Diplomarbeit

Höhere Technische Bundeslehranstalt Leonding
Abteilung für Informatik

AWO - Administration and Website for Opticians

Eingereicht von: **Eva Pürmayr, 5AHIF**
Danijal Orascanin, 5AHIF
Datum: **April 4, 2018**
Betreuer: **Michael Bucek**
Projektpartner: **Augenoptik Aigner**

Declaration of Academic Honesty

Hereby, I declare that I have composed the presented paper independently on my own and without any other resources than the ones indicated. All thoughts taken directly or indirectly from external sources are properly denoted as such.

This paper has neither been previously submitted to another authority nor has it been published yet.

Leonding, April 4, 2018

Eva Pürmayr, Danijal Orascanin

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorgelegte Diplomarbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Gedanken, die aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Die Arbeit wurde bisher in gleicher oder ähnlicher Weise keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Leonding, am 4. April 2018

Eva Pürmayr, Danijal Orascanin

Zusammenfassung

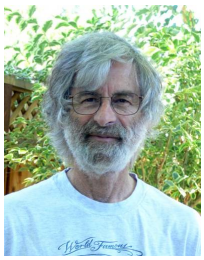
Here it is described what the thesis is all about. The abstract shall be brief and concise and its size shall not go beyond one page. Furthermore it has no chapters, sections etc. Paragraphs can be used to structure the abstract. If necessary one can also use bullet point lists but care must be taken that also in this part of the text full sentences and a clearly readable structure are required.

Concerning the content the following points shall be covered.

- *Definition of the project:* What do we currently know about the topic or on which results can the work be based? What is the goal of the project? Who can use the results of the project?
- *Implementation:* What are the tools and methods used to implement the project?
- *Results:* What is the final result of the project?

This list does not mean that the abstract must strictly follow this structure. Rather it should be understood in that way that these points shall be described such that the reader is animated to dig further into the thesis.

Finally it is required to add a representative image which describes your project best. The image here shows Leslie Lamport the inventor of LaTeX.



Zusammenfassung

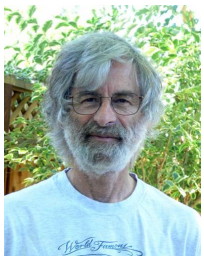
An dieser Stelle wird beschrieben, worum es in der Diplomarbeit geht. Die Zusammenfassung soll kurz und prägnant sein und den Umfang einer Seite nicht übersteigen. Weiters ist zu beachten, dass hier keine Kapitel oder Abschnitte zur Strukturierung verwendet werden. Die Verwendung von Absätzen ist zulässig. Wenn notwendig, können auch Aufzählungslisten verwendet werden. Dabei ist aber zu beachten, dass auch in der Zusammenfassung vollständige Sätze gefordert sind.

Bezüglich des Inhalts sollen folgende Punkte in der Zusammenfassung vorkommen:

- *Aufgabenstellung:* Von welchem Wissenstand kann man im Umfeld der Aufgabenstellung ausgehen? Was ist das Ziel des Projekts? Wer kann die Ergebnisse der Arbeit benutzen?
- *Umsetzung:* Welche fachtheoretischen oder -praktischen Methoden wurden bei der Umsetzung verwendet?
- *Ergebnisse:* Was ist das endgültige Ergebnis der Arbeit?

Diese Liste soll als Sammlung von inhaltlichen Punkten für die Zusammenfassung verstanden werden. Die konkrete Gliederung und Reihung der Punkte ist den Autoren überlassen. Zu beachten ist, dass der/die LeserIn beim Lesen dieses Teils Lust bekommt, diese Arbeit weiter zu lesen.

Abschließend soll die Zusammenfassung noch ein Foto zeigen, das das beschriebene Projekt am besten repräsentiert. Das folgende Bild zeigt Leslie Lamport, den Erfinder von LaTeX.



Acknowledgments

If you feel like saying thanks to your grandma and/or other relatives.

Inhaltsverzeichnis

Kapitel 1

Introduction

1.1 Initial Situation

Common word processors do not prepare print-like documents in so far as these programs do not reflect the rules of professional printing which have been grown over centuries. These rules contain clear requirements for balancing page layouts, the amount of white space on pages, font-handling, etc. Donald Knuth's TeX package (see [**?**]) is a word processor which conforms to these printing rules. This package was enhanced by Leslie Lamport by providing more text structuring commands. He called his package LaTeX [**?**].

When preparing a thesis, we want not only to have our content on a top level, we also want to commit to a high level of formal criteria. Therefore, we request our students to use one of these professional printing production environments like TeX or LaTeX.

Furthermore students should train their scientific writing skills. This includes a clear and structured break-down of their ideas, a high-level and clear wording, and the training of transparent citations of ideas from other sources than from theirs. A good source for more information concerning technical and scientific writing can be found in [**?**].

1.2 Goals

The general goals and objectives of the project are described here. Care must be taken that the goals documented here are purely project goals and have nothing to do with individual goals of the team members. If individual goals should be part of the thesis they are listed in appendix B.

1.3 Overview

Details of the diploma thesis have to be aligned between student and supervisor. This should be a basic structure to facilitate the first steps when students start to write their theses.

Abbildung 1.1: Don Knuth, the inventor of TEX

Never forget to add some illustrative images. Images must not be messed up with your normal text. They are encapsulated in floating bodies and referenced in your text. An example can be seen in figure 4.16. As you can see, figures are placed by default on top of the page nearby the place where they are referenced the first time. Furthermore you can see that a list of figures is maintained automatically which can be included easily by typing the command `\listoffigures` into your document.

1.4 Basic Terminology

As usual the very basic terminology is briefly explained here. Most probably the explanations here only scratch a surface level. More detailed explanations of terminology goes into chapter 5.

1.5 Related Work and Projects

Here a survey of other work in and around the area of the thesis is given. The reader shall see that the authors of the thesis know their field well and understand the developments there. Furthermore here is a good place to show what relevance the thesis in its field has.

1.6 Structure of the Thesis

Finally the reader is given a brief description what (s)he can expect in the thesis. Each chapter is introduced with a paragraph roughly describing its content. 5 Online Referenz: [Wik17]

Kapitel 2

Hintergründe

2.1 Ist-Situation

Im Unternehmen “Augenoptik Aigner” sind zur Zeit nur wenige Rechner im Einsatz, welche alle mit dem Betriebssystem “Windows 95” funktionieren. Auf diesen existiert ein Dos-Programm welches Kunden, Aufträge, Lieferanten und lagernde Produkte verwaltet. Weil mehrere Rechner im Einsatz sind, muss die aktuelle Version des Programms immer auf den jeweiligen Rechner kopiert werden, auf dem man dann gerne arbeiten würde. Dies ist natürlich sehr umständlich und zeitaufwendig, weshalb es Zeit wird, alle Rechner auf ein aktuelles Betriebssystem zu aktualisieren und eine zentrale Datenbank einzurichten, damit alle Rechner immer synchronisiert sind.

Kapitel 3

Verwendete Technologien

The details of the structure of your thesis have to be aligned with the supervising teacher. However, most of the theses require to have some description of the models used or some other theoretical background necessary to understand the rest of the text.

3.1 Entity Framework

Zur Erstellung einer Datenbank wurde Entity Framework 4.6.1 benutzt. Dabei handelt es sich um ein Framework zur Erstellung von objektrelationalen Abbildungen (ORM – object relational mapping) auf .NET-Objektstrukturen. Dieses Framework wurde von Microsoft entwickelt.

Kapitel 4

Beschreibung des Endproduktes

4.1 Administration

4.1.1 Kundenverwaltung

Um dem Benutzer eine kompakte Übersicht über seine Kunden zu geben, gibt es die Kundenverwaltung, bei der alle Kunden in einer Liste dargestellt werden. Angezeigt wird, die Id, Titel, Vorname, Nachname, Straße, PLZ, Ort, Telefon1 und das Land des Kunden. Diese Liste kann man filtern und sortieren (siehe Kapitel 'Filtern und Sortieren'). Zusätzlich zu dem normalen Filter, kann man bei der Kundenverwaltung ebenso gelöschte Kunden ein- oder ausblenden. Der Grund dafür wird später noch näher erklärt (Referenz)

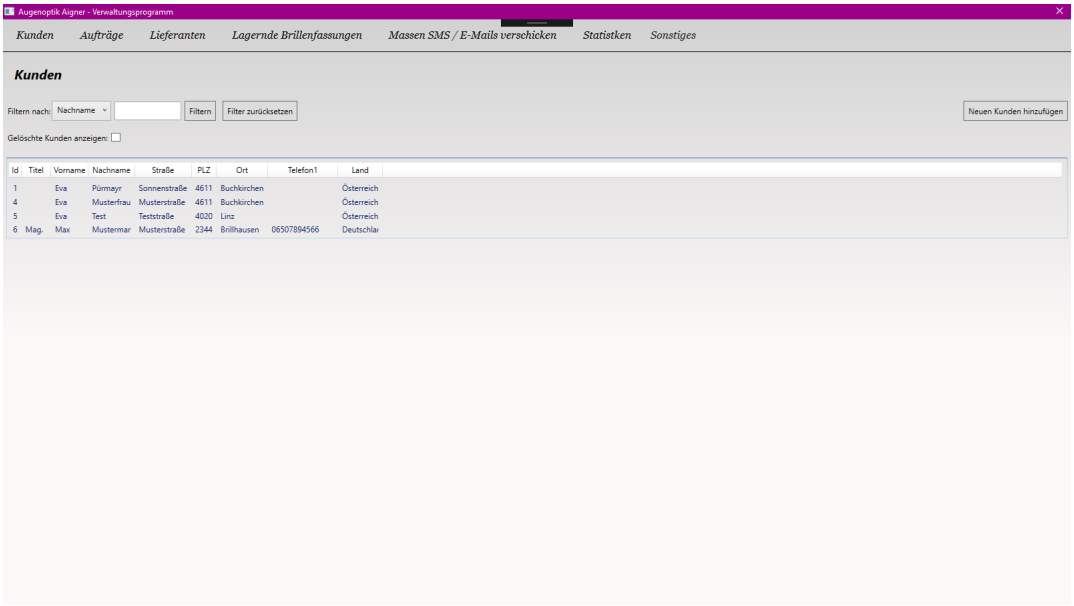


Abbildung 4.1: Screenshot der Kundenverwaltung

Beim Klick des Buttons Neuen Kunden hinzufügenërscheint ein neues Fenster, welches einen neuen Kunden erstellt. Hier kann der Benutzer noch mehr Daten eingeben, wie beispielsweise Hobbies, Job oder den Geburstag. Außerdem kann der Benutzer den Ort und das Land aus einer Drop-Down-List auswählen. Falls der Ort bzw. das Land noch nicht vorhanden ist, kann der Benutzer auf den danebenliegenden Knopf drücken und einen neuen Ort/Land anlegen.

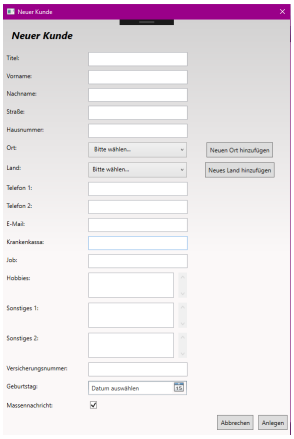


Abbildung 4.2: Screenshot Neuen Kunden anlegen

Falls der Benutzer nach dem Anlegen des Kunden noch Änderungen vornehmen möchte, kann er dies auf der Startseite durch einen Doppelklick auf den gewünschten Kunden erledigen. Dadurch erscheint ein neues Fenster, auf welchem der Benutzer alle Daten des Kundes bearbeiten kann und zusätzlich alle Bestellungen des Kundens sieht. Außerdem besteht hier die Möglichkeit den Kunden zu löschen. Damit ist allerdings aus datenbanktechnischen Gründen gemeint, den Kunden nicht mehr bearbeitbar zu machen allerdings kann der Benutzer keinen Kunden wirklich löschen. Der Grund dafür ist, dass jede Bestellung in der Datenbank auf einen Kunden verweisen muss und wenn ein Kunde bereits mehrere Bestellungen getätigt hat und danach den Kunden löschen möchte, müssten alle seine Bestellungen mitgelöscht werden.

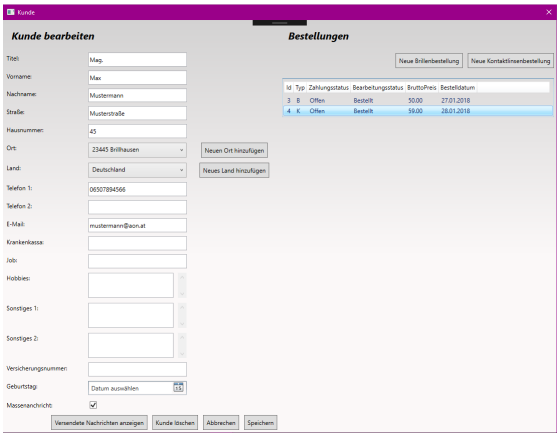


Abbildung 4.3: Screenshot der Kundetails

Technischer Hintergrund: Damit die Kunden auf der Startseite angezeigt werden können, müssen sie zuerst aus der Datenbank in eine ObservableCollection vom Typ Customer kopiert werden.

```
private ObservableCollection<Customer> GetAllCustomers()
       {
           var unitofWorkCustomers =
               this.Uow.CustomerRepository.Get().ToList();
           ObservableCollection<Customer> copiedCustomers = new
               ObservableCollection<Customer>();
           foreach (var item in unitofWorkCustomers)
           {
               Customer customer = new Customer();
               GenericRepository<Customer>.CopyProperties(customer, item);
               if (item.Town_Id != null)
               {
                   Town town = new Town(); //Referenced town must be copied as
                       well
                   GenericRepository<Town>.CopyProperties(town,
```

```
                this.Uow.TownRepository.GetById(item.Town_Id));
            customer.Town = town;
        }
        if (item.Country_Id != null)
        {
            Country country = new Country();
            GenericRepository<Country>.CopyProperties(country,
                this.Uow.CountryRepository.GetById(item.Country_Id));
            customer.Country = country;
        }
        copiedCustomers.Add(customer);
    }
    return copiedCustomers;
```

Danach wird auf Basis der kopierten Datensätze eine ICollectionView erstellt, welche die Daten dann anzeigt. Im Vergleich zur ObservableCollection bietet die ICollectionView beim Anzeigen viele Vorteile (siehe Kapitel Filtern und Sortieren). Damit ein Kunde gelöscht werden kann, enthält die Klasse Kunde eine Property namens 'Deleted', welche angibt, ob der Kunde gelöscht wurde. Wenn diese auf 'true' gesetzt wird, kann der Benutzer den Kunden durch die Checkbox auf der Startseite ausblenden. Falls er dies nicht tut, wird der Kunde auf der Startseite angezeigt, allerdings erscheint eine Fehlermeldung, wenn der Benutzer versucht die Detailsseite des Kunden zu öffnen. Dadurch ist es auch unmöglich neue Bestellungen für diesen Kunden anzulegen oder die Daten des Kunden zu bearbeiten. Die Bestellungen des gelöschten Kunden werden trotzdem normal angezeigt.

4.1.2 Aufträge

Grundsätzlich gibt es zwei Arten von Aufträgen: Brillen und Kontaktlinsen. Eigentlich haben beide Arten diesselben Eigenschaften, nur der Brillenauftrag hat eine Brillenfassung und der Kontaktlisenauftrag nicht. Außerdem hat ein Brillenauftrag einen Brillentyp und ein Kontaktlinsenauftrag einen Kontaktlinsentyp. Damit kann der Benutzer beispielsweise unterscheiden, ob es sich um eine Fern- oder Nahbrille handelt. Generell wird streng zwsichen Brilen- und Kontaktlinsenaufträgen unterschieden, deshalb werden unter dem Menüpunkt "Aufträge"auch zwei verschiedene Listen angezeigt. Wie gewohnt kann man diese Listen wieder filtern und sortieren. Allerdings kann der Benutzer von dieser Sicht aus keine neuen Aufträge erfassen, dazu muss er in der Kundenverwaltung zuerst einen Kunden auswählen.

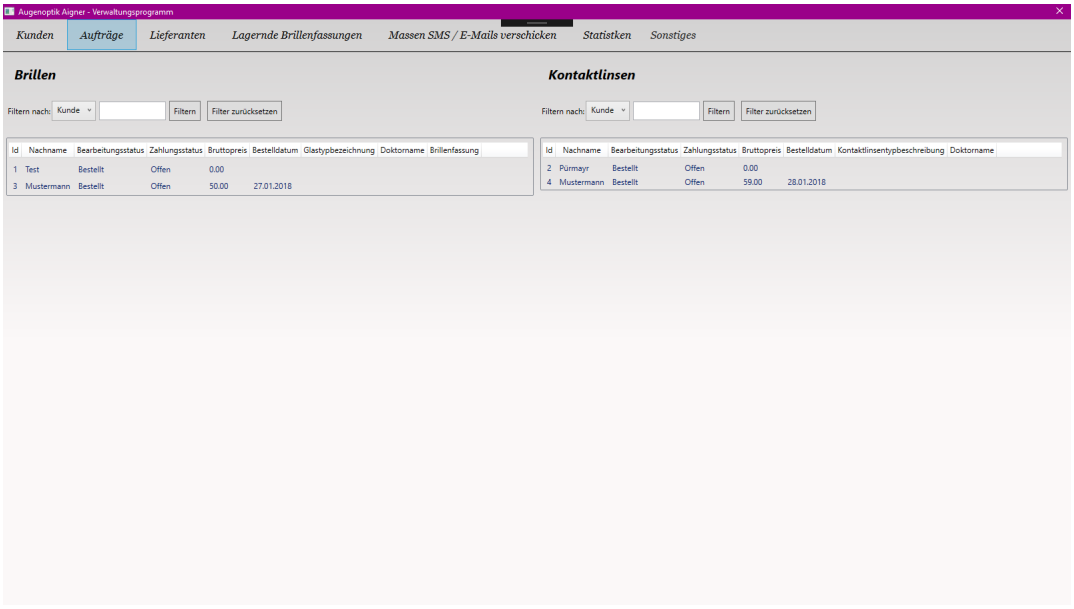


Abbildung 4.4: Screenshot der Auftragverwaltung

Wenn der Benutzer doppelt auf einen Auftrag klickt, erscheint entweder ein Detailsfenster eines Brillenauftrags oder Kontatklinsenauftrags. Im rechten Bereich des Fensters kann der Benutzer die einzelnen Preise der Komponenten angeben. Das Programm rechnet alle Preise brutto und gibt zum Schluss die darin enthaltene Mehrwertsteuer an. Es wird nach folgender Formel gerechnet: Zuerst werden linker und rechter Glaspreis, Preis von Sonstigem und wenn vorhanden der Verkaufspreis der Brillenfassung addiert. Davon wird das Krankenkassageld abgezogen, der Selbstbehalt hinzugezählt und der Rabatt (wird in Euro angegeben) wieder subtrahiert. Zwanzig Prozent davon sind schlussendlich die Mehrwertsteuer. Im unteren Bereich des Fensters kann der Benutzer Details zur Glasverarbeitung angeben. Wenn der Benutzer den Bearbeitungsstatus auf "Abgeholt" setzt, wird auch der Status der Brillenfassung auf "Verkauft" gesetzt

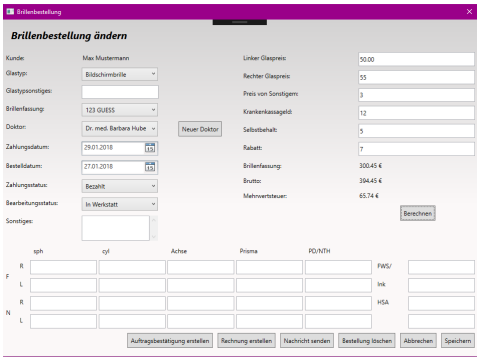


Abbildung 4.5: Screenshot eines Brillenauftrags

Dokumente erstellen

Unter den Angaben der Details zu den Gläsern, kann der Benutzer eine Auftragsbestätigung oder eine Rechnung erstellen. Die Dokumente werden als Word-Dokumente in einem Ordner abgespeichert. Das hat den Vorteil, dass der Benutzer selbst entscheiden kann, ob er noch etwas nachträglich ändert oder die Vorlage gleich ausdruckt oder versendet. Damit das funktioniert, muss der Benutzer eine Word-Vorlage erstellen, die Felder enthält (MergeFields), welche das Programm ersetzen kann. In der Datenbank wird immer nur der Pfad zur jeweiligen Auftragsbestätigung/Rechnung gespeichert. Das heißt, dass der Benutzer die erstellten Dokumente nach der Erstellung bearbeiten, löschen oder neu generieren kann. Der Dokumentname der generierten Datei besteht immer aus der Bestell-Id, dem Dokumenttyp (Rechnung oder Auftragsbestätigung), dem Nachnamen des Kunden und dem aktuellen Datum.


Augenoptik
AIGNER

Max Mustermann
Musterstraße 45
23445 Brillhausen

01.02.2018

Auftragsbestätigung

Ihre Bestellung/Ihr Auftrag vom 27.01.2018

Sehr geehrter Kunde,

vielen Dank für Ihren Auftrag (Nr. 3). Wir haben den Auftrag erhalten und werden uns so bald als möglich darum kümmern.

Bei Rückfragen stehen wir selbstverständlich jederzeit gerne zur Verfügung.

Mit freundlichen Grüßen

Augenoptik Aigner

Augenoptik Aigner
Kaiser Josef Platz 3
4600 Wels
Österreich
Tel.: 07242 22 43 84
E-Mail: office@augenoptik-aigner.at

Sparkasse OÖ
IBAN: AT64 2032 0321 0007 1003
BIC: ASPK AT2L XXX

USt.-ID: ATU 63 84 12 03
Geschäftsführer:
Wolfgang Aigner

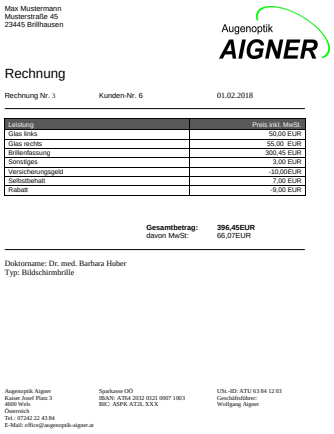
Abbildung 4.6: Generierte Auftragsbestätigung

Technischer Hintergrund: Zur Erstellung von Word-Dokumenten wird der Namespace Microsoft.Office.Interop.Word verwendet. Die Methode CreateDocument liest die Wordvorlage und ersetzt die Mergsfields. Der Parameter oTemplatePath beschreibt den Pfad der Wordvorlage und der String path den Namen, unter dem das Dokument abgespeichert werden soll. Der Code unterhalb ist nur ein Ausschnitt der Methode.

```
private static bool CreateDocument(int orderId, Object oTemplatePath, string
    path)
        {
            Application wordApp = new Application();
            Document wordDoc = new Document();
            try
            {
                Order order;
                Customer cus;

            //Ausgeschnitten: Hier werden order und cus Werte zugewiesen

                Object oMissing = System.Reflection.Missing.Value;

                wordDoc = wordApp.Documents.Add(ref oTemplatePath, ref oMissing,
                    ref oMissing, ref oMissing);

                foreach (Field myMergeField in wordDoc.Fields)
                {
                    Range rngFieldCode = myMergeField.Code;
                    String fieldText = rngFieldCode.Text;
```

```
                //only mergefields should be edited
                if (fieldText.StartsWith(" MERGEFIELD"))
                {
                    //Text comes in in format of "MERGEFILED fieldname \\*
                        MERGEFORMAT
                    //Ausgeschnitten: Hier wird der Name der Property aus dem
                        fieldText herausgeholt und auf Englisch \"ubersetzt
                    string value = String.Empty;
                    //Ausgeschnitten: Die Property wird in den Klassen
                        gesucht und der Wert gespeichert z.B: Properties der
                        Klasse Customer:
                     if (typeof(Customer).GetProperty(translatedFieldName) !=
                         null)
                    {
                        value =
                            cus.GetType().GetProperty(translatedFieldName).GetValue(cus,
                            null)?.ToString();
                    }

                    myMergeField.Select();
                    wordApp.Selection.TypeText(value);
                }
            }
            int idx = oTemplatePath.ToString().LastIndexOf("\\");
            string p = oTemplatePath.ToString().Substring(0, idx + 1);
            string completePath = p + path + ".docx";
            wordDoc.SaveAs(completePath);
            wordDoc.Close();
            wordApp.Quit();
            return true;
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
            wordApp.Quit();
            wordDoc.Close();
            return false;
        }
    }
```

Genau die selbe Mehtode wird benutzt, wenn eine Rechnung erstellt wird.



Max Mustermann
Musterstraße 45
23445 Brillhausen

Augenoptik
AIGNER

Rechnung

Rechnung Nr. 3 Kunden-Nr. 6 01.02.2018

Leistung	Preis inkl. MwSt.
Glas links	50,00 EUR
Glas rechts	55,00 EUR
Brillenfassung	300,45 EUR
Sonstiges	3,00 EUR
Versicherungsgeld	-10,00EUR
Selbstbehalt	7,00 EUR
Rabatt	-9,00 EUR

Gesamtbetrag: **396,45EUR**
davon MwSt: 66,07EUR

Doktorname: Dr. med. Barbara Huber
Typ: Bildschirmbrille

Augenoptik Aigner
Kaiser Josef Platz 3
4680 Wels
Österreich
Tel.: 07242 22 43 84
E-Mail: office@augenoptik-aigner.at

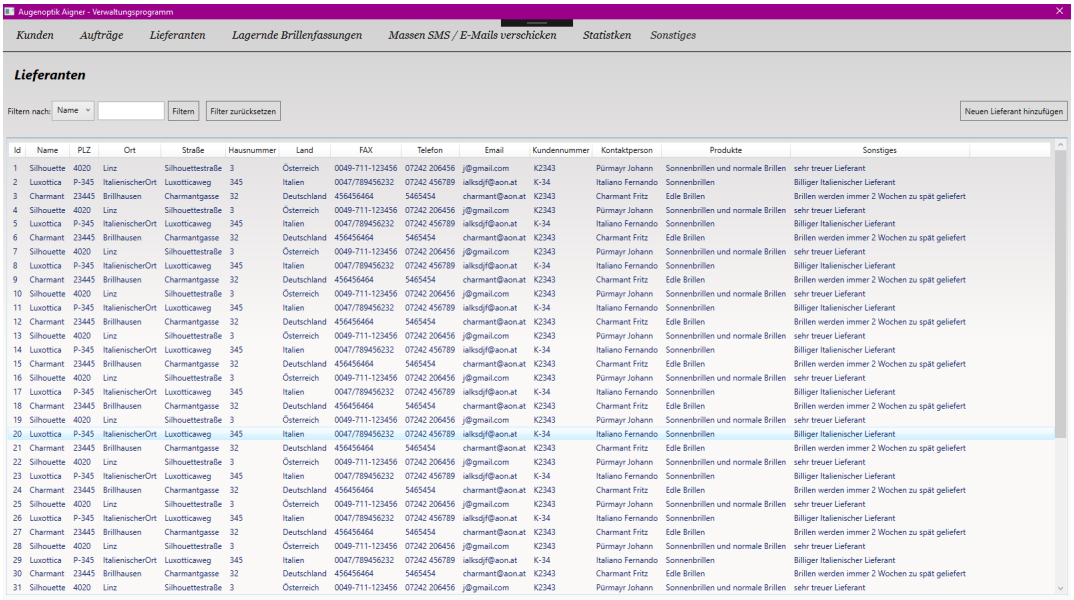
Sparkasse OÖ
IBAN: AT04 2032 0321 0007 1003
BIC: ASPK AT2I. XXX

USt.-ID: ATU 63 84 12 03
Geschäftsführer:
Wolfgang Aigner

Abbildung 4.7: Generierte Rechnung

4.1.3 Lieferantenverwaltung

Ebenso wie für die Kunden, gibt es auch eine Verwaltung für die Lieferanten des Benutzers. Auch die Lieferanten lassen sich filtern und sortieren (siehe Kapitel Filtern und Sortieren). Lieferanten haben folgende Attribute: Name, Ort, Land, Addresse, FAX, Telefon, E-Mail, Kundennummer (damit ist die Id des Benutzers beim jeweiligen Lieferanten gemeint), Kontaktperson, Produkte und Sonstiges. Einen neuen Lieferant kann man mittels dem Button links oben anlegen und Lieferanten bearbeiten und löschen kann der Benutzer durch einen Doppelklick auf den gewünschten Lieferanten.



Id	Name	PLZ	Ort	Straße	Hausnummer	Land	FAX	Telefon	Email	Kundennummer	Kontaktperson	Produkte	Sonstiges
1	Silhouette	4020	Linz	Silhouettestraße	3	Österreich	0049-711-123456	07242 206456	j@gmail.com	K2343	Pürmayr Johann	Sonnenbrillen und normale Brillen	sehr treuer Lieferant
2	Luxottica	P-345	ItalienischerOrt	Luxotticaweg	345	Italien	0047/789456232	07242 456789	ialksdjf@aon.at	K-34	Italiano Fernando	Sonnenbrillen	Billiger Italienischer Lieferant
3	Charmant	23445	Brillhausen	Charmantgasse	32	Deutschland	456456464	5465454	charmant@aon.at	K2343	Charmant Fritz	Edle Brillen	Brillen werden immer 2 Wochen zu spät geliefert
4	Silhouette	4020	Linz	Silhouettestraße	3	Österreich	0049-711-123456	07242 206456	j@gmail.com	K2343	Pürmayr Johann	Sonnenbrillen und normale Brillen	sehr treuer Lieferant
5	Luxottica	P-345	ItalienischerOrt	Luxotticaweg	345	Italien	0047/789456232	07242 456789	ialksdjf@aon.at	K-34	Italiano Fernando	Sonnenbrillen	Billiger Italienischer Lieferant
6	Charmant	23445	Brillhausen	Charmantgasse	32	Deutschland	456456464	5465454	charmant@aon.at	K2343	Charmant Fritz	Edle Brillen	Brillen werden immer 2 Wochen zu spät geliefert
7	Silhouette	4020	Linz	Silhouettestraße	3	Österreich	0049-711-123456	07242 206456	j@gmail.com	K2343	Pürmayr Johann	Sonnenbrillen und normale Brillen	sehr treuer Lieferant
8	Luxottica	P-345	ItalienischerOrt	Luxotticaweg	345	Italien	0047/789456232	07242 456789	ialksdjf@aon.at	K-34	Italiano Fernando	Sonnenbrillen	Billiger Italienischer Lieferant
9	Charmant	23445	Brillhausen	Charmantgasse	32	Deutschland	456456464	5465454	charmant@aon.at	K2343	Charmant Fritz	Edle Brillen	Brillen werden immer 2 Wochen zu spät geliefert
10	Silhouette	4020	Linz	Silhouettestraße	3	Österreich	0049-711-123456	07242 206456	j@gmail.com	K2343	Pürmayr Johann	Sonnenbrillen und normale Brillen	sehr treuer Lieferant
11	Luxottica	P-345	ItalienischerOrt	Luxotticaweg	345	Italien	0047/789456232	07242 456789	ialksdjf@aon.at	K-34	Italiano Fernando	Sonnenbrillen	Billiger Italienischer Lieferant
12	Charmant	23445	Brillhausen	Charmantgasse	32	Deutschland	456456464	5465454	charmant@aon.at	K2343	Charmant Fritz	Edle Brillen	Brillen werden immer 2 Wochen zu spät geliefert
13	Silhouette	4020	Linz	Silhouettestraße	3	Österreich	0049-711-123456	07242 206456	j@gmail.com	K2343	Pürmayr Johann	Sonnenbrillen und normale Brillen	sehr treuer Lieferant
14	Luxottica	P-345	ItalienischerOrt	Luxotticaweg	345	Italien	0047/789456232	07242 456789	ialksdjf@aon.at	K-34	Italiano Fernando	Sonnenbrillen	Billiger Italienischer Lieferant
15	Charmant	23445	Brillhausen	Charmantgasse	32	Deutschland	456456464	5465454	charmant@aon.at	K2343	Charmant Fritz	Edle Brillen	Brillen werden immer 2 Wochen zu spät geliefert
16	Silhouette	4020	Linz	Silhouettestraße	3	Österreich	0049-711-123456	07242 206456	j@gmail.com	K2343	Pürmayr Johann	Sonnenbrillen und normale Brillen	sehr treuer Lieferant
17	Luxottica	P-345	ItalienischerOrt	Luxotticaweg	345	Italien	0047/789456232	07242 456789	ialksdjf@aon.at	K-34	Italiano Fernando	Sonnenbrillen	Billiger Italienischer Lieferant
18	Charmant	23445	Brillhausen	Charmantgasse	32	Deutschland	456456464	5465454	charmant@aon.at	K2343	Charmant Fritz	Edle Brillen	Brillen werden immer 2 Wochen zu spät geliefert
19	Silhouette	4020	Linz	Silhouettestraße	3	Österreich	0049-711-123456	07242 206456	j@gmail.com	K2343	Pürmayr Johann	Sonnenbrillen und normale Brillen	sehr treuer Lieferant
20	Luxottica	P-345	ItalienischerOrt	Luxotticaweg	345	Italien	0047/789456232	07242 456789	ialksdjf@aon.at	K-34	Italiano Fernando	Sonnenbrillen	Billiger Italienischer Lieferant
21	Charmant	23445	Brillhausen	Charmantgasse	32	Deutschland	456456464	5465454	charmant@aon.at	K2343	Charmant Fritz	Edle Brillen	Brillen werden immer 2 Wochen zu spät geliefert
22	Silhouette	4020	Linz	Silhouettestraße	3	Österreich	0049-711-123456	07242 206456	j@gmail.com	K2343	Pürmayr Johann	Sonnenbrillen und normale Brillen	sehr treuer Lieferant
23	Luxottica	P-345	ItalienischerOrt	Luxotticaweg	345	Italien	0047/789456232	07242 456789	ialksdjf@aon.at	K-34	Italiano Fernando	Sonnenbrillen	Billiger Italienischer Lieferant
24	Charmant	23445	Brillhausen	Charmantgasse	32	Deutschland	456456464	5465454	charmant@aon.at	K2343	Charmant Fritz	Edle Brillen	Brillen werden immer 2 Wochen zu spät geliefert
25	Silhouette	4020	Linz	Silhouettestraße	3	Österreich	0049-711-123456	07242 206456	j@gmail.com	K2343	Pürmayr Johann	Sonnenbrillen und normale Brillen	sehr treuer Lieferant
26	Luxottica	P-345	ItalienischerOrt	Luxotticaweg	345	Italien	0047/789456232	07242 456789	ialksdjf@aon.at	K-34	Italiano Fernando	Sonnenbrillen	Billiger Italienischer Lieferant
27	Charmant	23445	Brillhausen	Charmantgasse	32	Deutschland	456456464	5465454	charmant@aon.at	K2343	Charmant Fritz	Edle Brillen	Brillen werden immer 2 Wochen zu spät geliefert
28	Silhouette	4020	Linz	Silhouettestraße	3	Österreich	0049-711-123456	07242 206456	j@gmail.com	K2343	Pürmayr Johann	Sonnenbrillen und normale Brillen	sehr treuer Lieferant
29	Luxottica	P-345	ItalienischerOrt	Luxotticaweg	345	Italien	0047/789456232	07242 456789	ialksdjf@aon.at	K-34	Italiano Fernando	Sonnenbrillen	Billiger Italienischer Lieferant
30	Charmant	23445	Brillhausen	Charmantgasse	32	Deutschland	456456464	5465454	charmant@aon.at	K2343	Charmant Fritz	Edle Brillen	Brillen werden immer 2 Wochen zu spät geliefert
31	Silhouette	4020	Linz	Silhouettestraße	3	Österreich	0049-711-123456	07242 206456	j@gmail.com	K2343	Pürmayr Johann	Sonnenbrillen und normale Brillen	sehr treuer Lieferant

Abbildung 4.8: Screenshot der Lieferantenverwaltung

4.1.4 Verwaltung der lagernden Brillenfassungen

Ebenso wie bei der Verwaltung der Kunden und der Lieferanten gibt es auch eine Verwaltung des lagernden Brillenfassungen. Jede Brille die der Benutzer verkauft hat eine eigene Fassung und die wird hier erfasst. Dabei hat jede Brillenfassung folgende Attribute: Modell, Marke, Farbe, Größe, Status (bestellt, lagernd oder verkauft), Einkaufspreis, Einkaufsdatum, Verkaufspreis, Verkaufsdatum und der Lieferant. Die Liste kann wie gewohnt gefiltert und sortiert werden. Um eine neue Brillenfassung zu erfassen kann der Benutzer auf den Button links oben klicken und um eine bestehende Brillenfassung zu bearbeiten oder zu löschen muss der Benutzer einen Doppelklick auf die gewünschte Brillenfassung tätigen.

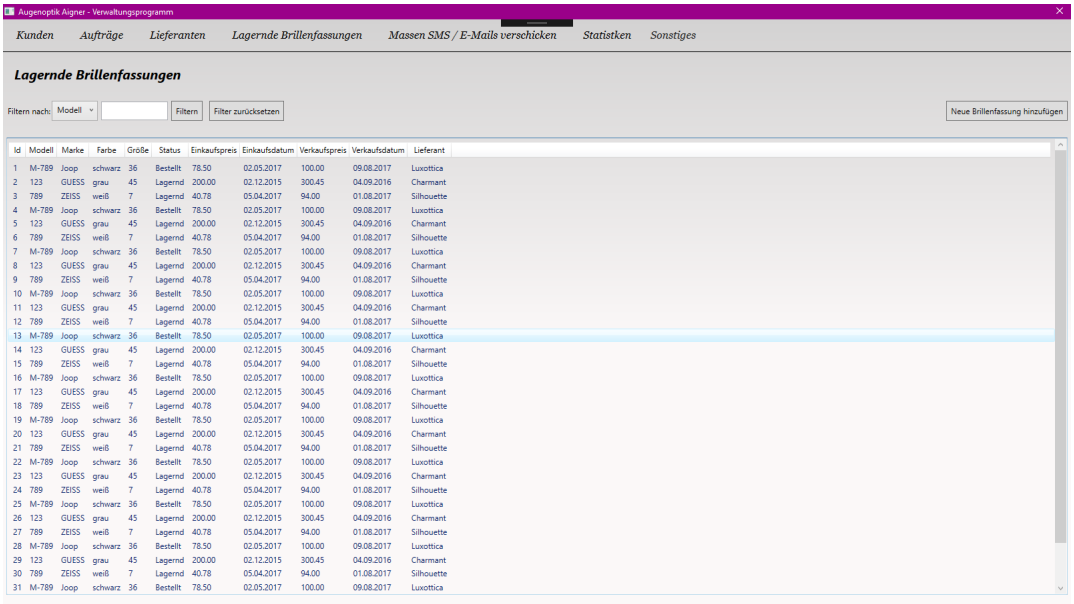


Abbildung 4.9: Screenshot der lagernden Brillenfassungen

4.1.5 E-Mail und SMS

Massennachrichten

Um regelmäßige Info- und Werbenachrichten auszusenden, bietet das Programm die Möglichkeit E-Mails oder SMS an alle Kunden zu versenden. Falls ein Kunde diese Nachrichten nicht mehr erhalten möchte, kann das der Benutzer bei dem einzelnen Kunden eintragen.

E-Mail

Wenn der Benutzer eine Massenemail versenden möchte, kann er einen Betreff und eine Nachricht eingeben, die nachher an alle Kunden gesendet wird. Als E-Mail-Adresse, wird die abgespeicherte Adresse des Kunden verwendet. Falls keine E-Mail-Adresse angegeben wurde, wird eine entsprechende Fehlermeldung angezeigt.

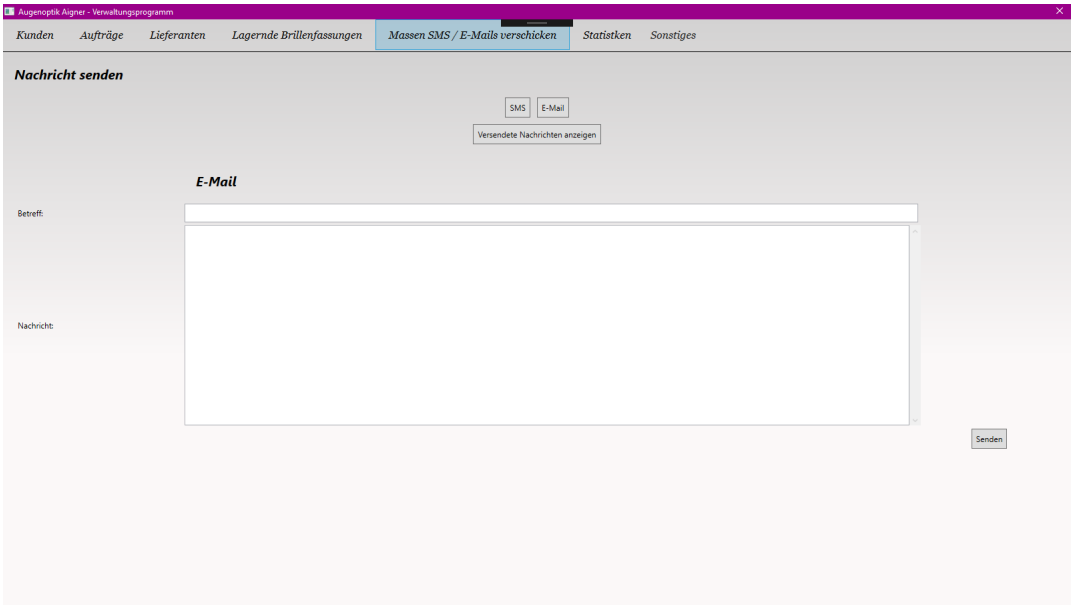


Abbildung 4.10: Screenshot der Massen E-Mails

Technischer Hintergrund:

Es wird für jeden Kunden die gleiche Mail erstellt (Klasse MailMessage vom Namespace System.Net.Mail). Dieser Klasse werden Attribute, wie Sender, Empfänger, Betreff, Nachricht usw. übergeben und mittels eines SMTP-Clients versendet. (Klasse SmtpClient ebenfalls vom Namespace System.Net.Mail). Dem Smtp-Client müssen noch Eigenschaften wie Host, Port und natürlich die E-Mail-Adresse, von der die E-Mail weggeschickt werden soll sowie das Passwort für die E-Mail-Adresse angegeben werden. In diesem Fall wurde eine G-Mail-Adresse verwendet, die extra für diesen Zweck erstellt wurde.

```
var message = new MailMessage();
message.To.Add(new MailAddress(item.Email));
message.From = new MailAddress("infodienst.augenoptikaigner@gmail.com");
message.Subject = this.Subject;
message.Body = this.Message;
this.Recipients.Add(new CustomRecipient() { Customer = item, Address =
    item.Email });

   using (var smtp = new SmtpClient())
    {
      var credential = new NetworkCredential
      {
         UserName = "infodienst.augenoptikaigner@gmail.com",
            Password = "7gnRwN4U"
        };
```

```
        smtp.Credentials = credential;
        smtp.Host = "smtp.gmail.com";
        smtp.Port = 587;
        smtp.EnableSsl = true;

        await smtp.SendMailAsync(message);
    }
```

Danach wird die gesendete Nachricht noch in die Datenbank abgespeichert, damit der Benutzer später alle gesendeten Nachrichten ansehen kann.

SMS

Zum Versenden der SMS wird der SMS-Dienst MessageBird verwendet (Referenz). Ähnlich wie beim Versenden einer E-Mail, gibt der Benutzer wieder eine Nachricht ein, allerdings kann er keine Betreff einfügen.

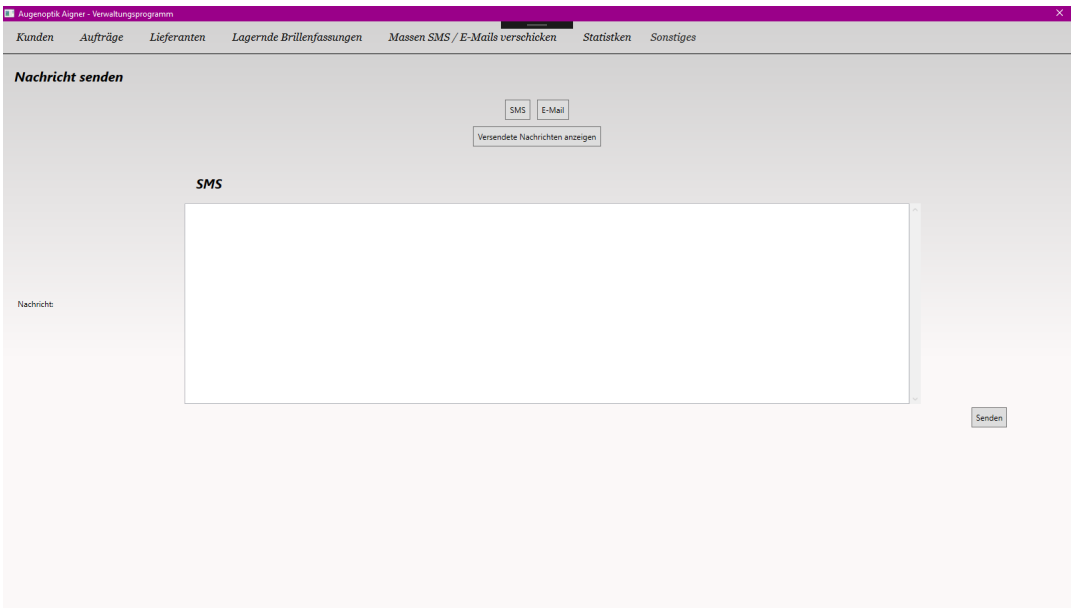


Abbildung 4.11: Screenshot der Massen E-SMS

Diese Nachricht wird dann an alle Kunden gesendet, außer jene, die keine Massennachricht mehr erhalten wollen. Als Telefonnummer wird standardmäßig die Telefonnummer 1 gewählt, außer diese ist nicht vorhanden, dann wird die Telefonnummer 2 gewählt.

Technischer Hintergrund:

Zum Senden einer Nachricht werden folgende Schritte benötigt:

```
IProxyConfigurationInjector proxyConfigurationInjector = null;
Client client = Client.CreateDefault(AccessKey, proxyConfigurationInjector);
```

Und zum Versenden einer Nachricht:

```
MessageBird.Objects.Message message = client.SendMessage("OptikAigner",
    this.Message, numbers);
```

Der AccessKey ist ein normaler String, der von MessageBird erstellt wird. Dabei kann jeder Benutzer von MessageBird mehrere AccessKeys bekommen, beispielsweise einen für Test-Nachrichten, die dann nicht versendet werden oder einen Key, mit dem dann echte SMS versendet werden. Für jede Nachricht die versendet wird, wird das Guthaben auf MessageBird dementsprechend verringert. Sollte das Guthaben auslaufen, wird eine Fehlermeldung angezeigt.

Einzelne Nachrichten

Dieselben Vorgänge werden auch verwendet um einzelne Nachrichten zu versenden. Dazu muss der Benutzer auf die Detailseite einer Bestellung klicken und dann auf „Nachricht senden“. Standardmäßig wird ein Text eingefügt, der dem Kunden mitteilt, dass seine Bestellung nun abholbereit ist. Sollte dies nicht der Grund sein, warum eine Nachricht gesendet werden soll, kann der Benutzer die Nachricht natürlich auch verändern.

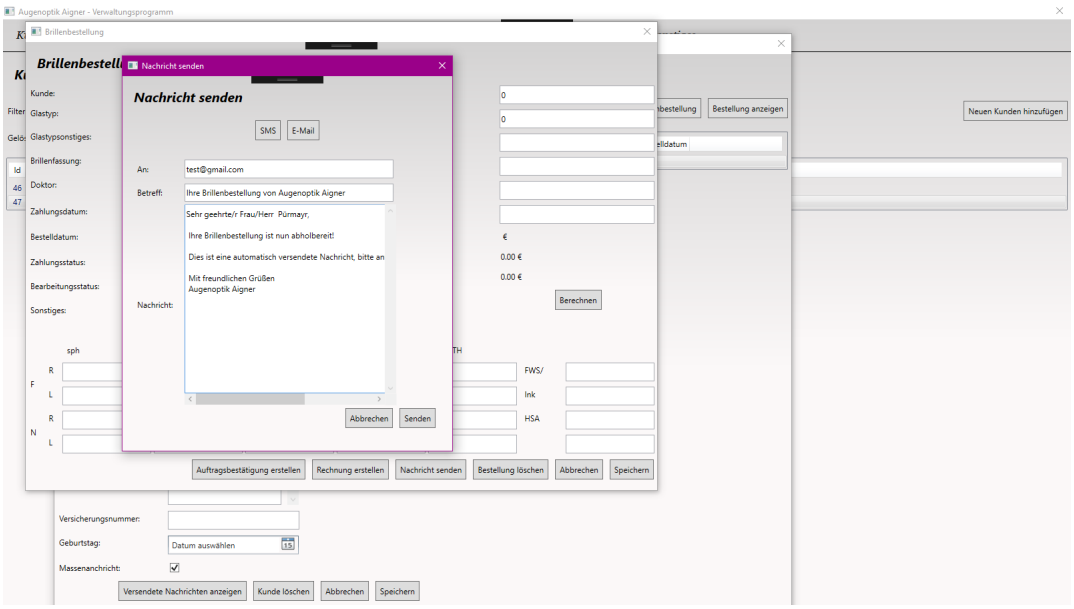


Abbildung 4.12: Screenshot der einzelnen Nachricht

Versendete Nachrichten

Außerdem ist es möglich, alle Nachrichten, die vom System aus gesendet worden sind, anzuzeigen. Um nur Nachrichten anzuzeigen, die an einen bestimmten Kunden gesendet worden sind, muss der Benutzer auf die Detailseite eines Kunden klicken und dann die „Versendeten Nachrichten“ anzeigen. Falls der Benutzer alle Nachrichten sehen will, die er versendet hat, kann er diese unter dem Menüpunkt „Massen SMS /E-Mails verschicken“ sehen.
Dazu wurden in der Datenbank extra die Tabellen CustomMessageünd CustomRecipientängelegt, um alle Nachrichten und deren Empfänger abspeichern zu können. Hier wird beispielsweise eine Massensms gespeichert:

```
var m = new CustomMessage();
m.Date = DateTime.Now;
m.MessageText = this.Message;
m.MessageType = OpticiatnMgr.Core.Entities.MessageType.SMS;
m.Recipients = new List<CustomRecipient>();
var numbers = GetPhoneNumbers();
for (int i = 0; i < this.Customers.Count; i++)
{
    m.Recipients.Add(new CustomRecipient() { Customer =
        this.Customers[i], Address = numbers[i].ToString() });
}
this.Uow.MessageRepository.Insert(m);
this.Uow.Save();
```

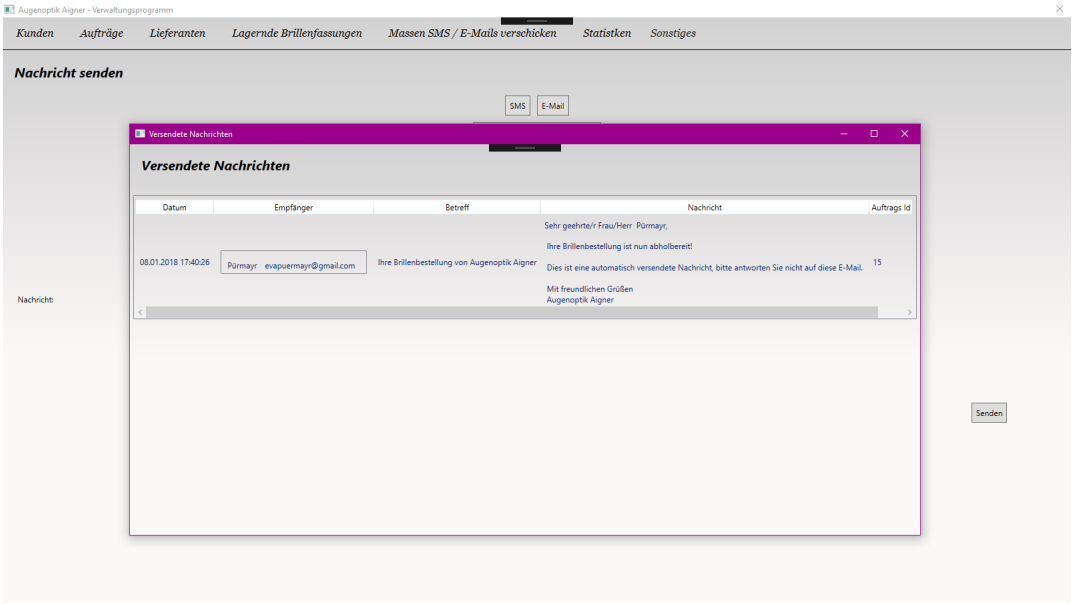


Abbildung 4.13: Screenshot der versendeten Nachrichten

4.1.6 Statistiken

Unter dem Menüpunkt "Statistiken" erhält der Benutzer eine Übersicht über alle verkauften Brillen und Kontaktlinsen. Dazu wird ein Liniendiagramm der verkauften Brillen/Kontaktlinsen von diesem Jahr und dem Jahr davor angezeigt. Damit eine Brille/-Kontaktlinse in der Statistik mitberücksichtigt wird, muss ein Zahlungsdatum angegeben werden und der Bezahlungsstatus muss auf „Bezahlt“ gesetzt werden.

Technischer Hintergrund

Zur Darstellung wurde das package System.Windows.Controls.DataVisualization.Toolkit 4.0.0 verwendet (hier Referenz einfügen). Dazu wird in dem .xaml File der Namespace angegeben:

```
<Page xmlns:toolkitCharting="clr-namespace:System.Windows.Controls
DataVisualization.Charting;assembly=System.Windows.Controls.DataVisualization
.Toolkit">
```

Um ein Liniendiagramm zu erzeugen:

```
<toolkitCharting:Chart Title="{Binding Title}">
          <toolkitCharting:LineSeries Title="{Binding NewYear}"
              DependentValueBinding="{Binding Value}"
              IndependentValueBinding="{Binding Key}" ItemsSource="{Binding
              NewValues}"/>
```

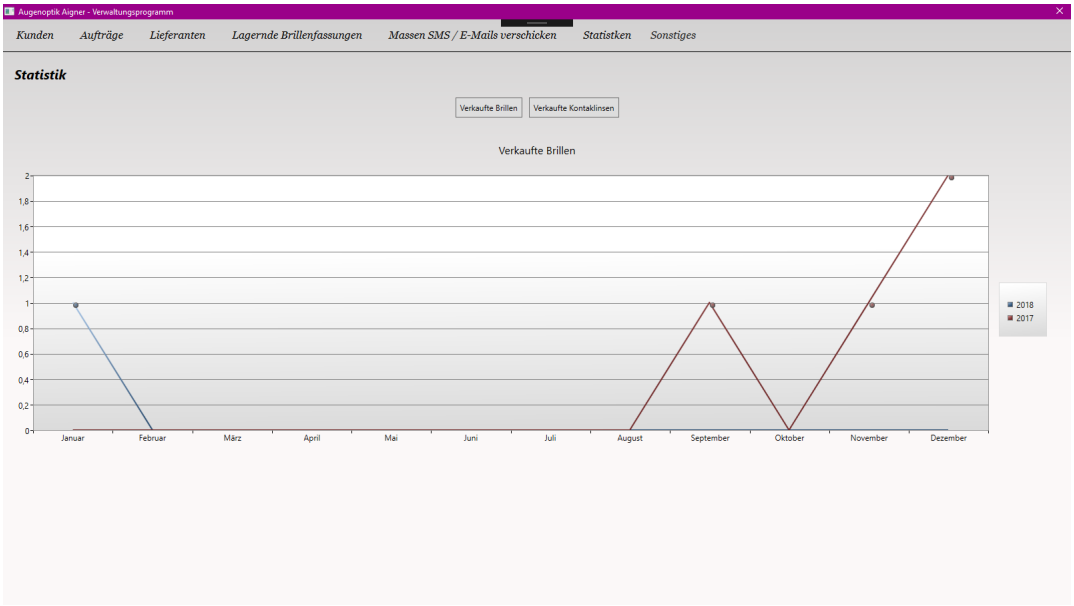


Abbildung 4.14: Screenshot der Statistiken

```
            <toolkitCharting:LineSeries Title="{Binding OldYear}"
                DependentValueBinding="{Binding Value}"
                IndependentValueBinding="{Binding Key}" ItemsSource="{Binding
                OldValues}"/>
</toolkitCharting:Chart>
```

Dabei sind "NewValues" und "OldValues" vom Typ:

```
        public ObservableCollection<KeyValuePair<string, int>> NewValues { get;
            set; }
        public ObservableCollection<KeyValuePair<string, int>> OldValues { get;
            set; }
```

Die Daten werden mittels Linq(Referenz) erfasst.

4.1.7 Filtern und Sortieren

Filtern

Auf allen Hauptseiten der Applikation (Kunden, Brillen- und Kontaktlinsenaufträge, Lieferanten, Lagernde Brillenfassungen) sowie auf den Seiten unter dem Menüpunkt „Sonstiges“ (Orte, Länder, Brillen- und Kontaktlinsentypen) ist es möglich die Datensätze zu filtern. Dies passiert immer nach demselben Schema, dennoch ist diese Funktion für jede dieser Seiten einzeln implementiert. Dazu muss der Benutzer das Feld aussuchen, nach welchem er gerne filtern möchte, danach einen Text eingeben und dann auf „Filtern“ oder

die Taste „Enter“ drücken. Das Programm gibt nun nur jene Datensätze aus, bei denen das gewünschte Feld den eingegebenen Text enthält. Neben dem „Filtern“-Schalter befindet sich ein „Filter löschen“-Schalter, der wieder alle Datensätze zum Vorschein bringt.

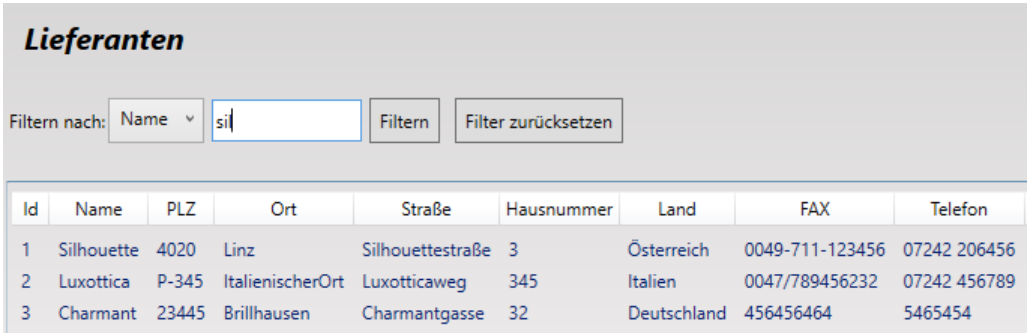


Abbildung 4.15: Screenshot des Filters

Im nachfolgenden Beispiel wird anhand der „Lagernden Brillenfassungen“ erklärt wie der Filter funktioniert. Im ViewModel gibt es ein Feld, welches „PropertiesList“ heißt (vom Typ ObservableCollection¡string¿. In diesem werden alle Felder der jeweiligen Klasse aufgezählt. Davor werden noch Felder, nach denen der Benutzer später nicht filtern sollte, herausgestrichen. Zusätzlich wird jedes Feld in Deutsch übersetzt. Dies geschieht mittels einem ResourceManager, der ein kleines Wörterbuch verwaltet, welches eine Übersetzung für jedes Feld bereithält. Das ist die Liste der Felder aus denen der Benutzer später sein „Filterfeld“ auswählen kann.

```
public ObservableCollection<String> PropertiesList { get; }

private ResourceManager manager = Properties.Resources.ResourceManager;

private ObservableCollection<string> GetAllProperties()
{
   ObservableCollection<string> props = new
       ObservableCollection<string>(typeof(EyeGlassFrame).GetProperties()
.Select(p => p.Name).ToList());
   ObservableCollection<string> newList = new ObservableCollection<string>();
   props.Remove("Timestamp"); //Shouldnt be able to filter by timestamp
    props.Remove("Supplier_Id"); //Shouldnt be able to filter by supplier_id
    foreach (var item in props)
    {
         var germanItem = manager.GetString(item);
         if (germanItem != null)
             newList.Add(germanItem);
    }
    return newList;
```

```
}
```

Wenn der Benutzer einen Text eingibt und danach „Filtern“ drückt, wird das Feld, das er gewählt hat zuerst mit Hilfe des ResourceManagers auf Englisch übersetzt. Danach wird die Mehtode Filter() aufgerufen, die den passenden Filter setzt, falls der Benutzer einen Text eingegeben hat.

```
public void Filter()
        {
            try
            {
                if (!String.IsNullOrEmpty(this.FilterText))
                {
                    this.EyeGlassFramesView.Filter = new
                        Predicate<object>(Contains);
                }
                else
                    this.EyeGlassFramesView.Filter = null;

            }
            catch (Exception e)
            {
                Console.WriteLine(e.StackTrace);
            }
        }
```

Dabei muss man wissen, dass EyeGlassFramesView vom Typ ICollectionView ist. Diese Variable wird im Konstruktor aus der Liste der wirklichen Brillenfassungen erzeugt (EyeGlassFrames). Der Typ ICollectionView ist als Anzeigeelement für Listen gedacht, weshalb es auch ein Feld namens „Filter“ gibt. Durch die Methode „Contains“ wird dieser auch gesetzt.

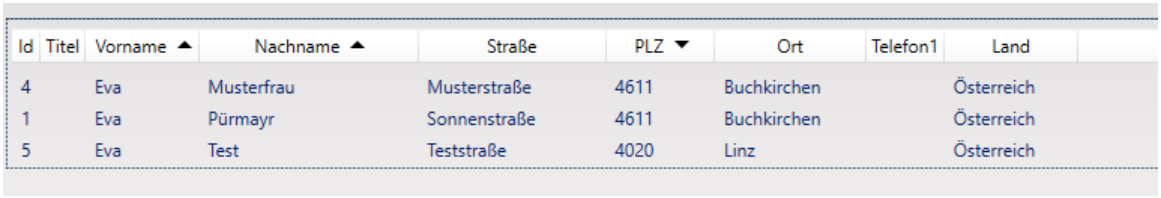
```
public ObservableCollection<EyeGlassFrame> EyeGlassFrames { get; set; }
public ICollectionView EyeGlassFramesView { get; set; }
this.EyeGlassFramesView = CollectionViewSource.GetDefaultView(EyeGlassFrames);
private bool Contains(object f)
        {
            EyeGlassFrame frame = f as EyeGlassFrame;
            if (typeof(EyeGlassFrame).GetProperty(TranslatedFilterProperty) !=
                null)
            {
                return frame.GetType().GetProperty(this.TranslatedFilterProperty)
                .GetValue(frame,
                    null)?.ToString().ToUpper().IndexOf(this.FilterText.ToUpper())
                    >= 0;
            }
            else if (typeof(Supplier).GetProperty(TranslatedFilterProperty) !=
                null) //Does the user filter by suppliername?
```

```
        {
            return frame.Supplier?.GetType()
            .GetProperty(this.TranslatedFilterProperty)
            .GetValue(frame.Supplier,
                null)?.ToString().ToUpper().IndexOf(this.FilterText.ToUpper())
                >= 0;
        }
        else
        {
            MessageBox.Show("Beim Filtern ist ein Fehler aufgetreten!");
            return false;
        }
    }
```

Die Methode Contains gibt zurück, ob das Objekt „f“ dem angegebenen Filter entspricht. Dazu wird zunächst überprüft, ob die Klasse EyeGlassFrame die Property enthält, nach der der Benutzer filtert. Wenn ja, gibt die Methode zurück, ob in dieser Property die Zeichenkette vorkommt, nach der der Benutzer sucht. Danach überprüft das Programm ob die gesuchte Eigenschaft eine Eigenschaft der Klasse Supplier ist. Das passiert, weil jede lagernde Brillenfassung einen Lieferanten hat. Deswegen kann es sein, dass der Benutzer nach einer Eigenschaft filtert, die gar nicht in der Klasse EyeGlassFrame enthalten ist, sondern nur in der Klasse Supplier. Wenn keiner dieser Fälle zutrifft, was nicht vorkommen sollte, wird eine Fehlermeldung zurückgegeben.

Sortieren

Bei den allen Hauptseiten, auf denen Daten angezeigt werden, ist eine dynamische Sortierung implementiert. Diese macht es dem Benutzer möglich, nach drei Spalten gleichzeitig auf- oder absteigend zu sortieren. Dazu muss der Benutzer auf eine beliebige Spaltenüberschrift klicken. Das ist dann die Spalte, nach der zuerst aufsteigend sortiert wird. Drückt der Benutzer erneut auf dieselbe Spaltenübersicht, werden die Datensätze nach dieser Spalte absteigend sortiert. Wenn der Benutzer nach einer zweiten Spalte sortieren möchte, muss er zusätzlich die Shift-Taste drücken, während er die Spaltenüberschrift auswählt. Wiederrum muss der Benutzer ein zweites Mal mit der Shift-Taste die gleiche Spaltenüberschrift anklicken, um absteigend zu sortieren. Dasselbe gilt für die dritte Spalte. Um die Sortierung wieder zurücksetzen zu können, kann der Benutzer eine andere Spaltenüberschrift mit einem normalen Klick wieder sortieren. Im nachfolgenden Bild hat der Benutzer zuerst nach dem Vornamen, dann nach Ort und zum Schluss nach Nachnamen sortiert.



Id	Titel	Vorname	Nachname	Straße	PLZ	Ort	Telefon1	Land
4		Eva	Musterfrau	Musterstraße	4611	Buchkirchen		Österreich
1		Eva	Pürmayr	Sonnenstraße	4611	Buchkirchen		Österreich
5		Eva	Test	Teststraße	4020	Linz		Österreich

Abbildung 4.16: Screenshot des Sortierens

Zur Implementierung gibt es eine Klasse SortManager, die die Sortierung für alle Hauptseiten regelt. Dazu wird im Konstruktor ein neuer SortManager initialisiert.

```
SortManager = new SortManager();
public SortManager SortManager { get; set; }
```

Zusätzlich werden drei Events von der View abonniert.

```
<i:Interaction.Triggers>
                <i:EventTrigger EventName="Loaded">
                    <cmd:EventToCommand Command="{Binding Initialized}"
                                        PassEventArgsToCommand="True" />
                </i:EventTrigger>
            </i:Interaction.Triggers>
```

Dieses Event wird von der ListView bereitgestellt. In jedem ListViewHeader werden noch Shift+LeftClick und MouseDown abonniert.

```
<GridViewColumn DisplayMemberBinding="{Binding FirstName,
    UpdateSourceTrigger=PropertyChanged}">
                        <GridViewColumnHeader Content="Vorname">
                            <GridViewColumnHeader.InputBindings>
                                <MouseBinding Gesture="Shift+LeftClick"
                                    Command="{Binding SortShift}"
                                    CommandParameter="FirstName" ></MouseBinding>
                            </GridViewColumnHeader.InputBindings>
                            <i:Interaction.Triggers>
                                <i:EventTrigger EventName="MouseDown">
                                    <cmd:EventToCommand Command="{Binding
                                        SortCommand}"
                                        PassEventArgsToCommand="True" />
                                </i:EventTrigger>
                            </i:Interaction.Triggers>
                        </GridViewColumnHeader>
                    </GridViewColumn>
```

Im ViewModel werden dann folgende Methoden aufgerufen:

```
private void Init(RoutedEventArgs p)
```

```
{
    SortManager.Init(p);
}
//Click without shift key
private void SortS(RoutedEventArgs e)
{
    var tmp = this.CustomersView;
    SortManager.SortNormal(e, ref tmp);
}

//Click with shift
private void SortSh(object p)
{
    var tmp = CustomersView;
    SortManager.SortShift(p, ref tmp);
}
```

In der Methode SortManager.Init(RoutedEventArgs p) werden durch die Variable p alle GridViewColumnHeader abgespeichert. Der Grund dafür ist, dass bei dem Event SortShift keine EventArgs mitgegeben werden können, weil es sich um ein MouseBinding handelt und nicht um ein normales Event. Dadurch kann die Methode SortShift(object p, ref ICollectionView View) nicht wissen, welche Spaltenüberschrift gedrückt wurde und daher müssen am Anfang einmal alle GridViewColumnHeaders abgespeichert werden. Wenn die Methode SortNormal(RoutedEventArgs e, ref ICollectionView View) aufgerufen wird, wird zunächst überprüft ob die Spaltenüberschrift schon einmal gedrückt wurde (dann soll nämlich die Sortierrichtung geändert werden). Wenn ja, werden die Suchrichtung sowie die Richtung des Pfeils neben der Spaltenüberschrift geändert. Ansonsten werden alle Pfeile neben den Überschriften gelöscht, die Suchrichtung auf aufsteigend gesetzt und ein neuer Pfeil gesetzt. Ein Auszug der SortNormal-Methode:

```
//Same column pressed?
          if (SortHeaders.Count > 0 && SortHeaders[0] == columnHeader)
          {
              //Change sort direction
              dir = View.SortDescriptions[0].Direction;
              dir = dir == ListSortDirection.Ascending ?
                  ListSortDirection.Descending : ListSortDirection.Ascending;
              header = ChangeArrow(columnHeader, dir);
          }
          else
          {
              //Remove arrow from old column header
              if (SortHeaders.Count > 0)
              {
                  foreach (var item in SortHeaders)
                  {
                      item.Column.HeaderTemplate = null;
```

```
                item.Column.Width = item.ActualWidth - 20;
            }
        }
        SortHeaders.Clear();
        SortHeaders.Add(columnHeader);
        //default sort direction is ascending
        dir = ListSortDirection.Ascending;
        header = SetNewArrow(columnHeader, dir);
    }
    View.SortDescriptions.Clear();

    View.SortDescriptions.Add(new SortDescription(header, dir));
```

SortHeaders ist eine globale Variable vom Typ List¡GridViewColumnHeader¿, der die Spalten enthält, nach den aktuell sortiert wird. Die lokale Variable dir bezeichnet die gewünschte Sortierrichtung und ist vom Typ ListSortDirection. Der String header enthält die Property, auf die der GridViewColumnHeader bindet. Diese ist natürlich Englisch und stellt wieder ein Übersetzungsroblem dar. Wie schon weiter oben erwähnt, ist der Typ ICollectionView extra für das Darstellen von Listen gemacht, deshalb enthält er auch eine Eigenschaft SortDescriptions, in welche man beliebig viele SortDescriptions einfügen kann und nach welchen die Liste automatisch sortiert wird. In den Methoden ChangeArrow und SetNewArrow wird die Spalte entsprechend breiter gemacht und das passende vorgefertigte Template gesetzt.

```
column.Column.HeaderTemplate = Application.Current.FindResource("ArrowUp") as
    DataTemplate;
```

In diesem Beispiel wird dem GridViewColumnHeader column ein Pfeil der nach oben ausgerichtet ist beigefügt. In der Methode SortShift(object p, ref ICollectionView View) wird mittels dem Parameter p der Name der Property übergeben, an die sich die Spalte bindet. Dieser wird händisch in der View übergeben (siehe oben) und ist englisch, weshalb er zuerst übersetzt werden muss. Danach wird der passende GridViewColumnHeader nach der Überschrift in den am Anfang angelegten GridViewColumnHeaders gesucht.

```
var columnHeader = AllHeaders.Where(h => String.Equals(h.Content.ToString(),
    germanColumnName)).ToList().FirstOrDefault();
```

Danach wird wieder überprüft, ob dieselbe Spalte zweimal hintereinander ausgewählt wurde, sodass dann die Sortierrichtung gewechselt werden kann. Ansonsten wird überprüft ob schon drei Spalten ausgewählt wurden und wenn nicht wird eine neue Spalte zu den Sortierspalten hinzugefügt. Auszug der SortShift-Methode:

```
if (View.SortDescriptions.Count >= 1)
        {
            ListSortDirection dir;
            int index = View.SortDescriptions.Count - 1;
            //Change sorting direction
```

```
if (View.SortDescriptions.Count == index + 1 &&
    View.SortDescriptions[index].PropertyName == columnName)
{
    dir = View.SortDescriptions[index].Direction;
    dir = dir == ListSortDirection.Ascending ?
        ListSortDirection.Descending :
        ListSortDirection.Ascending;
    View.SortDescriptions.RemoveAt(index);
    View.SortDescriptions.Insert(index, new
        SortDescription(columnName, dir));
    ChangeArrow(columnHeader, dir);
    SortHeaders.Add(columnHeader);
}
else if (View.SortDescriptions.Count(s => s.PropertyName ==
    columnName) == 0)
{
    if (View.SortDescriptions.Count >= 3)
    {
        MessageBox.Show("Sie k\"onnen maximal nach drei Spalten
            sortieren!", "Hinweis", MessageBoxButton.OK,
            MessageBoxImage.Exclamation);
        return;
    }
    dir = ListSortDirection.Ascending;
    SetNewArrow(columnHeader, dir);
    View.SortDescriptions.Add(new SortDescription(columnName,
        dir));
    SortHeaders.Add(columnHeader);
}
```

Kapitel 5

Selbstevaluation

The details of the structure of your thesis have to be aligned with the supervising teacher. However, most of the theses require to have some description of the models used or some other theoretical background necessary to understand the rest of the text.

Kapitel 6

Summary

Here you give a summary of your results and experiences. You can add also some design alternatives you considered, but kicked out later. Furthermore you might have some ideas how to drive the work you accomplished in further directions.

Literaturverzeichnis

[Wik17] Wikipedia. .net, March 2017. Page Version ID: 163867861. URL: https://de.wikipedia.org/w/index.php?title=.net&oldid=163867861.

Abbildungsverzeichnis

Tabellenverzeichnis

Project Log Book

Date	Participants	Todos	Due

Anhang A

Additional Information

If needed the appendix is the place where additional information concerning your thesis goes. Examples could be:

- Source Code
- Test Protocols
- Project Proposal
- Project Plan
- Individual Goals
- ...

Again this has to be aligned with the supervisor.

Anhang B

Individual Goals

This is just another example to show what content could go into the appendix.