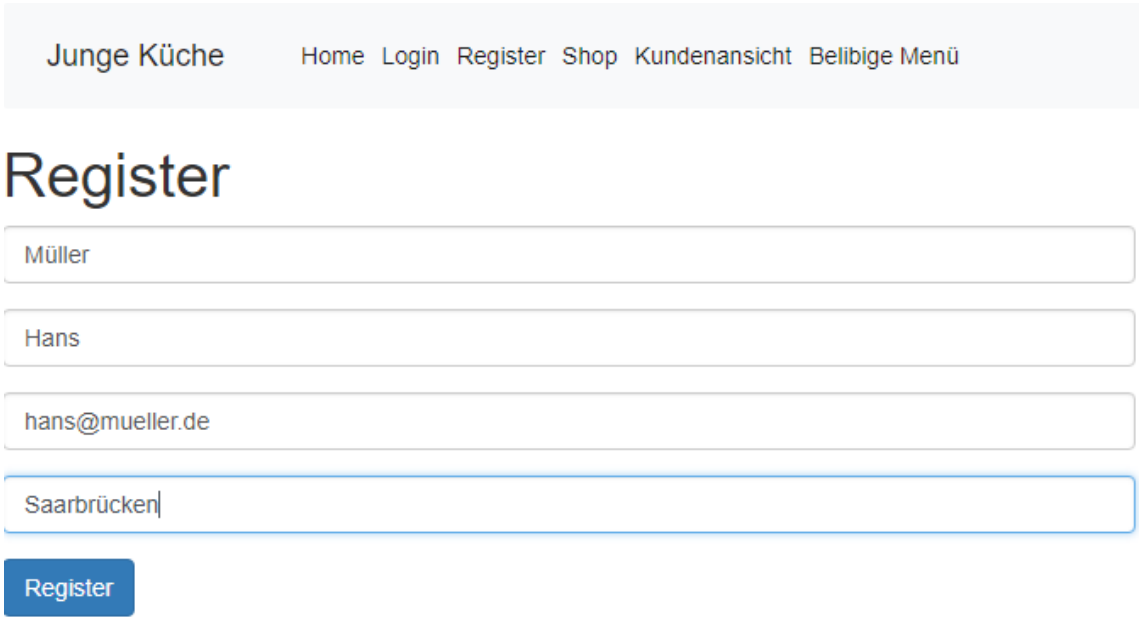

Abbildung A.1: Der Kunde gibt seine Daten ein.

Die eingegebene Daten werden über SurfaceController zum Umbraco Databank zu geschickt.

Listing A.4: RegisterConsontroller

```
using System.Web.Mvc;
using Umbraco.Web.Mvc;
using newKonzept.Models.Register;

namespace newKonzept.Controllers.Register
{
        public class RegisterController : SurfaceController
        {
        // GET: Register
                public ActionResult Register(RegisterModell model)
                {
                        if (!ModelState.IsValid)
                        {
                                return CurrentUmbracoPage();
                        }

                        var memberService = Services.MemberService;

                        if (memberService.GetByEmail(model.Email)
                           != null)
                        {
                                ModelState.AddModelError("", "
                                   Mietglider/in mit diesem Konte
                                   schon existiert!");
                                return CurrentUmbracoPage();
                        }

                        //Schick die Information an Umbraco
                        var member = memberService.
                           CreateMemberWithIdentity(model.Email,
                           model.Email, model.Name, "neueKunden");
                        member.SetValue("ort", model.Ort);
                        member.SetValue("password", model.Password)
                           ;
                        member.IsApproved = false;
                        memberService.AssignRole(member.Username, "
                           Normal");
                        memberService.SavePassword(member, model.
                           Password);
                        Members.Login(model.Email, model.Password);
                        memberService.Save(member);

                        return Redirect("/");
                }
        }
}
```

Der Kunde ist mit automatisch generierter PIN im Umbraco Member-Section gespeichert. Jetzt wartet der Kunde auf die Bestätigung.

A.2 Kundenansicht

Listing A.5: Class E-Mail zum Kunde schicken

```
namespace newKonzept.Controllers.Register
{
public class CheckIsApprovedController : SurfaceController
{
// GET: CheckIsApproved

public ActionResult Index()
{
return PartialView("RegisterPartial", new RegisterModell());
}

[HttpPost]
public ActionResult HandleFormSubmit(RegisterModell model)
{
void MemberService_Saving(IMemberService sender, SaveEventArgs<
    IMember> e)
{
foreach (IMember umbMember in e.SavedEntities)
{

if (!umbMember.IsApproved)
{
continue;
}

bool oldValue = ApplicationContext.Current.Services.MemberService.
    GetById(umbMember.Id).IsApproved;

if (oldValue != umbMember.IsApproved)
{
MailMessage message = new MailMessage();
message.To.Add(model.Email);
message.Subject = "PIN";
message.From = new System.Net.Mail.MailAddress("office@jungekueche.
    com");
message.Body = model.Password;
SmtpClient smtp = new SmtpClient();
smtp.Send(message);
}
}
}
return RedirectToCurrentUmbracoPage();
}
}
}
```

A.3 Anhang - Auftraggeber-Ansicht

Listing A.6: Migrationsclass

```
using System;
using System.IO;
using System.Web.Mvc;
using Umbraco.Core.Models;
using Umbraco.Web.Mvc;

namespace newKonzept.Controllers.MemberController
{
public class MemberController : SurfaceController
{
// GET: Member
public ActionResult ImportMembers()
{
string _line = "";
using (StreamReader sr = new StreamReader(System.Web.HttpContext.
   Current.Server.MapPath("~/DtenbankDatei.csv")))
{
do
{
//Liest die Reihe vom CSV-Datei
_line = sr.ReadLine();

//Prueft, ob die Reihe leer ist.
if (!String.IsNullOrEmpty(_line))
{
//trennt die Reihen mithilfe von ;-Zeichen
string[] _splits = _line.Split(';');

if (_splits.Length == 8)  //Die Zahl hier ist gleich von den Zahl
   der angegebenen Attribute in der alten Datenbank
{
//Die Zahlen im _splits[] zeigen die Positionen, in denen sich
   zugehoeriges Attribut in der altenDatenbank befindet
//Estellt neues Member
IMember member = Services.MemberService.CreateMember(
_splits[3],     // Username
_splits[2],     // Email
_splits[6],     // Display name
"neueKunden"    // Member type
);

//Member ist bestaetigt zu Login
member.IsApproved = true;

//Prueft, ob zugehoeriges Attribut existiert im Member-Section und
   dann sezt den Wert ein.
if (member.HasProperty("nameDesProperty"))
member.SetValue("nameDesProperty", _splits[4]);


if (member.HasProperty("nameDesProperty") && !String.IsNullOrEmpty
```

```
    (_splits[5]))
member.SetValue("nameDesProperty", _splits[5]);


if (member.HasProperty("nameDesProperty"))
member.SetValue("nameDesProperty", _splits[0]);


if (member.HasProperty("nameDesProperty"))
member.SetValue("nameDesProperty", _splits[1]);

try
{

//Spechert Member
Services.MemberService.Save(member);

//Speichert PIN des Members
Services.MemberService.SavePassword(member, _splits[7]);

//Korrekte Role zuweisen
Services.MemberService.AssignRole(member.Id, "MemberRole");

}
catch { }
}
}
} while (!sr.EndOfStream);

}

return null;

}
}
}
```

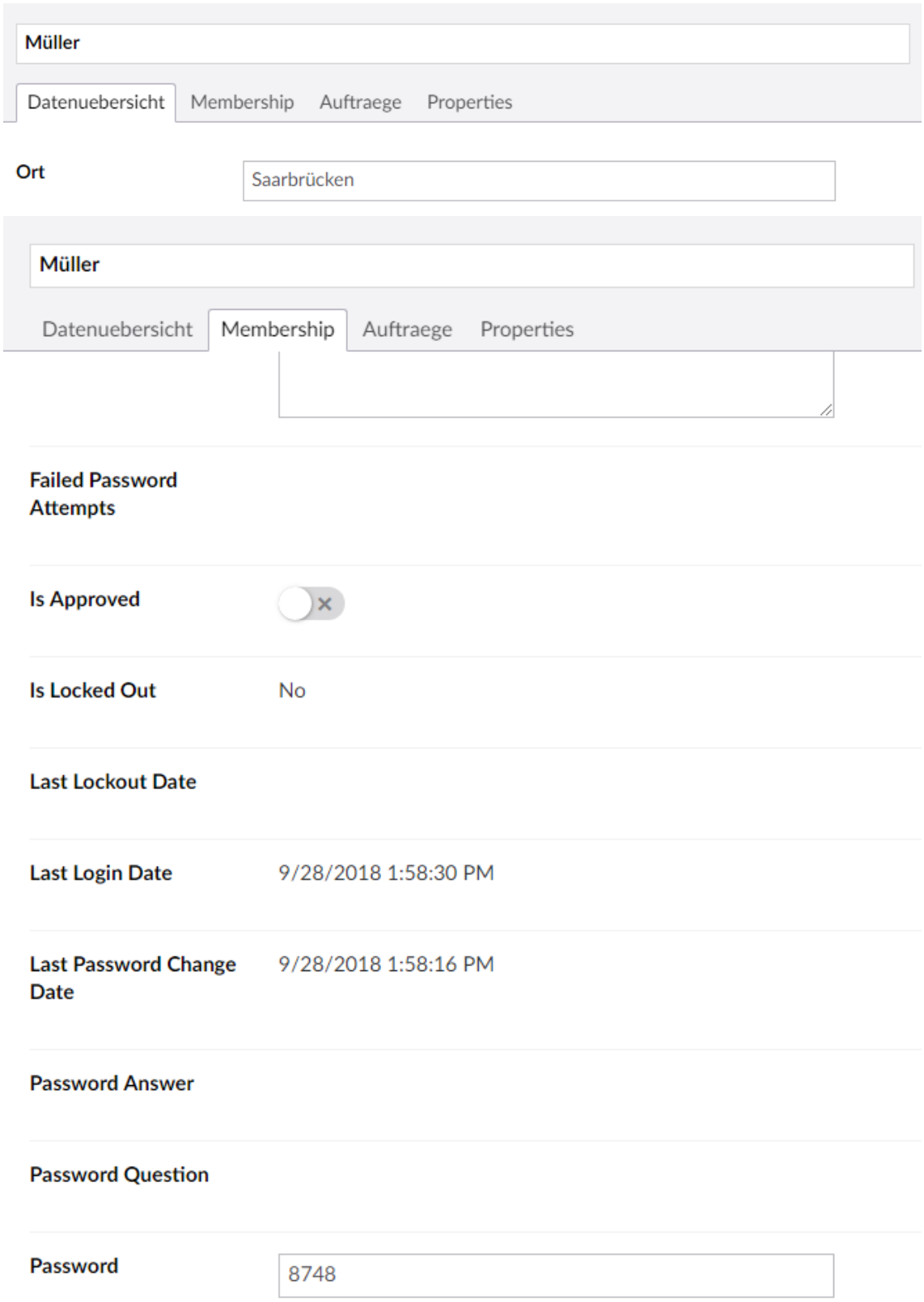

Abbildung A.2: Die Kunde im Members gespeichert.