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 der Basis des Framework ASP erstellt. Der Grund für diese Analyse ist das nicht mehr zeitgemäße Design. Weiterhin ist das Ziel, die Migration der bereits erwähnten Software zu einem Content Management System (Umbraco) zu realisieren. Die Software wird analysiert und bewertet, und auf dieser Grundlage wird ein neues Konzept entwickelt. In der folgenden Dokumentation werden die Vorteile von Umbraco erläutert. Eine ausführliche Beschreibung der Methodik zur Entwicklung dieses Konzept veranschaulicht mithilfe der verwendeten Werkzeuge den Migrationsprozess. Zum Schluss wird das Konzept anhand eines Prototyps des Programms dargestellt.

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 [18]

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.[18]

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. [10]

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 [4].

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 [20].

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. [5]

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.[3]

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. [6]

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 [17]:

- 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] [17]
- 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.[7]

3 IST-Analyse

In dieser Arbeit wird das Bestellsystem 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 befinden sich auf einer gemeinsamen Seite. Das Anmeldeformular hat zwei Felder, in die der Benutzer seinen PIN und seine E-Mail eingeben muss. Neben den Feldern befindet sich Hinweis, der den Kunden mit den nötigen Informationen versorgt, wie er sich anmelden oder registrieren kann. In der Abbildung 3.1 ist das 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. Im zweiten steht die Informationen des Auftrags, darauf folgt die Menü-Auswahl und im vorletzten Schritt werden die Zubehör- und die Serviceauswahl angezeigt. Der letzte Abschnitt dient der Erklärung des Registrierungsprozesses. Die Abbildung 3.2 gibt die Übersicht über Registerformular.

3.1.3 Auftraggeber-Verwaltung

Im Fall einer Registeranfrage kann der Auftraggeber diese bestätigen oder löschen. Dem Auftraggeber stehen drei Optionen zur Verfügung: Auftragsverwaltung, E-Mail-Verwaltung und Umsatzverwaltung. Zuerst soll die Auftragsverwaltung betrachtet werden. Dort finden sich elf Optionen, neun davon leiten zu eigenen Unteroptionen weiter. Die anderen beiden sind „Speichern“ und „Abmelden“. Anhand dieser Optionen kann der Auftraggeber die Kundendaten einsehen, editieren, bearbeiten und löschen. Außerdem kann er

Abbildung 3.2: Die bisherige Eingabemaske für das Registrierungsformular

die dazugehörigen Nachrichten lesen oder löschen. Auch kann er neue und alte Aufträge einsehen und bearbeiten oder löschen. Die Abbildung 3.3 zeigt die Ansicht der Auftraggeber-Verwaltung

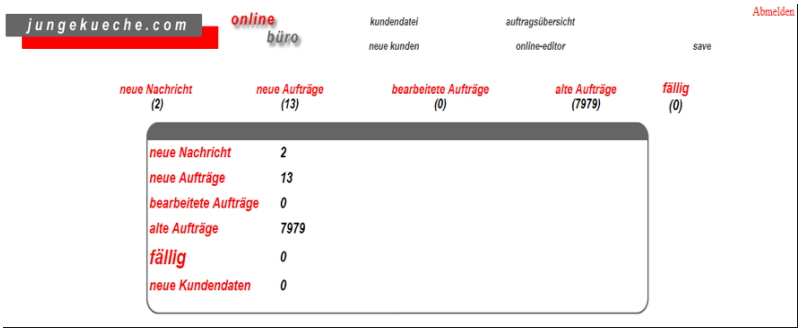


Abbildung 3.3: Die bisherige Eingabemaske für die Auftragsverwaltung

3.1.4 Kundenansicht

Wenn die Anfrage des Kunden bestätigt wird, erhält der Kunde einen PIN per E-Mail. Mit diesem PIN und seiner E-Mail kann der Kunde sich bei „jungekueche“ anmelden. Nach dem Login-Prozess gelangt er auf eine persönliche Nutzerseite. Dort findet er eine Übersicht über seine aktuellen und bisherigen Bestellungen, ein Kontaktfenster, in dem er Nachrichten einsehen oder schreiben kann, sowie eine Art Newsletter. Außerdem gibt es eine Option, über die der Kunde neue Bestellungen aufgeben kann. Abbildung 3.4 veranschaulicht diese Ansicht.

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, den der Nutzer verwendet. Im Backend werden die Daten verarbeitet. Die Verbindung zwischen den beiden

Abbildung 3.4: Die bisherige Eingabemaske für die Auftragsverwaltung

wird über JavaScript und JQuery realisiert. Wenn der Benutzer eine Tätigkeit ausführt, werden die eingegebenen Daten über JavaScript-Methoden zum Backend weitergeleitet. Dort werden sie verarbeitet und über die JavaScript-Methode aufgerufen. Dort werden sie bearbeitet und wieder zu dem Frontend zugeschickt. Das Backend entsteht aus ASP-Funktionen, in denen sich verschiedene Methoden sowie Datenbanken befinden. In den Datenbanken werden die eingegebenen Daten gespeichert. Ein Beispiel wird in der Abbildung 3.5 dargestellt.

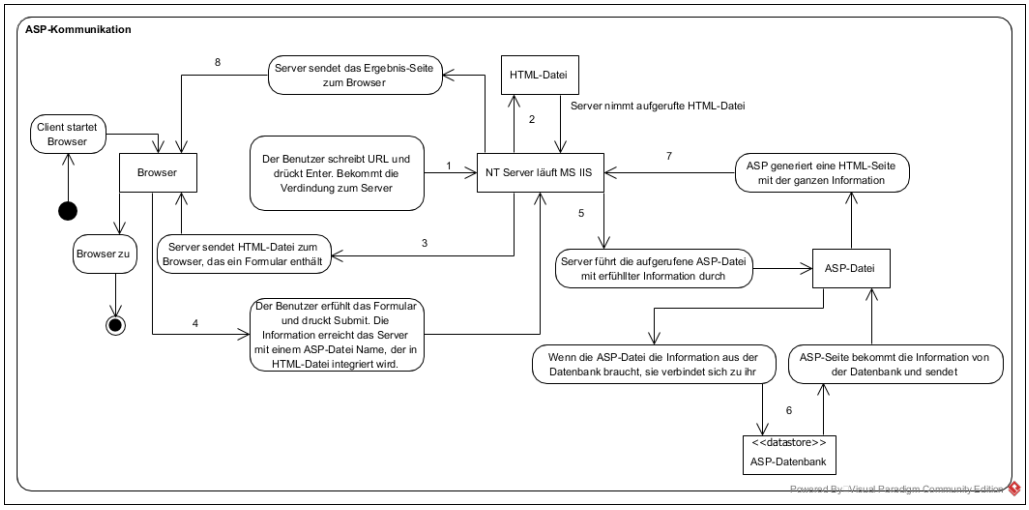


Abbildung 3.5: ASP Architektur Model

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

Nach einem Klick auf das Feld „Bestellformular & Login“ erscheinen das Anmeldeformular sowie die Bestellseite. Wenn der Kunde auf den Button „Anmeldung“ klickt, wird eine HTML-Post-Methode aktiviert. So werden zunächst JavaScript-Methoden aufgerufen. Diese werden benutzt, um die Eingabe auf Richtigkeit zu prüfen. Danach werden Methoden aus der ASP-Datei aufgerufen. Das Registerformular funktioniert nach dem-

selben Prinzip. Nach der Eingabe der benötigen Information werden diese geprüft. Bei einem korrekten Input werden die Eingaben zu den zugeordneten Daten bzw. Methoden geschickt. In der Abbildung 3.6 ist eine Übersicht über das Bestellformular zu sehen.

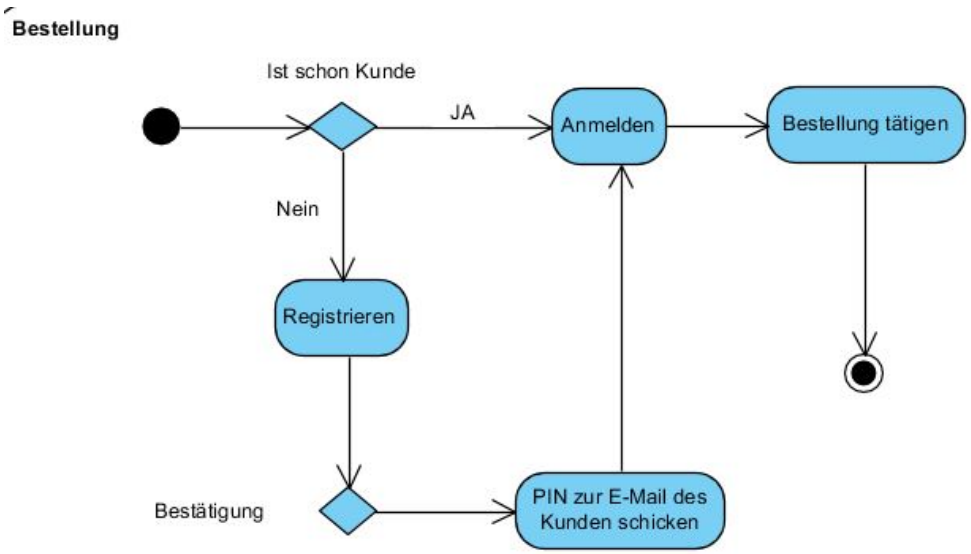


Abbildung 3.6: Die bisherige Eingabemaske für den Vorgang der Bestellung/Registrierung

3.2.2 Auftraggeber-Verwaltung

Die Auftragsverwaltung setzt sich, wie bereits beschrieben wurde, aus verschiedenen Optionen zusammen. Die Funktionsweise der jeweiligen Option soll im Folgenden separat betrachtet werden.

3.2.2.1 Nachricht-Sektion

Sie ermöglicht dem Auftraggeber, die Information zu lesen oder zu löschen. Nach der zugeordneten Wahl wird die Information in der Datenbank „komentar“ entweder aufgerufen oder entfernt. In der Abbildung 3.7 ist diese Aktivitäten zu sehen.

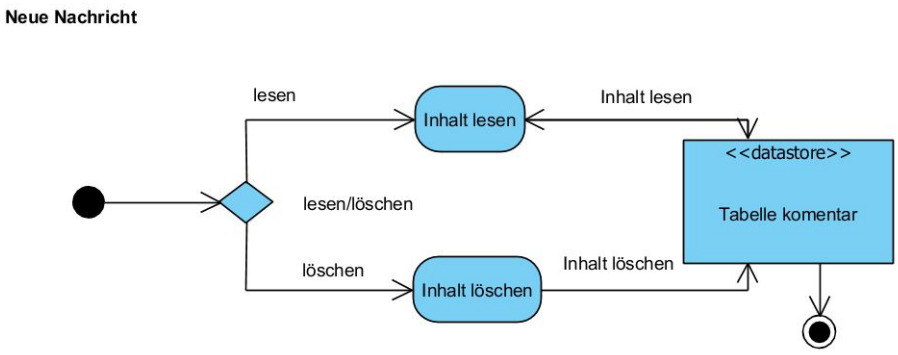


Abbildung 3.7: Die bisherige Eingabemaske für den Nachrichten lesen oder löschen

3.2.2.2 Aufträge: Neue, alte und bearbeitende

Bei allen drei Arten von Aufträgen stehen identische Verwaltungsmöglichkeiten zur Verfügung. Der Auftraggeber kann den Auftrag sehen, editieren oder löschen. Jede dieser

Aktivitäten ist mit mehreren Datenbanken verbunden, die verschiedene Informationen über die gewählten Artikel beinhalten. In der Abbildung 3.8 wird zum Beispiel die Funktionalität 'löschen' gezeigt. Es spielt keine Rolle, welche Art von Auftrag gewählt wird, denn die Funktionalität umfasst dieselben Datenbanken.

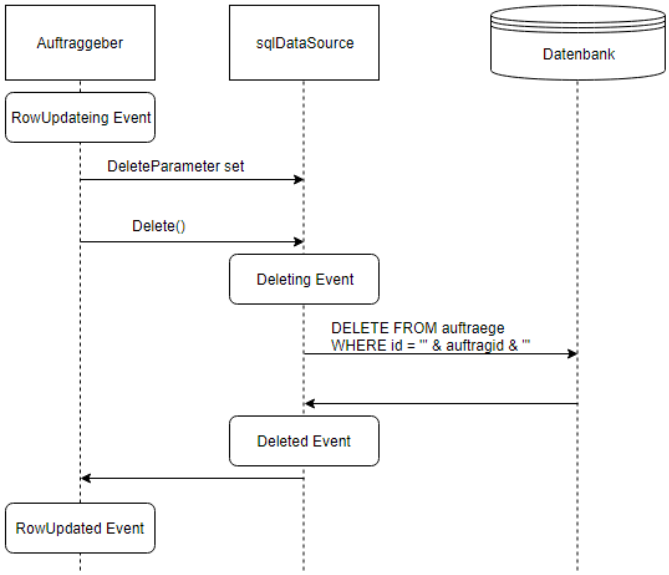


Abbildung 3.8: Die bisherige Eingabemaske für den Auftrag löschen

3.2.2.3 Kundendatei

Hier befinden sich die Information über die Kunden. Es gibt die Möglichkeit, die Information zu editieren oder zu löschen. Um einen bestimmten Kunden schneller zu finden, steht eine Suchmaschine zur Verfügung. Eine Übersicht wird in der Abbildung 3.9 dargestellt.

Abbildung 3.9: Die bisherige Eingabemaske für die Kundendatei

Wenn man editieren oder löschen möchte, geschieht dies über die JavaScript-Funktion. Diese Funktion ruft die Methoden auf, die sich in der zugeordneten ASP-Datei befinden. Im „Editor-Fenster“ lassen sich die Daten des Kunden bearbeiten, Login-Daten wie E-Mails an Kunden übermitteln, wichtige Information für alle Kunden senden, es kann dem Kunden ein V.I.P-Status gegeben, es können spezifische Warnungen an den jeweiligen Kunden gegeben, Informationen geschrieben sowie Unterschriften editiert werden etc. Das Fenster wird in Abbildung 3.10 gezeigt.

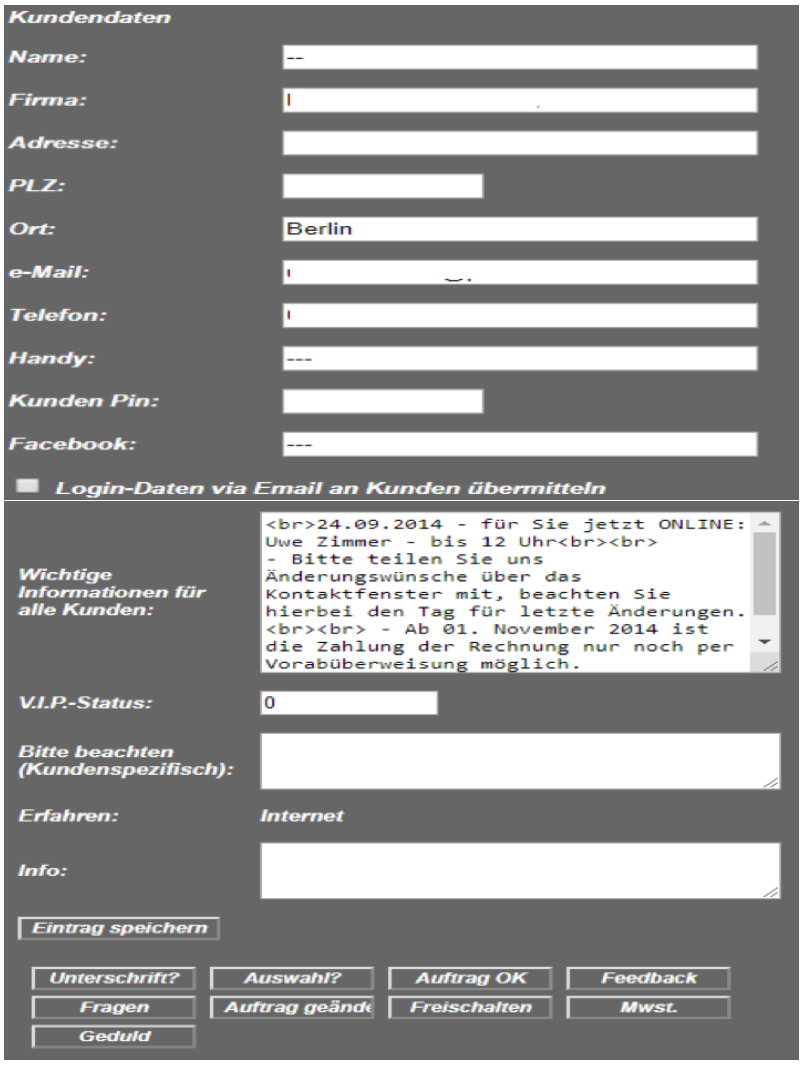


Abbildung 3.10: Die bisherige Eingabemaske für das Kunden-Editor

Durch JavaScript-Methoden wird eine zusätzliche Möglichkeit generiert, neue Nachrichten an dem Kunden zu schreiben. Sie wird aktiviert, wenn in der Liste des Adressbuchs mit dem Cursor über den Namen eines bestimmten Kunden gefahren wird. Die Kommunikation zwischen dem Auftraggeber und dem Kunden wird auch in der Datenbank „komentar“ gespeichert.

3.2.2.4 Online-Editor

Mit dieser Option können neue Artikel erstellt, editiert und gelöscht werden. Es gibt „Artikel-Arrangements“ und „Artikel-Standard“. Bereits erstellte Artikel können entweder gelöscht, editiert oder zugeordnet werden. JavaScript-Funktionen werden verwendet, um die Eingaben zu prüfen. Wenn die Eingaben korrekt ausgefüllt worden sind, werden die obengenannten Methoden aus den jeweiligen Dateien aufgerufen. In der folgenden Abbildung 3.11 ist eine Übersicht über die Seite zu sehen.

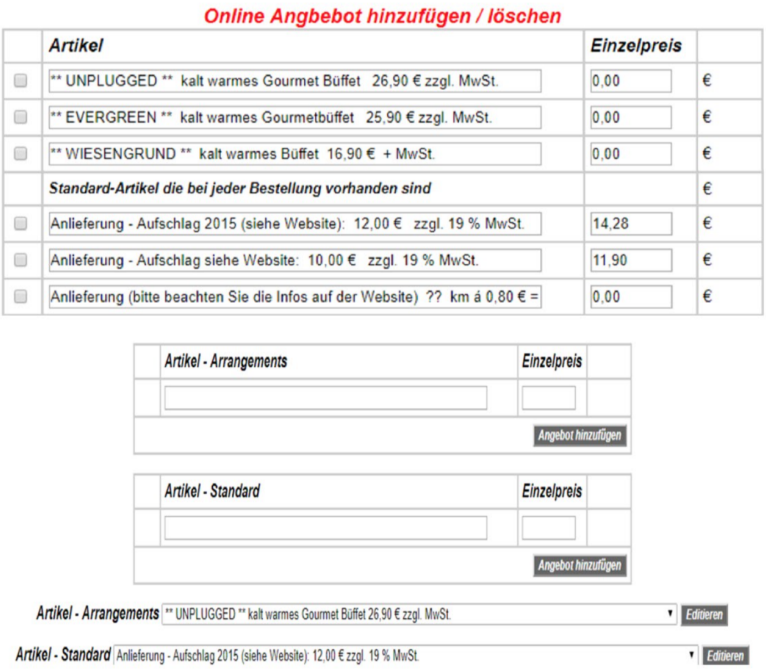


Abbildung 3.11: Die bisherige Eingabemaske für die Übersicht das Menü

Wenn die Option „Editieren“ zu den „Artikel-Arrangements“ ausgewählt wird, wird eine neue Seite geöffnet, in der es verschiedene Möglichkeiten gibt, die Inhalte und Daten zu bearbeiten. Abbildung 3.12 zeigt diese Seite. Bei „Artikel-Standard“ gibt es nur eine Möglichkeit. Sie wird in der Abbildung 3.13 dargestellt.

3.3 Testen

Für diese Arbeit wird ein webbasierter Sicherheit-Checker von 1&1 verwendet. Eine grundlegende Recherche hat gezeigt, dass die Webseite auf einer mittleren Sicherheitsstufe 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.
- Apache-Status ist verboten. Die Status-Seite ist öffentlich nicht erreichbar. So wird den Schutz von potenziellen Angreifern erhöht.
- Die Server Version ist öffentlich nicht einsehbar. Damit kann ein Angreifer nicht einfach bekannte Schwachstellen ausnutzen.

Wie die Abbildung 3.14 zeigt, ist die Webseite gesichert, aber es gibt einige Bereiche, die eine Optimierung benötigen würden. Die Seite lädt langsam, wird im Browser relativ oft gefunden, aber ist sie gesichert. Über einen anderen Web-Checker werden weitere Funktionalitäten der Webseite analysiert. Observatory [7] von Mozilla analysiert den Browser im vier Schritten.

- Hypertext Transfer Protocol (HTTP)

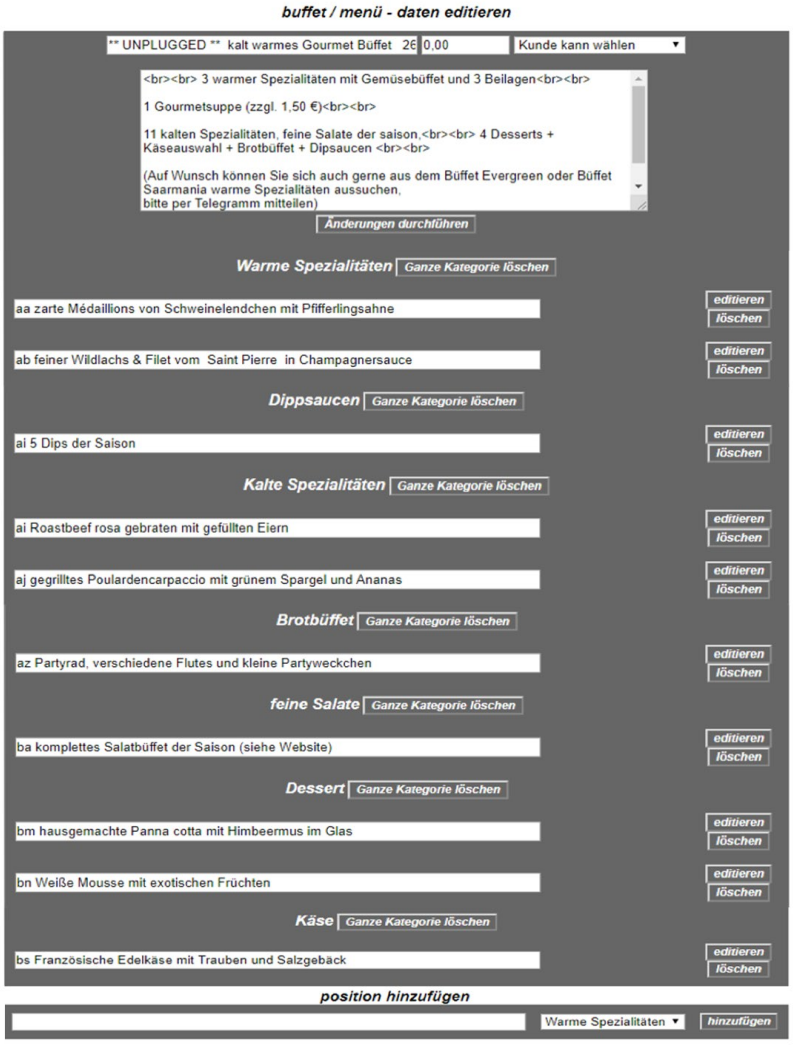


Abbildung 3.12: Die bisherige Eingabemaske für die Übersicht des Menü-Editor-Arrangements

- Transport Layer Security (TLS)
- Secure Shell (SSH)
- Third-Party Tests

Hier werden nur die ersten beiden Schritte betrachtet, da nur sie für diese Arbeit relevant sind.

Abbildung 3.13: Die bisherige Eingabemaske für die Übersicht des Menü-Editor-Standard

Abbildung 3.14: Ergebnis vom 1zu1

HTTP Observatory

Hier wird die aktuelle Sicherheit der Webseite im öffentlichen Internet betrachtet. Alle Anfragen werden über POST- oder GET-Methoden durchgeführt und die Rückmeldungen erfolgen im JSON-Format. In dieser Analyse wird gezeigt, dass Content Security Policy (CSP)nicht implementiert wurde. CSP verhindert Weitbereich von Cross-Site-Scripting (XSS)- und „clickjacking"-Attacken. Cookes sind teilweise gesichert. Hier fehlen das „Secure"-Attribut sowie „SameSite" und „Prefix". Durch „Secure" werden die Cookies davon abgehalten, dass sie über nicht gesichertes HTTP versendet werden. „SameSite" hält die Cookies von Cross-Site-Request-Forgery (CSRF)-Attacken und Cross-Site-Übersendung ab, d. h. es ist nicht auszuschließen, dass ein Hacker-Angriff auf das Computersystem des Benutzers erfolgt. Es werden keine _Host- und _Secure-Prefixe auf den Namen der Cookies verwendet. Das bedeutet, dass sie nicht gegen Überschreiben geschützt sind. Die Abbildung 3.15 zeigt die genaue Bewertung der Webseite mit dem Resultat der durchgeführten Analyse.

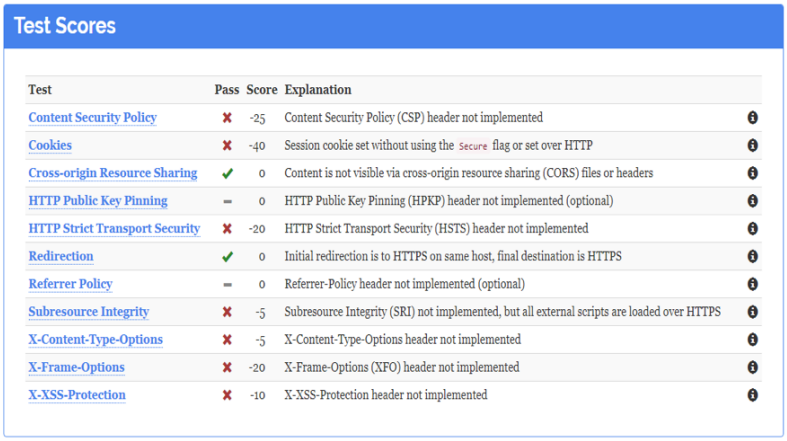
TLS

Die hier betrachtete Webseite verwendet das Signaturverfahren SHA-256-With-RSA. Das bedeutet, dass eine Kombination zwischen SHA256 (kryptologische Hashfunktion,

die Hashwerte mit einer Länge zwischen 256 und 512 Bits erzeugt) und Rivest–Shamir–Adleman (RSA) (dadurch können symmetrische Schlüssel jeder üblichen Länge erzeugt werden) vorliegt (siehe Abbildung 3.16).

3.4 Warum ist eine Migration notwendig?

ASP ist eine nicht Objekt-Orientierte Skripte Sprache. Es ist anfällig zu Unterbrechung der Applikationen, z. B. wenn die Internet Information Services (IIS) gestoppt und neu



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 3.15: Ergebnis vom HTTP Observatory

Certificate Information

Common name:	jungekueche.com
Alternative Names:	
First Observed:	2018-08-06 (certificate #187995380)
Valid From:	2018-07-27
Valid To:	2018-10-25
Key:	RSA 2048 bits
Issuer:	Let's Encrypt Authority X3
Signature Algorithm:	SHA256WithRSA

Abbildung 3.16: Ergebnis vom TLS Observatory

gestartet werden. Wenn eine ASP-Seite aufgerufen wird, wird der Text linear analysiert. Die Frontend-Verwendung des Bestellsystems ist verschachtelt und es gibt redundante Optionen. Das Login- und Registrierungs-Menü sowie das Bestellformular befinden sich auf einer Seite. Dadurch wird das Öffnen der Seite verlangsamt. Es gibt unnötigen Quellcode, der entfernt werden kann. Ein anderer Nachteil des Systems ist, dass es nicht dynamisch orientiert ist, d. h. die Umgebung ist nicht benutzerfreundlich. Der Auftraggeber muss für jede nötige Änderung den Entwickler kontaktieren. Dadurch ist der Benutzer vom Hersteller des Programms abhängig. Aktuell gibt es eine große Auswahl an CMS, durch die dem Auftragsgeber zahlreiche Möglichkeiten für Änderungen seiner Website zur Verfügung stehen. Im folgenden Kapitel wird eines dieser CMS erläutert.

4 Anforderungsanalyse

In der hier durchzuführenden Analyse soll ein neues Konzept für das Bestell- und Verwaltungssystem dargestellt werden. Auf der Basis der IST-Analyse wird eine Verbesserung mit aktuellen Webtechnologie und Verfahren durchgeführt. In diesem Kapitel sollen die bestehenden Probleme und das Optimierungspotenzial näher betrachtet werden. Das neue Bestellsystem soll nicht kompliziert sein, sondern benutzerfreundlich. Die Optionen sollen deutlich zu erkennen und einfach zu bedienen sein. Sicherheit ist ein wesentlicher Teil der webbasierten Entwicklung, weshalb zusätzliche Prüfungen in dieser Richtung eingeführt werden. Die Auftragsverwaltung und die Kundenübersicht werden ihre behalten. Das gesamte Konzept wird aus einer Umbraco-Instanz heraus verwaltet. Im folgenden Unterkapitel werden die oben beschriebenen Anforderungen erweitert und erläutert.

4.1 Paket A

Die aktuelle Software gibt dem Auftraggeber wenige Möglichkeiten, Änderungen im Frontend vorzunehmen. Es wurde immer ein Entwickler benötigt, um etwas zu ändern. Beispielsweise kann der Auftraggeber nicht einmal die Schrift oder die Farbe des Textes editieren. In diesem Unterkapitel wird versucht, dem Auftraggeber in diesem Bereich völlige Freiheit zu lassen. Es wird besonders darauf geachtet, dass die Möglichkeiten, zu ändern und editieren, erhalten bleiben. Als Beispiele wurden Schriftart und Farbe vorgegeben. Dazu wird die Verwendung von Grids, Makros und Formularen empfohlen. Diese und weitere Begriffe werden im nächsten Kapitel erläutert.

4.2 Paket B

4.2.1 Kundenverwaltung

In diesem Unterkapitel werden die Anforderungen betrachtet, die der Auftraggeber gestellt hat, um die Kundenaktivitäten besser kontrollieren zu können. Es wird eine eigene Seite für ihn erstellt, in der Kunden Bestellungen erstellen und einsehen oder Nachrichten verschicken oder lesen können.

4.2.1.1 Kundenerfassung

Hier werden die Anforderungen zum Onlinebestellsystem und der Kundenübersicht beschrieben.

1. 1. Der Kunde kann sich bei der Erstbestellung registrieren und erhält per E-Mail eine PIN. Bei jeder weiteren Bestellung kann er sich mit seiner E-Mail und der PIN einloggen. Es soll darauf geachtet werden, dass beim Bestellen die E-Mail-Adresse überprüft werden muss. Wenn die eingegebene E-Mail-Adresse bereits verwendet wurde, soll ein Hinweis erscheinen. Für Neukunden steht eine private Profil-Seite zur Verfügung. Nach der Registrierung ist es möglich, dass der Kunde eine erste Bestellung tätigt, für weitere Anmeldungen muss er jedoch vom Auftraggeber bestätigt werden. Nach dieser Bestätigung erhält der Kunde bereits erwähnte PIN. In

den Abbildungen 3.2 und 3.1 sind das Register- und das Anmeldeformular zu sehen. Nach den vorgegebenen Anforderungen sind zu diesem Punkt die Funktionalitäten nicht zu ändern. In die Entwicklung müssen die bereits erläuterten Funktionalitäten und Aktivitäten in der Umbraco implementiert werden.

4.2.1.2 Kundenansicht

1. Wenn der Kunde vom Auftraggeber verifiziert wird, kann er sich mit seiner E-Mail und seiner PIN einloggen und seine aktuelle Bestellung wie auch die vergangenen Bestellungen einsehen. Für Kommunikation und Absprachen wird ein Kontaktfenster verwendet. Der Auftraggeber kann entweder allgemeine Information an seine Kunden schicken oder kundenspezifisch antworten. Die Abbildung 3.4 stellt die bisherige Kundenansicht mit den alten Funktionalitäten dar. Diese sollen nicht geändert, sondern in dem neuen System implementiert werden. Die Kunden sollen über die Member-Section im Umbraco integriert werden.

4.2.1.3 Auftraggeber-Ansicht

1. Der Auftraggeber erhält eine Kundenübersicht, die er filtern kann. In der Detailansicht sieht er die Informationen über Kunden. Dort kann er den Bestellverlauf nachverfolgen und aus der Ansicht heraus E-Mails an den Kunden senden (Mail-Vorlage oder frei gestaltbar). Dies wird in den Abbildungen 3.9 und 3.10 dargestellt. In dem neuen Konzept wird gefordert, dass die Funktionalitäten der alten Software im neuen System gleich 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 (z. B. gelesen und nach Kunden) integriert werden. Der „Chat“ sollte aus der Kundenkartei-Ansicht ebenfalls funktionieren und die Kommunikation soll an einen Auftrag gebunden sein. Die Kommunikationsübersicht wird in Abbildung 4.1 illustriert.

Im neuen System soll dasselbe Funktionalität des Chats bleiben.

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. Derzeit sind die Arrangements noch nicht mit den Webseiten gekoppelt, aber sie müssen in der neuen Software daran gebunden werden.

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 gefordert, dass die Artikel einfacher verwaltet werden können und die Artikelseite der Webseite direkt mit dem Artikel gekoppelt sein soll.

   Die Bisherige Übersicht zum Artikel ist in den Abbildungen 3.11 und 3.12 zu sehen.

Abbildung 4.1: Die bisherige Eingabemaske für die Nachricht

4.2.3 Auftragsverwaltung

4.2.3.1 Übersicht

1. Der Auftraggeber erhält eine Übersicht über seine aktuellen Aufträge. Für jede Art von Auftrag (neu, bearbeitet, vergangen und fällig) gibt es einen 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 von Kunde zu Umbraco-Member geben. In der Abbildung 4.2 ist die Übersicht zu sehen.

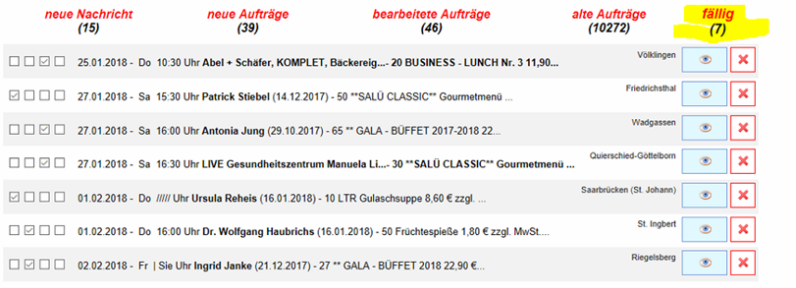


Abbildung 4.2: Die bisherige Eingabemaske für die Übersicht

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 vorgegeben) sein.
3. Die Daten der Auftraggeber und der Kunden von zweitem Blatt müssen für Auftraggeber zur Verfügung stehen, 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 Formular für Terminanfragen über die Webseite. Diese erzeugt eine E-Mail an den Auftraggeber, der in der E-Mail nur einen Link betätigen muss, 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. Siehe dazu Abbildung 4.4.

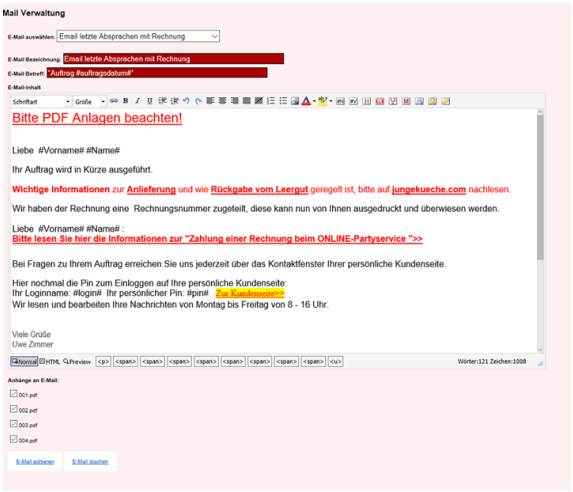


Abbildung 4.4: Die bisherige Eingabemaske für die E-Mail-Verwaltung

4.2.5 Umsatzverfassung

1. Hier kann der Auftraggeber die Umsätze der letzten Monate / Jahre, die über die Aufträge zustande gekommen sind, einsehen. Dabei ist es wichtig, dass diese Datensätze nicht direkt an die Aufträge gekoppelt sind, sondern aus einer separaten 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.

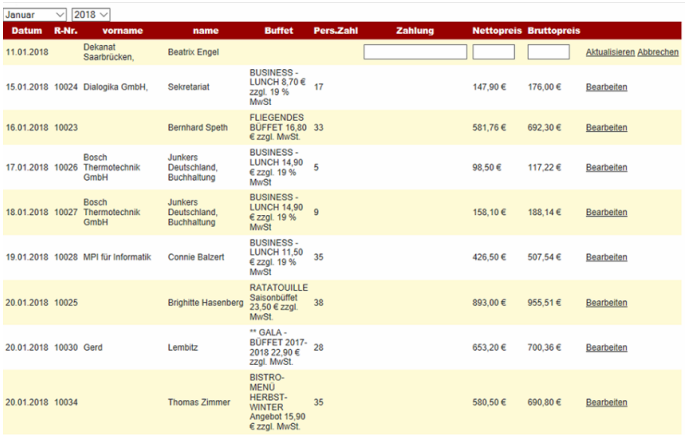


Abbildung 4.5: Die bisherige Eingabemaske für die Umsatzerfassung

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 gleichbleiben. Das Projekt ist in zwei große Pakete unterteilt – Paket A und Paket B. Im ersten Paket wird die Verwaltung des Frontend aufgebaut. Das Ziel ist es, maximale Flexibilität für den Website-Besitzer herzustellen und diesem zu ermöglichen, durch einfache Tätigkeiten bestimmte Zwecke zu erfüllen. In diesem Bereich soll der Inhalt der Webseite editiert, geändert und erweitert werden können. Umbraco verfügt über sogenannte „Grids“, die dazu dienen, das Design der Seite zu manipulieren und die bereits besprochenen Möglichkeiten auszunutzen. Dazu werden eigene Makros benutzt, in die ein Quellcode der SHOP-Komponenten hineingeschrieben wird. Somit kann der Auftraggeber Makros in beliebigen Teilen der Seiten einfügen. Umbraco-Forms werden als Formulare benutzt, damit der Nutzer die Felder selbst anordnen kann. Beide Pakete werden als Startknoten aus einer Umbraco-Instanz heraus verwaltet. Paket B ist als Kern der Seite aufgebaut und entspricht dem Online-Bestellsystem. Hier ist es wichtig, dass eine unkompliziert und reibungslos bedienbare, 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 bereits erklärt wurde, ist Umbraco [19] ein Content-Management-System, das flexibel und benutzerfreundlich ist. Um die Funktionen besser verständlich zu machen, wird es in diesem Unterkapitel erläutert. Das User Interface von Umbraco ist in drei Teile unterteilt. Der erste Teil ist die Hauptfunktion bzw. Section. Dort befinden sich die Hauptoptionen: Content, Media, Settings, Developer, Users, Members, Forms. Damit zwischen „Member“ und „User“ eindeutig unterschieden werden kann, werden die beiden Begriffe erklärt. „User“ ist jemand, der Zugriff zum „Umbraco-Backoffice“ und dort wiederum bestimmte Rechte hat. Ein „Member“ wird von Umbraco für die Registrierung und Authentifizierung eines externen Besuchers benutzt. Das sind Personen, die nur das Frontend benutzen dürfen. Es kann auch eine „Custom Section“ erstellt werden, die im einem weiteren Unterkapitel erklärt wird.

Der nächste Teil ist die Unternavigation oder auch Tree genannt. Dort stehen alle Unteroptionen, wobei jede Hauptfunktion ihre eigenen Unteroptionen hat. Im Folgenden

werden die zugehörigen Funktionalitäten der Unteroptionen (Trees) betrachtet. Die Abbildung 5.1 stellt das Funktionsprinzip von Umbraco [11] dar.

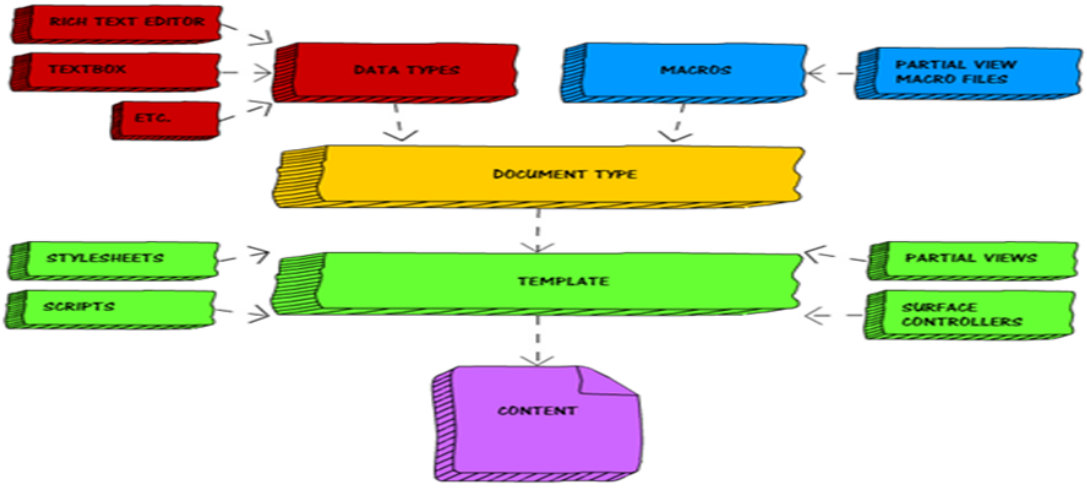


Abbildung 5.1: Funktionalität vom Umbraco-Backoffice [21]

- **Trees vom Content-Section:** In diesem Tree befinden sich alle Seiten, die im Website-Frontend erscheinen können. Dort befindet sich auch der Recycle Bin oder Papierkorb. Damit lässt sich die gelöschte Seite zurücksetzen.
- **Trees von der Media-Section:** Hier stehen alle hochgeladen Videos und Bilder zur Verfügung.
- **Trees vom Settings-Section:** In diesem Tree befinden sich CSS, JavaStript, Document Types und zugehörige Templates. PatialView ist auch dort, das eine Übertragungsfunktion von Backend zum Frontend hat.
- **Trees vom Developer-Section:** Hier stehen Trees zur Verfügung, durch die der Entwickler bereits erstellte Seiten weiter entwickeln kann. 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 das Umbraco-Backend benutzen dürfen. Für einen bestimmten User können verschiedene Rechte freigegeben werden.
- **Trees vom Member-Section:** Hier sind Benutzer, die Frontend benutzen dürfen.
- **Trees vom Forms-Section:** Hier lassen sich verschiedene Arten von Formularen erstellen.

Der dritte Teil des Umbraco-Backoffice ist der Editierbereich. Dort werden alle Eigenschaften der jeweiligen Tree-Optionen dargestellt.

In der Abbildung 5.2 wird gezeigt, wie das Umbraco-Backoffice aussieht.

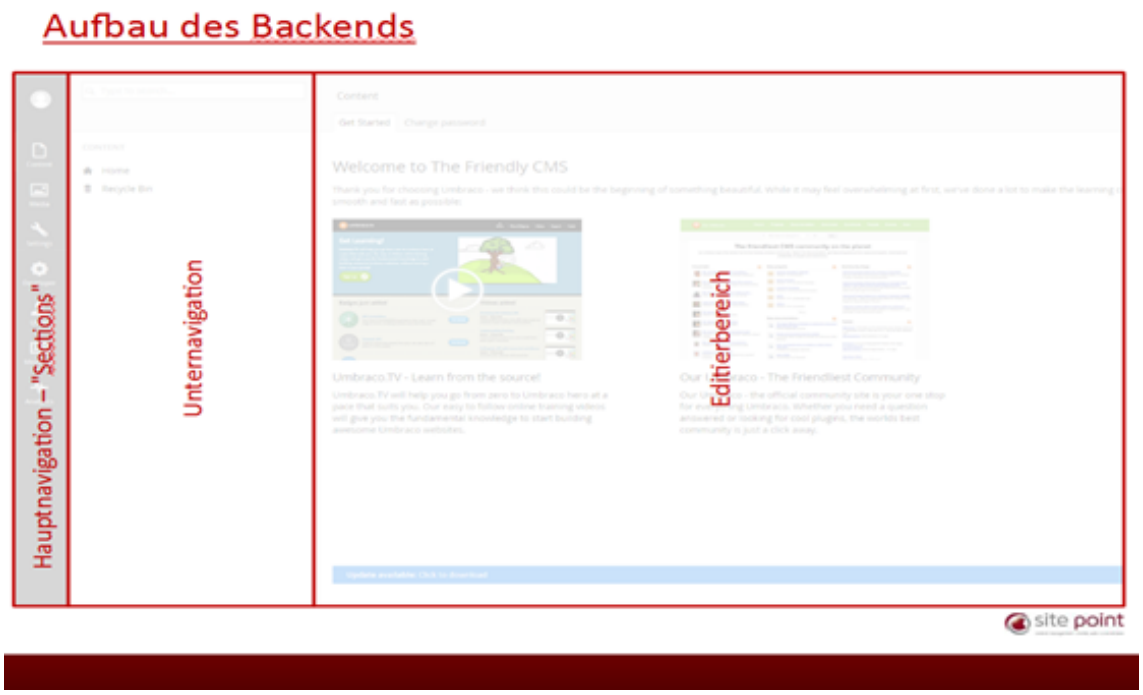


Abbildung 5.2: Übersicht vom Umbraco-Backoffice
[2]

5.2 Paket A

Damit der Auftraggeber mehr Flexibilität erhält, um die Frontend-Seite zu editieren, werden „Grids“ (Rahmen) verwendet. Ein Grid enthält zwölf Spalten, wobei diese auch verbunden werden können. So ist es möglich, dass die Seite beliebig aufgeteilt wird, wie in Abbildung 5.3 gezeigt wird.

Im Grid lassen sich eigene Einstellungen vornehmen. In dieser Arbeit werden zwei Beispielen gezeigt, wie Grids verwendet werden könnten, so etwa um die Farbe und die Größe der Schrift zu ändern. Für das geforderte Ziel werden der Rich Text Editor und Makros verwendet. Diese können im Rich Text Editor implementiert werden. Das soll in einem späteren Kapitel gezeigt werden. Hier werden nur die bereits erwähnten Beispiele betrachtet. Sie werden in den „Stylesheets“ unter „rte“ eingegeben. In Abbildung 5.4 wird gezeigt, wie CSS-Befehle in Umbraco geschrieben werden können.

Über den Rich Text Editor kann der Inhalt des Frontend ebenfalls editiert werden, aber nur mit festen Optionen. Diese Stylesheets sind in den Rich Text Editor integrierbar, wodurch der Auftraggeber die Schriftart und die Farbe ändern kann. In der Abbildung 5.5 wurde die Schrift auf rote Farbe und die Größe 18 pt geändert.

5.3 Paket B

5.3.1 Kundenverwaltung

Die Kundenverwaltung umfasst den Bereich, der sich auf die Kunden bezieht.

Mehr wird im nächsten Unterkapitel erklärt.

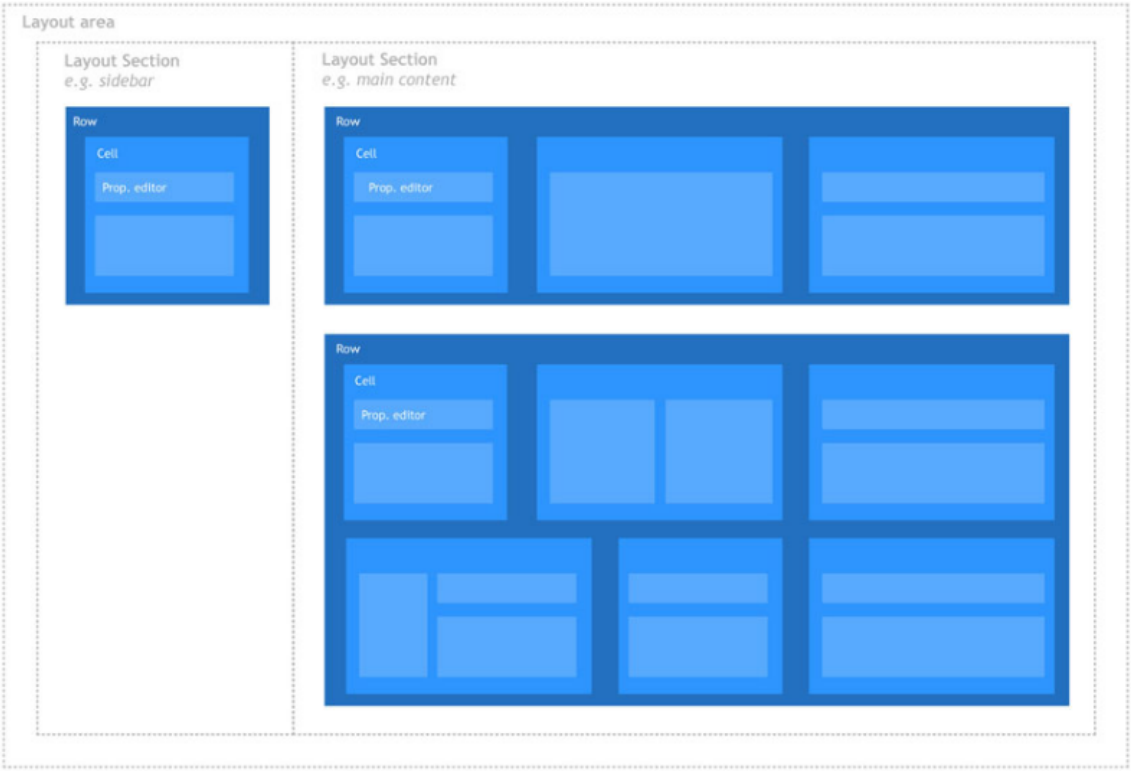


Abbildung 5.3: Übersicht vom Umbraco-Grids

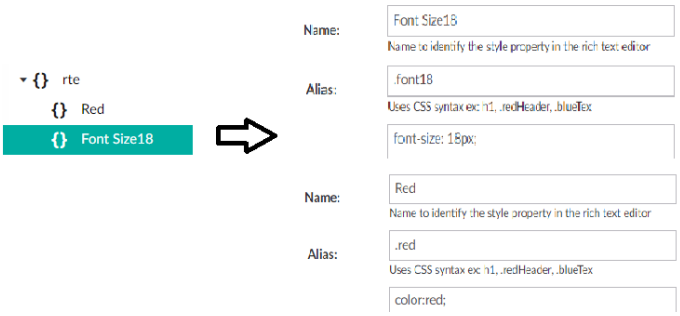


Abbildung 5.4: Styling vom Umbraco-Grids

5.3.1.1 Kundenerfassung

1. Die vorgegebenen Bedingungen für diesen Bereich umfassen Registrierung und Anmeldung. Nach wie vor kann sich der Kunde anhand einer Personal-Identification-Number (PIN) und seiner E-Mail-Adresse zu dem System anmelden. Diese PIN wird automatisch generiert und nach einer Bestätigung an die E-Mail-Adresse des Kunden gesendet. Dabei ist zu beachten, dass überprüft wird, ob sich die E-Mail-Adresse bereits in dem Register befindet. Außerdem wird gefordert, dass der Auftraggeber den Antrag des Kunden bestätigen kann. Nach der Bestätigung kann sich der Kunde anmelden und dort seine private Seite des Websystems verwalten. Die Funktionalitäten bleiben die gleichen wie bei dem alten System. Die Kundenerfassung wird durch Member-Application Programming Interface (API) von Umbraco realisiert. Über „MemberService“ ist die MemberAPI erreichbar. Diese Bibliothek wird unter den Services [14] property von dem SurfaceController [13] zu Verfügung gestellt. Das ist eine komplexe Methode, für die verschiedene Dateien benötigt werden: (Controller, Model [13], Partial View und Source Datei, in der View-Methoden

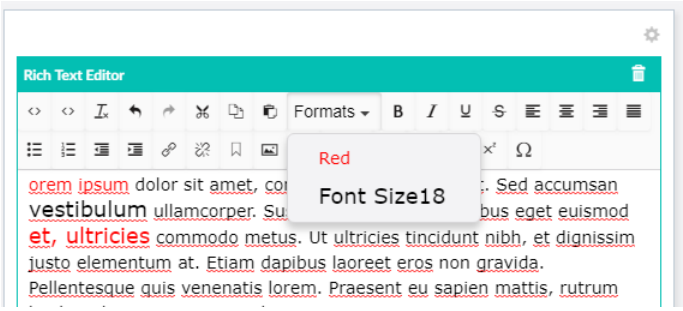


Abbildung 5.5: Styling in Umbraco-Grids - Rich text editor

über eine Aufruf-Funktion aufgerufen werden). Die zur Registrierung benötigte Information findet sich in der Model-Datei. Dort stehen die Model Properties oder auch die Parameter, mit denen hier gearbeitet wird. Über das Model werden die Properties von Partial View zum SurfaceController oder umgekehrt übertragen. Der SurfaceController ist gewissermaßen die ‚Autobahn' zur Umbraco-Datei, ein Model View Controller (MVC), der mit Umbraco interagiert. Er wird von der Bibliothek Umbraco.Web.Mvc.SurfaceController geerbt. Über Partial View werden die Verbindungen zwischen dem Kontakt-Formular und den Model Properties hergestellt. Hier handelt es sich eigentlich um eine Teilansicht, die vom Umbraco-Frontend benutzt wird. Dort befindet sich das Umbraco User-Interface (UI). Die folgende Abbildung 5.6 zeigt, wie die obengenannten Begriffe sich zueinander verhalten.

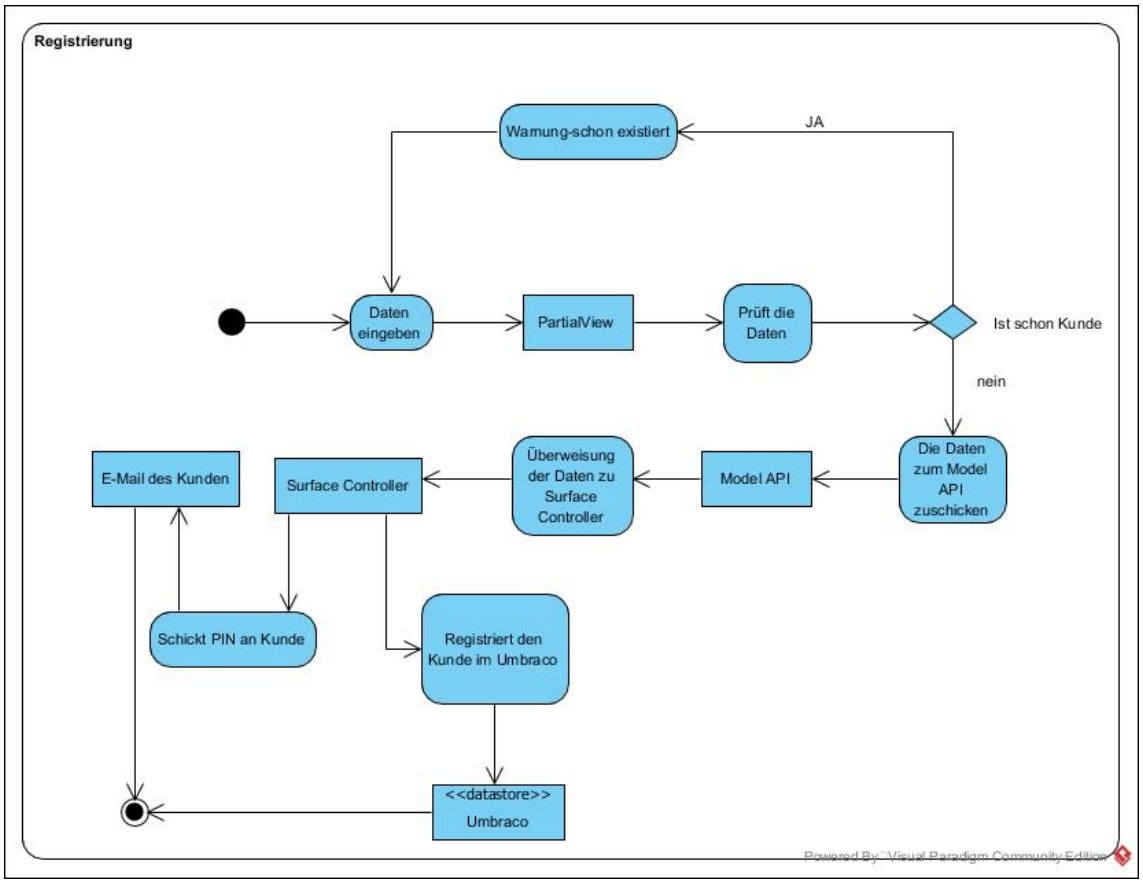


Abbildung 5.6: Neues Konzept zum Registrieren

Diese Möglichkeiten vom Umbraco erlauben dem Benutzer, sich leicht und bequem

beim Server anzumelden oder sich zu registrieren. In Anhang - Kundenerfassung wird noch einmal übersichtlich gezeigt, wie genau vorzugehen ist, damit diese Anforderung erfüllt wird.

5.3.1.2 Kundeansicht

Um dem Kunden die Handhabung zu erleichtern, sollen folgende Möglichkeiten zur Verfügung stehen:

- 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 Listing 5.1 und Anhang A.5 zeigen, wie der Kunde seinen PIN bekommt, obwohl er sie bei der Registrierung nicht eingegeben hat.

Listing 5.1: JavaScript PIN Generator

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

2. 2. Nach der Registrierung sieht der Kunde eine neue Seite. Dort kann er die obengenannten Optionen verwenden. Wenn der Kunde eine neue Bestellung tätigen will, wird ein neues Fenster geöffnet, in dem er die gewünschten Artikel wählen und bestellen kann. Die gewählten Produkte werden in den Datenbanken gespeichert, wo sie als vergangene Bestellungen verwendet werden können. Das selben Prinzip steht auch für die Kommunikation zwischen dem Auftraggeber und dem Kunden zur Verfügung. Wenn eine Nachricht geschrieben wird, wird sie in einer anderen Datenbank gespeichert. Vom Umbraco wird die Information direkt an den Kunden gesendet. Der Auftraggeber und der Kunde können in der Datenbank die Bestellungen und die Nachfragen ansehen. In den folgenden Kapiteln wird detailliert erklärt, wie die Kommunikation und die Aufträge funktionieren. 3.Wie gefordert, wird dem Kunde eine Profile-Seite über Member angezeigt. Das bedeutet, dass diese Seite für den jeweiligen Kunden personalisiert ist. Die Abbildung wird über memberId geschehen. Es wird ein Filter erstellt, durch den sich diese Personalisierung herstellen lässt. memberId befindet sich in der Umbraco-Member-Datenbank. Mithilfe von Macros und Grids wird die Kundeansicht entwickelt. Das, was im Kundenansicht gemacht wurde, ist Verwendung von Macros und Grids. (zur Veranschaulichung siehe Abbildung 5.7).

Über ein Makro wie in Listing 5.2 ist es möglich, die Meldungen vom Auftraggeber unter „Wichtige Information“ zu sehen. Im Makro wurde eine Methode geschrieben, über die die Übertagung von der Member-Section zur Content-Section ermöglicht wird. Abbildung 5.7 zeigt, wie der Auftraggeber dem Kunden eine Informationsmeldung aus dem Member-Bereich schickt.

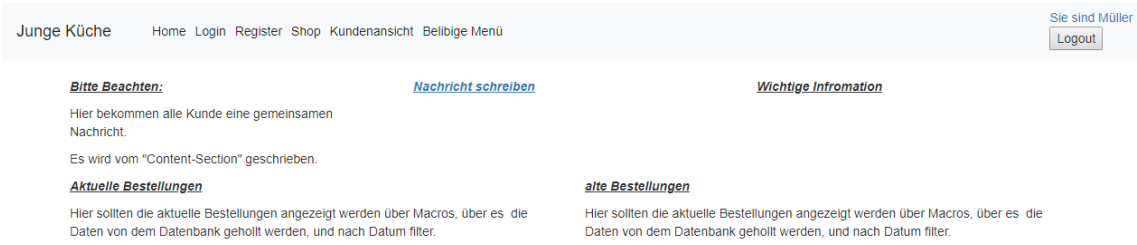


Abbildung 5.7: Kundeansicht

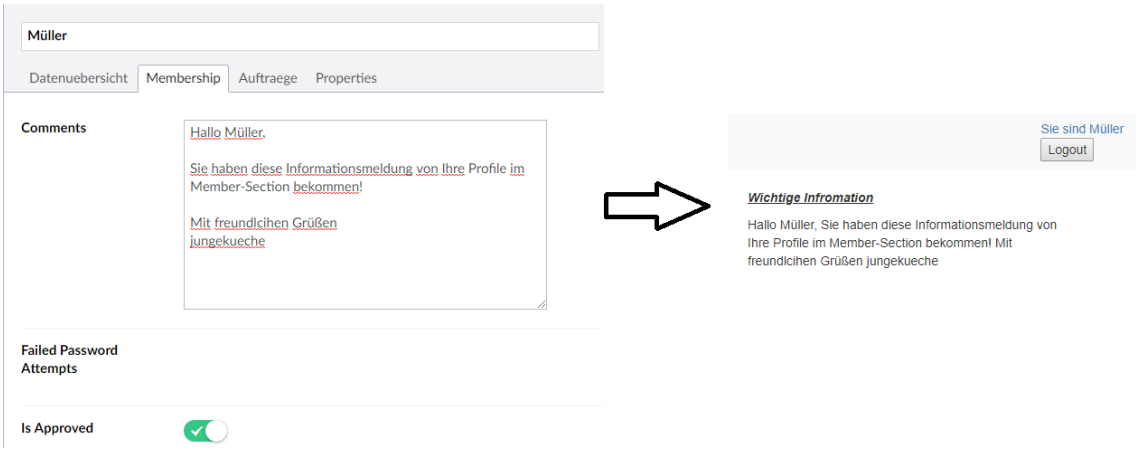


Abbildung 5.8: Übertragung der Informationsmeldung von Member-Section zu Kundenansicht

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.

5.3.1.3 Auftraggeberansicht

1. Mithilfe der Umbraco-Member-Section kann der Auftraggeber seine Kunden filtern, nach dem Name suchen oder löschen, etwa nach vorgegebenen Einstellungen wie List-View[16]. Es können auch zusätzliche Properties erstellt werden. Diese lassen sich können über ein Extra-Attribut in der Model-Datei mit dem Frontend verbinden. Hier werden die neuen Tabs und Properties erstellt. Als Beispiel wird hier die Tab-„Datenuebersicht“ gezeigt. Dort befindet sich eine Property „Ort“. Im Tab-Membership ist „Password“ ein zusätzliches Beispiel. Davon können mehrere eingebaut werden. Die Abbildung 5.10 zeigt diesen Tab und die zugehörige Property dar.

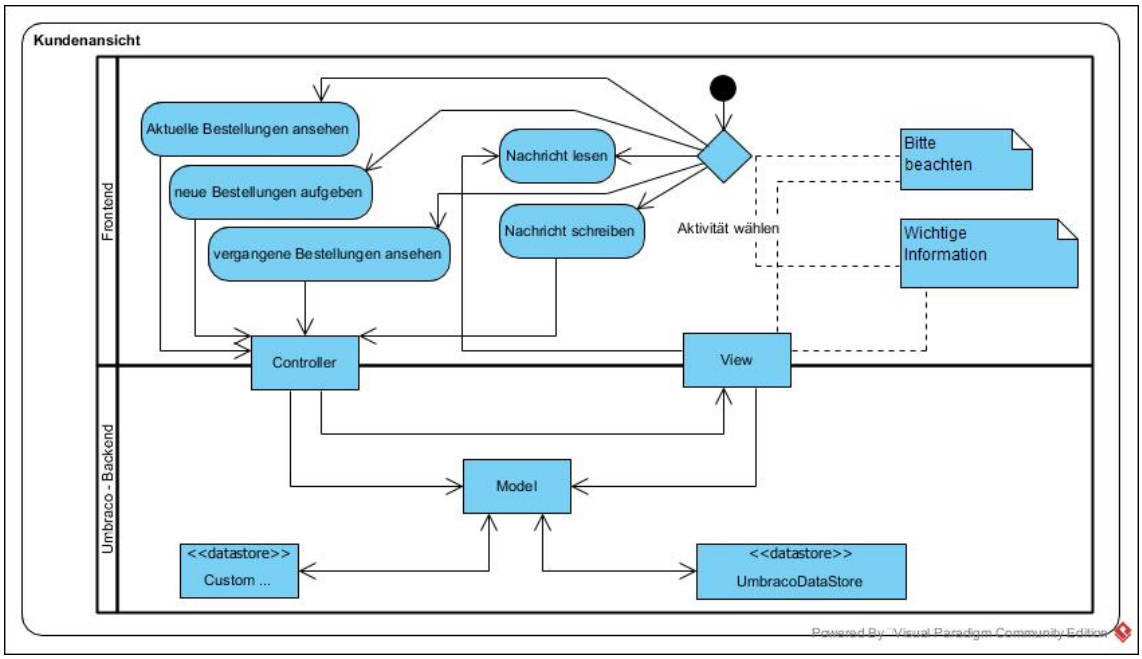


Abbildung 5.9: Funktionalität von Kundenansicht

Abbildung 5.10: Beispiel zur Erstellung eines extra Tab und Property

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

Wie bereits erläutert, wurde ein Makro erstellt, durch das die Information dem jeweiligen Kunden zugeordnet ist. Um die Migration problemlos durchführen zu können, muss eine neue Zuordnung im Member-Bereich hergestellt werden. Die neuen Properties müssen mit den alten übertragenen Attributen der Member-Datenbank übereinstimmen. Dies wird im Unterkapitel „Übersicht“ erläutert.

5.3.1.4 Kommunikation

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

Dafür wird eine ContentAPI [12] verwendet. In der Kundenansicht kann der Kunde über ein View-Nachricht-Formular Nachrichten verschicken. Mithilfe des Models und des SurfaceController wird die Nachricht in der Umbraco-Content-Section erstellt. Dort wird sie als neue Untertreenode im vorgesehenen Tree „Nachrichten“ positioniert. Dieser Tree wird mithilfe von Umbraco-ListView modelliert. Umbraco-ListView ermöglicht beliebig viele Untertreenodes, die in einem einzigen Tree gelagert werden. Um die Nachrichten dem zugehörigen Kunden zuzuordnen, wird in die jeweilige Nachricht die IdNummer

des Kunden integriert. Mithilfe dieser Nummer kann auch ein Filter erstellt werden. Somit kann die Kunde nur seine privaten Nachrichten lesen. Das Listing 5.3 und die Abbildungen 5.11 und 5.12 zeigen den Verlauf der Kommunikation.

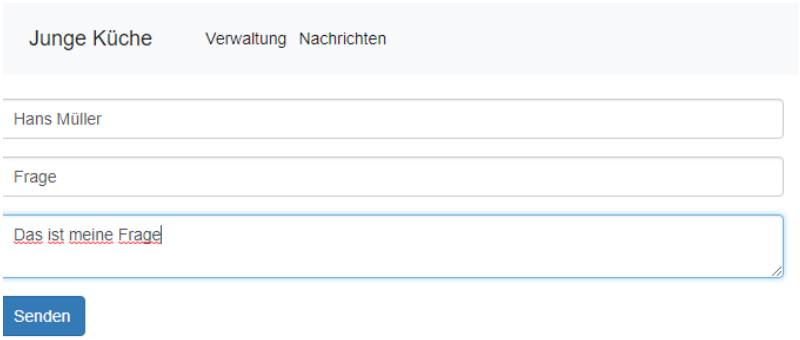


Abbildung 5.11: Nachricht Formular

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();
```

```
}
}
}
```

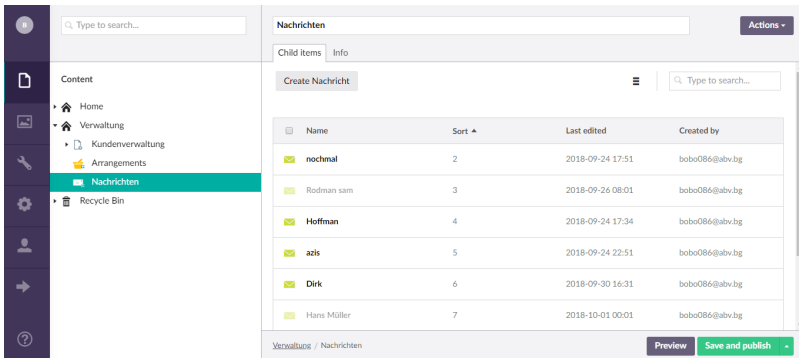


Abbildung 5.12: Nachricht in Umbraco-Backoffice

Wenn die Nachricht noch nicht gelesen wurde, ist sie hellgrau markiert In dem geöffneten Fenster wird die gesendete Nachricht angezeigt. Daneben befindet sich das „Antwort“-Tab, über das der Auftraggeber die Frage beantworten kann. Die Nachrichten werden nach memberID gefiltert. In den Abbildungen 5.13 und 5.14 wird die Rückfrage dargestellt.

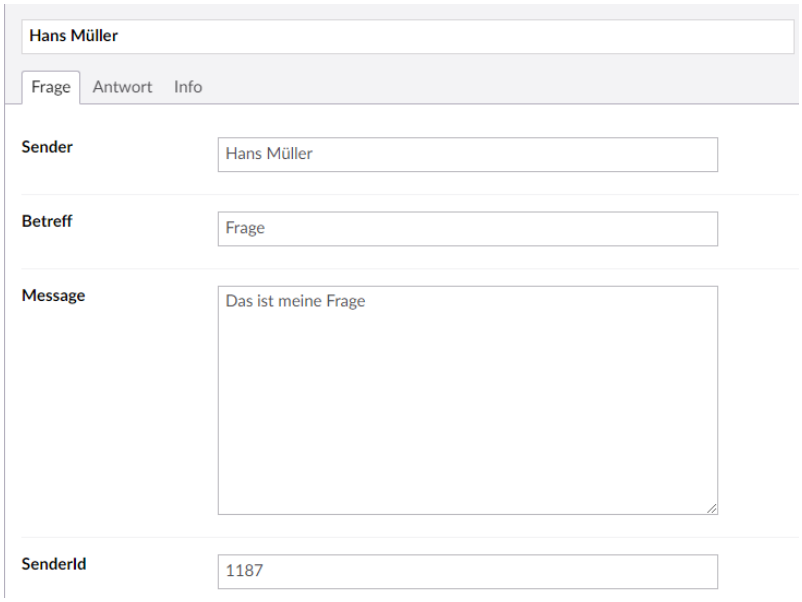


Abbildung 5.13: Die Nachricht gelesen

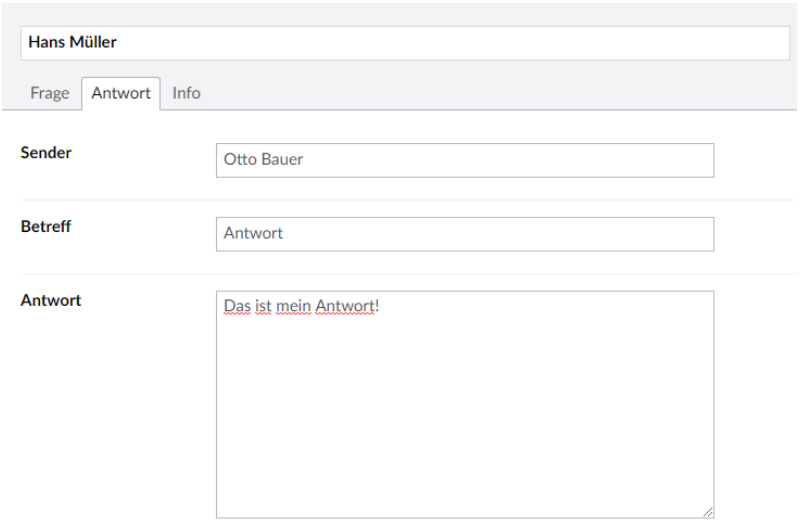


Abbildung 5.14: Antwort Schreiben

Im Listing 5.4 ist der Filter, der 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>
```

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

Abbildung 5.15: Antwort erhalten

Abbildung 5.16: Alle Nachrichten

5.3.2 Artikelverwaltung

5.3.2.1 Erfassen, editieren und löschen

Nach der vorgegebenen Aufgabe muss der Online-Editor zwei Kategorien enthalten – Arrangements und Artikel-Standard. Diese müssen mit den zugehörigen Webseiten gekoppelt werden. Eine Anforderung lautet, dass die Artikel-Verwaltung einfach funktionieren muss und die Artikel, die sich in der Webseite befinden, direkt mit den Artikeln, die in der Umbraco-Content-Section stehen, gekoppelt sein müssen. Auf der alten Seite ist die Artikel-Verwaltung nicht mit den zugehörigen Seiten gekoppelt. Die Migration der Artikel ist nicht gefordert.

Das entsprechende Ziel wird über Makros, die PetaPoco-Datenbank [9] und die von Umbraco vorgegebenen Funktionen erreicht. Zuerst wird Umbraco ListView verwendet, damit sich die erstellten Artikel in der Umbraco-Content-Section unter einer gemeinsamen Unteroption befinden. Außerdem ist vorgegeben, dass die neu erstellten Artikel direkt editiert werden können. Abbildung 5.17 zeigt, wie die Artikel-Verwaltung aussieht.

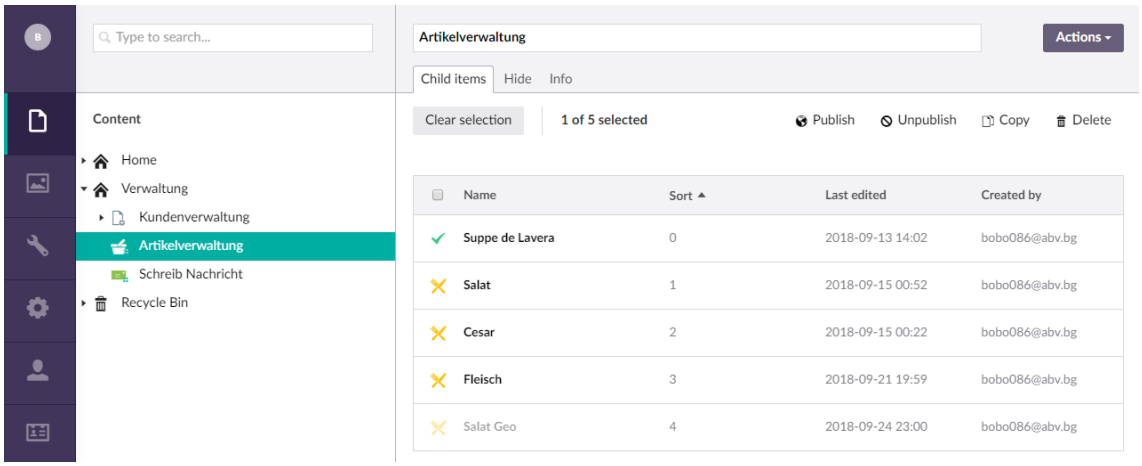


Abbildung 5.17: Übersicht der Artikel-Verwaltung

Wenn der Artikel „unpublish“ ist, kann er nicht im Frontend verwendet werden. Wenn die Artikel bereits erstellt wurden, wird ein Quellcode im Makro programmiert. Dieses Makro wird in das oben erwähnte Grid integriert. Dadurch wird der Inhalt der Artikel zum Frontend übertragen. Der Inhalt wird nicht nur mit den Webseiten, sondern auch mit dem Shop-Menü gekoppelt. Damit erhält der Auftraggeber mehr Selbständigkeit. In der nächsten Abbildung wird gezeigt, wie mit der Erstellung eines Artikels seine Daten sowohl an die zugehörige Seite als auch an das Shop-Menü gekoppelt werden. Wenn

der Kunde registriert ist, kann er die von dem Auftraggeber eingegebenen Artikelaus dem Shop wählen. Nach der Bestellung werden die Daten in der Datenbank „Artikel“ gespeichert. Gleichzeitig werden diese Daten zusammen mit den Kunden-Daten auch in der Datenbank „Auftraege“ gespeichert. In Abbildung 5.18 ist zu sehen, dass die Daten der Artikelverwaltung erfolgreich zum Shop gesendet wurden.

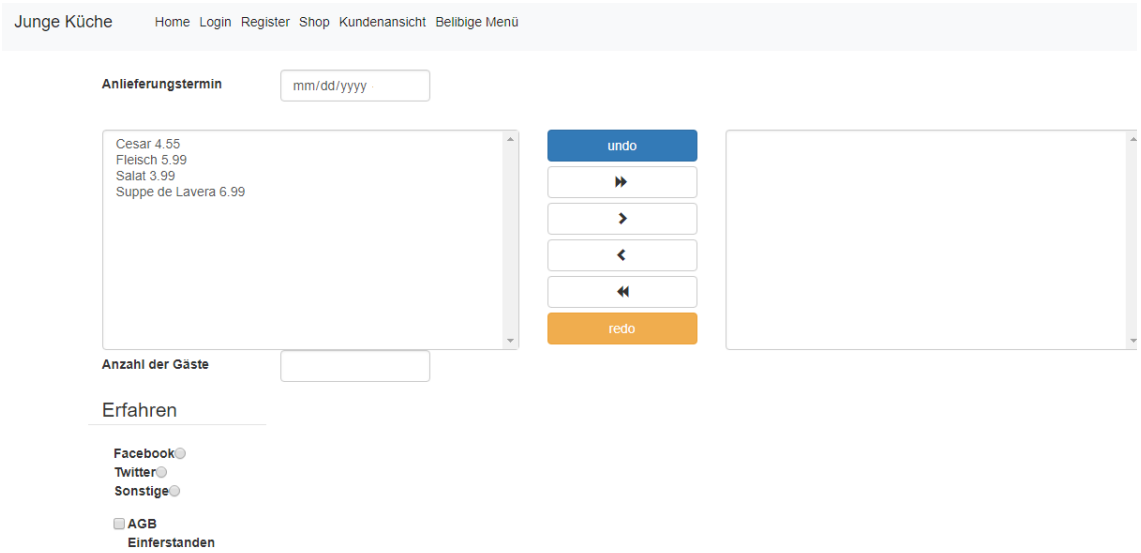


Abbildung 5.18: Übersicht des Shops

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. Das vorgeschlagene Konzept umfasst die Migration der Access-Datenbankdatei und die Erstellung einer neuen Custom-Section.

1. Um eine neue Section aufzubauen, wird zunächst eine Klasse-Datei benötigt, mit der Methoden aus den Bibliotheken „umbraco.buisnesslogic“ und „umbraco.interfaces“ aufgerufen werden. Über die „Application“-Methode und das „IApplication“-Interface wird die Custom-Section erstellt. Danach wird die Übersicht über den AngularJS-Controller und HTML erstellt. Diese Dateien werden zum API-Controller über das Paket-Manifest referenziert. In diesem Controller werden die eigentlichen Funktionalitäten der Custom-Section realisiert. Mithilfe des API-UmbracoAuthorizedJsonController werden die Daten über die Model-Attribute der bereits erstellten und im vorherigen Unterkapitel erwähnten Datenbank „Auftraege“ zur Verfügung gestellt. Die Datenbanken werden über PetaPoco erstellt. In der Abbildung 5.19 wird die Kommunikation gezeigt.

2. Wie gefordert, soll eine Migration der Access-Datenbank zu Umbraco durchgeführt werden. Das angestrebte Ziel ist, dass die alten Daten in der neuen zugehörigen Datenbank gespeichert werden. Die neuen betrachteten Datenbanken sind Custom-„Artikel“ und „Umbraco-Member“. Nach grundlegender Recherche wurde ein Konzept erstellt. Grundsätzlich stehen in der Umbraco-Member-Section [8] drei Unteroptionen zur Verfügung – Members, Member Types und Member Groups. In Member Type werden die Einstellungen (Properties) des Members aufgebaut. Dort wird ein neuer Member Type mit extra Properties erstellt. Das folgende Konzept geht davon aus, dass die Datenbanken von

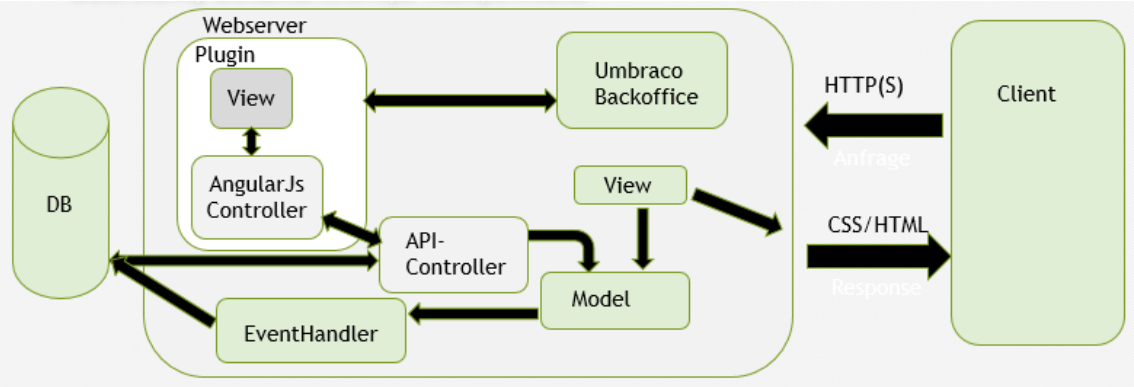


Abbildung 5.19: Eine Übersicht von Kommunikation zwischen verschiedene Elemente

Access in CSV-Format konvertiert sind. Das bedeutet, dass die Access-Datenbankformat-Datei in eine Text-Datei umgewandelt wird und dann die Methode StreamReader genutzt werden kann. Um das vorgegebene Ziel zu realisieren, muss ein SurfaceController erstellt werden, der die Übertragung leitet. Umbraco verfügt über spezielle ‚Autobahnen', durch die es möglich ist, festgelegte Sections in Umbraco zu manipulieren. Diese werden „Services" genannt und über ApplicationContext aufgerufen. In diesem Fall wird „MemberService" genutzt. Damit die Datei gelesen werden kann, wird die Funktion StreamReader verwendet. Damit wird die gelesene Datei in einer Variablen gespeichert. Der Inhalt der Datei wird von diesen Variablen mithilfe von dem Befehl-split zu einem Array gespeichert. Jeder Teil des Array wird über MemberService-Befehle und -Methoden in Umbraco Members gespeichert. Im Anhang A.6 ist der Quellcode nachzulesen und in der Abbildung 5.20 ein Schema zur Migration.

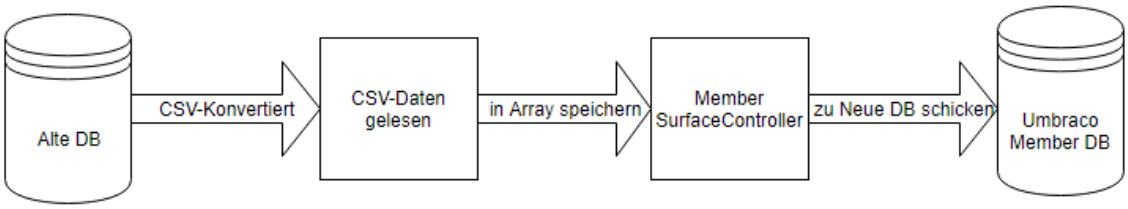


Abbildung 5.20: An neue Datenbank schicken

Wenn die Kundendaten schon übertragen wurden, kann dasselbe Prinzip der Migration für die Artikel verwendet werden. Hier wird als Ziel anstelle der MemberService- Datenbank die Artikel-Datenbank verwendet. Nach der Umwandlung der Access-Datenbankdatei in das CSV-Format wird diese Datei über die Methode StreamReader gelesen, danach wird die Information in einem Array gespeichert. Über den API-Controller werden die Daten in die Artikel-Datenbank übertragen. In Listing 5.5 wird anschaulicher gezeigt, wie die Übertragung der Daten abläuft.

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 gefordert, dass der Auftraggeber die Aufträge bearbeiten kann, den Status ändern, Positionen editieren, hinzufügen oder löschen, dem Kunden Freigaben erteilen (zum Beispiel das Aussuchen der Positionen) und eine Rechnungsnummer vergeben. Diese Aktivitäten sind dieselben wie auf der alten Webseite und müssen in einer eigenen Umbraco-Section umgesetzt werden. Die Erstellung einer Custom-Section ist bereits bekannt. Hier wird sie mit der Auftraege- und der Member-Datenbank über AngularJS-Controller anhand des Package-Manifest mit dem API-UmbracoAuthorizedJsonController verbunden. In den Listings 5.6 und 5.7 wird gezeigt, wie die Daten aus 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 aus der Member-Datenbank werden als String über MemberService aufgerufen und in ein Array gespeichert. Dieses Array wird von der AngularJS aufgerufen und an den HTML-Plugin-Editor geschickt. Die Daten des Members können in der Detailansicht nicht geändert werden, daher werden sie als String verwendet.

6 Schluss

6.1 Zusammenfassung

Ziel dieser Arbeit war es, eine veraltete webbasierte Software zu analysieren, mit der eine Migration zu einem neuen System durchgeführt werden sollte. Hierbei lag das Augenmerk auf der Problemerkennung und -darstellung anhand der Analyse und der Anforderungen sowie auf der Problemlösung. Dazu wurde zunächst die allgemeine Aufgabenstellung definiert und eingegrenzt. Die erstellten Anwendungsfälle haben einen Überblick über die zu bearbeitenden Aufgaben gegeben. Eine IST-Analyse wurde durchgeführt, um in einer Detailansicht jeden Punkt der Software zu betrachten. Danach wurden Werkzeuge angewendet, um eine Bewertung dieser Punkte durchzuführen. Es wurde festgestellt, welche Anforderungsmerkmale für das neue Websystem gelten und welche alten Funktionalitäten beibehalten werden sollten. Als Basis für die neue Software wurde Umbraco CMS ausgewählt. Die neue erstellten Datenbanken mussten an die alten angepasst werden, damit eine reibungslose Migration durchgeführt werden konnte.

Im Konzept wurden die allgemeinen Anforderungsmerkmale konkretisiert. Ein Entwurf der fachlichen und technischen Architektur wurde für jeden zugehörigen Teil der vorgegebenen Anforderungen festgelegt, ebenso wie die Implementierung. Die Daten der Kunden wurden erfolgreich in der Member-Section von Umbraco gespeichert, womit der Auftraggeber die Kundendaten editieren oder löschen kann. Auch die Möglichkeit für den Auftraggeber, private Information an die Kunden zu senden, wurde gegeben. Das Konzept zur Migration wurde theoretisch erstellt. Der Prozess erfolgte durch Konvertierung der Datenbankdateien in eine CSV-Datei und anhand des SurfaceController wurden die Dateien in die neue Datenbank übertragen. Weitere Tests der Software und Verbesserungen hinsichtlich der Kommunikation, der Datenbankverbindungen und der Artikelverwaltung sind zu empfehlen. Bei der Bewertung der Implementierung hat sich gezeigt, dass die wichtigsten funktionalen Anforderungen erfüllt sind und sich das System leicht durch zusätzliche Funktionen erweitern lässt.

6.2 Ausblick und Kritik

Der Funktionsumfang der entwickelten Software bietet dem Auftraggeber eine flexible und unkomplizierte Verwaltungsmöglichkeit. Einige Merkmale, die aus den Anforderungen ermittelt wurden, wie die Verbindung von Datenbank zum Frontend, fehlen teilweise noch. Eine Implementierung dieser Komponenten kann vorgenommen werden, da diese leicht ins Konzept zu integrieren sind. Zu diesen Merkmalen zählen auch die Verbindungen zur noch nicht entwickelten E-Mail- und Umsatzverwaltung. Das Kommunikationskonzept wurde erfolgreich entwickelt, aber aufgrund einer Erwartung, dass die Kommunikation im Member-Bereich sich befinden musste, wurde das Konzept zurückgewiesen. Aufgrund von fehlenden verbundenen Datenbanken funktionieren die Artikelverwaltung, die Übersicht und die Detailansicht nicht. Eine Erläuterung der E-Mail- und der Umsatzverwaltung konnte aus Zeitgründen nicht erfolgen.

Literatur

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

[2] Thomas Beckert. *.Net Webkonzepte und Werkzeuge*. Script 11. Juli 2017.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[21] droom.biz. *E Member To Rule Them All Umbraco Architecture Diagram*. Sep. 2018. URL: http://droom.biz/umbraco-architecture-diagram/e-member-to-rule-them-all-umbraco-architecture-diagram/.

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

CSP Content Security Policy

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.

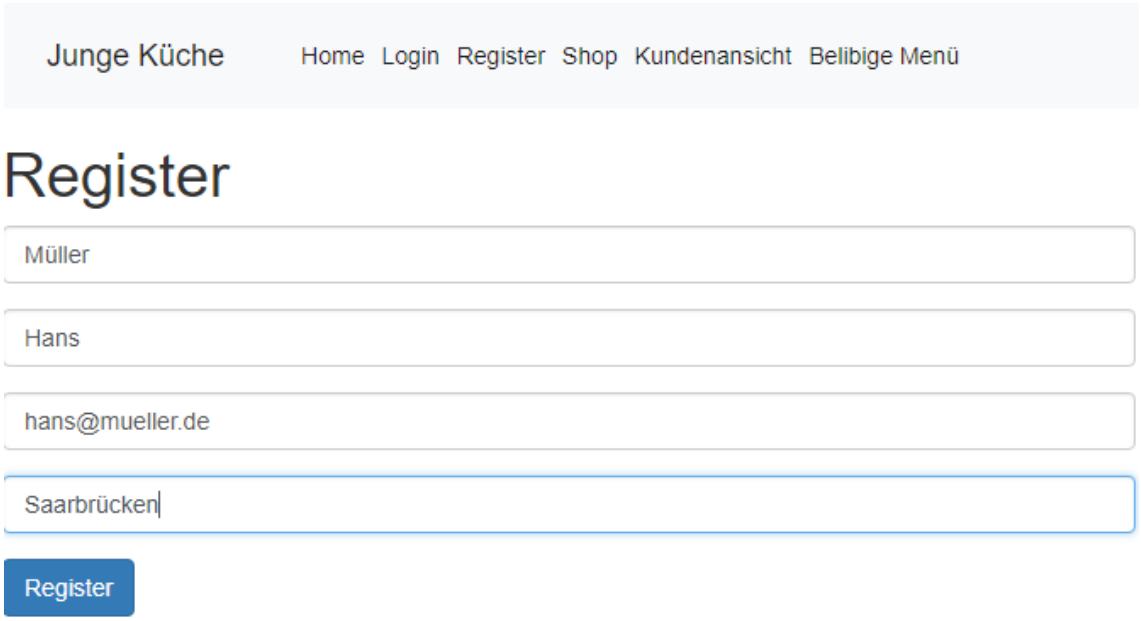


Abbildung A.1: Der Kunde gibt seine Daten ein.

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;

}
}
}
```

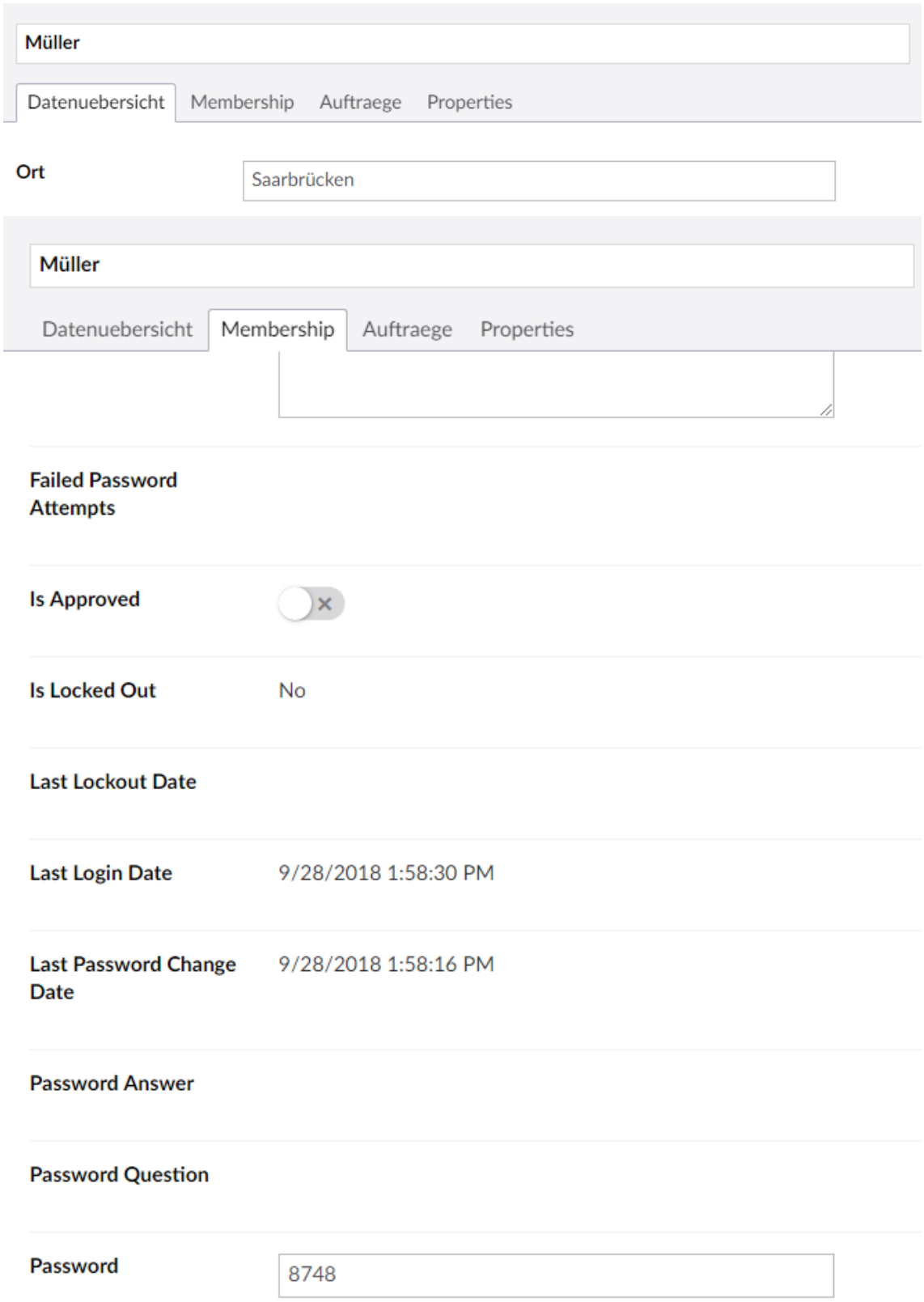


Abbildung A.2: Die Kunde im Members gespeichert.