Projekt: Pendenzen Allpower

Christian Seiler
christian@christianseiler.ch

Oktober - November 2016

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

QUELLCODES

QUELLCODES

KAPITEL 1

EINFÜHRUNG

1.1 ALLGEMEIN

Eine To-do-Liste (engl. zu tun) auch Pendenzenliste oder Aufgabenliste genannt wird beim Aufgabenmanagement oder Aufgabenplanung einzelner Personen, Gruppen oder Projekten verwendet. Es wird festgehalten, welche Aufgaben anstehen, wer dafür verantwortlich ist und bis wann sie erledigt werden müssen.

1.2 AUSGANGSLAGE

Bislang wurde bei Allpower resp. deren Abteilung Einkauf eine Pendenzenliste in Form einer Exceldatei geführt und deren Programmierung bestand hauptsächlich aus verschiedenen Formeln und bedingten Formatierungen.

Abbildung 1.2.1 Microsoft Excel

Der Einsatz dieser Pendenzenliste brachte bei einem 15-köpfigen Team verschiedene Nachteile in sich:

- nicht Multiuser fähig
  Da die Pendenzenliste in einer einzige Datei geführt wurde, war es nicht möglich, mit mehreren Benutzern gleichzeitig auf diese Datei zuzugreifen.

- nicht erweiterbar
  Obwohl in der Datei eine nahezu beliebige Anzahl Pendenzen eingefügt werden kann, sind die Formeln und bedingten Formatierungen auf eine bestimmte Anzahl Zeilen begrenzt und muss manuell erweitert werden, wenn diese Anzahl erreicht ist.

- nicht erweiterbar
  Einzelne Formeln der Tabelle sind recht komplex und nicht leicht zu ändern. Will man nun eine neue Funktion (z. B. eine neue Zeile, einen neuen Status etc.) einbauen, muss sehr viel Aufwand in Kauf genommen werden um dies zu realisieren.

- keine Nachverfolgung von Änderungen
  Jede Änderung einer Zelle überschreibt den vorherigen Wert. Es ist so nicht möglich, nachzuverfolgen, was der Ursprüngliche Wert war, geschweige, wer diesen geändert hat. Dies macht es zu einer enormen Fehlerquelle ohne Möglichkeit zur Nachverfolgung.

1.3 LÖSUNGSFINDUNG

Um die Nachteile zu beheben, vor allem aber, damit verschiedene Benutzer gleichzeitig deren Pendenzen bearbeiten können, wurden verschiedene mögliche Lösungen angedacht. Jedes dieser möglichen Lösungen bringt seine eigenen Vor- und Nachteile mit.

Die in Betracht gezogenen Lösungsmöglichkeiten sind:

- Web Applikation
- Access Datenbank
- Eigenproduktion

1.3.1 Web Applikation

Abbildung 1.3.1 PHP

Der Einsatz eines Webformulars ermöglicht es, eine plattformunabhängige Lösung zu erstellen. Dies bedeutet, dass unabhängig von Betriebssystemen und Geräteart auf die Pendenzen zugegriffen werden kann.

Das Web Formular würde auf Basis einer MySQL einer relationale Datenbank und einer Webseite geschrieben in PHP und JavaScript aufgebaut.

Vorteile

- Plattformunabhängig
- keine Installation notwendig
- Erweiterbar

Nachteile

- Programmierkenntnisse in PHP und SQL erforderlich

1.3.2 Access Datenbank

Abbildung 1.3.2 Microsoft Access

Microsoft Access ist eine Anwendung innerhalb der Microsoft Office Suite. MS Access verbindet relationale Datenbank mit den Werkzeugen einer Integrierte Entwicklungsumgebung (IDE) und verfügt über viele verschiedene Vorlagen, so auch Datenbanken zum Verwalten von Problemen, Aufgaben oder Projekten.

Vorteile

- keine zusätzliche Installation notwendig
- Vorlagen vorhanden

Nachteile

- Roh-Daten sind für den User zugänglich gespeichert
- Keine Änderungsverfolgung
- Anpassen der Vorlagen zeitraubend und aufwändig

1.3.3 EIGENPRODUKTION

Abbildung 1.3.3 Visual Studio

Auf Basis des UserSetup Projekts aufbauende, in C# geschriebene Applikation welche auf den Workstations installiert wird. Auf dem Server kann die bereits vorhandene MySQL Datenbank genutzt werden um die Daten zu speichern.

Vorteile

- Kann beliebig erweitert werden
- Corporate Identity (CI)
- Kann vollständig an die betrieblichen Anforderungen angepasst werden

Nachteile

- Programmierkenntnisse in C# und SQL erforderlich

1.4 ENTSCHEIDUNG

Durch das Ausprobieren der von Microsoft erstellten Access-Vorlage und ersten Versuchen diese an die gestellten Anforderungen anzupassen, stellte sich heraus, dass das Anpassen zu viel Zeit beanspruchen würde und das Ergebnis mit grosser Wahrscheinlichkeit nicht in vollem Masse befriedigend sein würde.

In einem Testversuch eine Webapplikation mit PHP zu erstellen, hat sich gezeigt, dass die Kenntnisse nicht ausreichen, um die Anforderung in einem ansprechenden Design und in einem angemessenen Zeitrahmen zu erstellen.

Da bereits eine eigene, in C# produzierte Applikation besteht, lag es Nahe, die Pendenzen App ebenfalls auf diese Weise zu erstellen. Dies im Punkt 1.3.3 genannten Nachteile fallen daher nur noch geringfügig ins Gewicht.

1.5 SOURCE CODE

Der in dieser Dokumentation abgedruckte Source Code beschreibt die wichtigsten Funktionen der Applikation. Es ist dabei zu beachten, dass vereinzelte Bezeichnungen und Funktionen zu Gunsten der Lesbarkeit verkürzt und nicht vollständig dargestellt ist.

KAPITEL 2

PRODUKTION

Die Eigenproduktion basiert auf dem vorausgegangenen Projekt UserSetup[1], wobei auch in dieser Produktion die Programmiersprache C# verwendet wird. Zusätzlich kommt die Datenbank MySQL als Speicher für die Daten zum Einsatz.

Abbildung 2.0.1 C#

2.1 DATENBANK

Die Daten der Pendenzen-App werden in einer zentralen Datenbank auf dem Server gespeichert. Da die Software MySQL bereits aus dem Projekt Intranet vorhanden ist, konnte die Datenbank für diese Applikation auch verwendet werden. Dazu wurde eine neue Datenbank angelegt in der die Informationen zu den Pendenzen gespeichert werden.

Abbildung 2.0.2 MySQL

2.1.1 DATENTYPEN

Um diesen Abschnitt zu verstehen, werden zunächst die verschiedenen Datentypen einer relationale Datenbank aufgezeigt. Ein Datentyp beschreibt eine Zusammenfassung von Objektmengen bzw. Wertebereichen, den Variablen und

[1] Siehe www.christianseiler.ch/downloads

Konstanten annehmen können. Die Bezeichnungen der Datentypen leiten sich aus den englischen Bezeichnungen ab (int von Integer für Ganzzahl).

Datentyp	**MySQL**	**Beschreibung**
Integer	int(n)	Ganzzahl der Länge n
Character String	varchar(n)	Zeichenkette maximale Länge n
Date	date	Datum im Format yyyy-MM-dd
Time	time	Zeit im Format HH:mm:ss
Date & Time	datetime	Datum und Zeit im Format yyyy-MM-dd HH:mm:ss
Longtext	longtext	Zeichenkette maximale Länge 2^{32}

Tabelle 2.1.1 Datentypen

Eine SQL Datenbank bringt, neben den Datentypen, welche im gesamten Gebiet der Programmierung auftreten, einige eher eigene Begriffene mit. Diese zu kennen erlaubt es die anschliessenden Tabellenstrukturen zu verstehen.

MySQL	**Beschreibung**
NULL	Die Spalte kann leer sein, nicht zu verwechseln mit “0”
NOT NULL	Die Spalte darf nicht leer sein (nachfolgend mit “NN” abgekürzt). Wenn dies nicht zusammen mit DEFAULT x verwendet wird, muss sichergestellt werden, dass diese Spalte gefüllt wird, ansonsten werden die Daten nicht gespeichert.
DEFAULT x	Der Standartwert, wenn nichts anderes festgelegt wird, meist im Zusammenhang mit entweder NULL, NOT NULL oder einem festgelegten Wert wie “open”, (nachfolgend mit ‘DFT’ abgekürzt)
AUTO_INCREMENT	Der Wert dieser Spalte wird automatisch um den angegebenen Wert erhöht (ohne Angabe = 1)

Tabelle 2.1.2 Begriffe

2.1.2 AUFBAU

Die Datenbank besteht aus 4 einzelnen Tabellen, welche für unterschiedliche Zwecke genutzt werden. Der Aufbau sieht wie nachfolgend beschrieben aus.

Tabelle	Nutzung
admin	Speichert alle globalen Einstellungen
company	Speichert alle Firmenadressen
country	Speichert die Länder mit deren Länderkürzel
pendenz	Speichert die verschiedenen Pendenzen

Tabelle 2.1.3 Datenbankorganisation

Die Tabelle Admin (Tabelle 2.1.4) wird für sämtliche Einstellungen genutzt um das Programm zu betreiben. Derzeit sind nur wenige Einstellungen in dieser Tabelle gespeichert, ist aber für weitere Entwicklungen vorbereitet.

Spalte	Spezifikation	Beispiel
id	varchar(45) NN	password
option	varchar(45) DFT NULL	12345

Tabelle 2.1.4 Admin

Die Tabelle Country (Tabelle2.1.5) speichert die verwendeten Länder mit den dazugehörigen Kurzzeichen.

Spalte	Spezifikation	Beispiel
id	varchar(6) NN	I
country	varchar(45) DFT NULL	Italien
ISO2	varchar(2) DFT NULL	IT

Tabelle 2.1.5 Country

Die Tabelle Company (Tabelle 2.1.6) stellt alle Felder zur Verfügung, um die verschiedenen Informationen einer Firma zu speichern.

Spalte	Spezifikation	Beispiel
id	varchar(6) NN	APW (Firmenkürzel)
name	varchar(45) NN	Allpower
street	varchar(45) DFT NULL	Sägestrasse 26
pobox	varchar(45) DFT NULL	Postfach 1
plz	varchar(10) DFT NULL	5600
city	varchar(45) DFT NULL	Lenzburg
country	varchar(45) DFT NULL	Schweiz
phone	varchar(45) DFT NULL	056 200 93 39
website	varchar(45) DFT NULL	www.allpower.ch
mail	varchar(45) DFT NULL	info@allpower.ch
history	longtext	Der Verlauf aller Änderungen

Tabelle 2.1.6 Company

Spalte	Spezifikation	Beispiel
id	int(11) NN AUTO_INCREMENT	einzigartige ID
lieferant	varchar(10) NN	Lieferant
referenz	longtext	Referenznummer entspricht Bestellnummer im BusPro
document	longtext	Dokumentennummer der Lieferfirma (Auftragsbestätigung / Lieferschein / Gutschrift / Rechnung)
erfasstam	datetime NN	Erfassungszeit
erfasstvon	varchar(5) NN	Erfasser
bearbeiter	varchar(5) DFT NULL	aktueller Sachbearbeiter
due	date DFT NULL	Fälligkeitsdatum
detail	longtext	History
finalized	date DFT NULL	Abschlussdatum
state	varchar(45) DFT 'open'	Aktueller Status der Pendenz
departmen	varchar(45) DFT NULL	Abteilung, welche die Pendenz betrifft

Tabelle 2.1.7 Pendenz

KAPITEL 3

PENDENZENMANAGEMENT

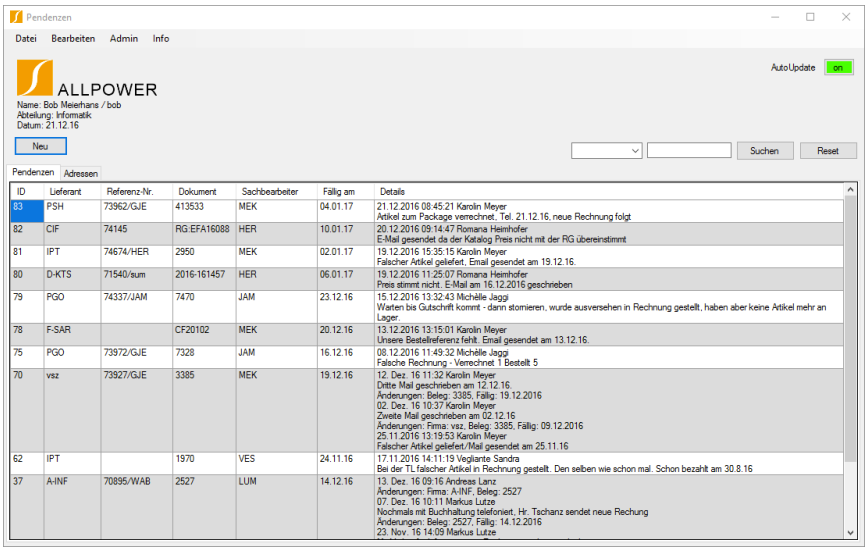

Abbildung 3.0.1 User Interface Pendenzen

Der Einstieg in das Pendenzenmanagementtool zeigt sich, in dem es direkt nach dem Start eine Auflistung aller offenen Pendenzen anzeigt. Diese Pendenzen werden automatisch aktualisiert.

3.1 PENDENZENÜBERSICHT

Das Update der Pendenzenübersicht wird über Multithreading im Hintergrund erarbeitet. Dazu wird beim Start der App die Funktion threadStarter() aufgerufen, welche dann den eigentlichen Thread threadTask() startet (siehe Code 3.1.1). Der

Wert **isOn** bezeichnet einen Bool-Wert (Wahr oder Falsch). Dieser Wert wird von verschiedenen Funktionen der App manipuliert und so das AutoUpdate temporär angehalten oder gestartet.

```
1  private void threadStarter() {
2      var thread = new Thread(threadTask);
3      thread.IsBackground = true;
4      thread.Start();
5  }
6  private void threadTask() {
7    while (true) {
8      Thread.Sleep(500);
9      if (isOn) { Invoke((MethodInvoker) delegate { reloadData(
           tabControl.SelectedIndex); }); }
10     if (!isOn) { }
11   }
12 }
```

Quellcode 3.1.1 Multithreading

Die Funktion reloadData(**int** tabIndex) (Code 3.1.2), die durch threadTask() aufgerufen wird, prüft zunächst, welcher Tab aktiviert ist. Neben dem Tab Pendenzen gibt es noch die Adressen (siehe Kapitel 4). Abhängig vom aktivierten Tab, ruft die Funktion die entsprechenden Funktionen auf. Im Falle der Pendenzen ist dies loadIssues() (Code 3.1.3) und formatDataView(), hingegen wenn der Adressen-Tab aktiv ist, wird die Funktion loadContact(**int** id) (Code 4.0.1) ausgeführt.

```
13 private void reloadData(int tabIndex) {
14     if (tabIndex == 0) {
15         loadIssues();
16         formatDataView();
17     } else if (tabIndex == 1) {
18         loadContact(idLabel.Text);
19     }
20 }
```

Quellcode 3.1.2 Reload Data

Die Funktion loadIssues() (Code 3.1.3) sendet eine SQL Query an den Server, prüft ob das Ergebnis von den bereits vorhandenen Daten abweicht und leitet die Daten die Datentabelle weiter. Die Scroll-Position wird dabei vor jeder Anfrage gespeichert und anschliessend wiederhergestellt.

Die Funktion formatDataView() formatiert die Ansicht der Tabelle. Sie ersetzt die Kopfzeilen und setzt die Spaltenbreite abhängig vom Inhalt.

```
21 private void loadIssues() {
22     var r = issueDataView.FirstDisplayedScrollingRowIndex;
23     DataTable newTable = db.Select(query);
24     if (!sameTable(oldTable, newTable)) {
25         issueDataView.DataSource = newTable;
26         oldTable = newTable;
27     }
28     if (r >= 0)
29     issueDataView.FirstDisplayedScrollingRowIndex = r;
30 }
```

Quellcode 3.1.3 Load Issues

3.2 PENDENZEN ANLEGEN

Beim Erstellen einer neuen Pendenz werden die erforderlichen Felder in einem Formular abgefragt. Daten wie Name und Abteilung des Erstellers werden direkt aus dem Active Directory (AD) abgefragt und eingesetzt. Das Fälligkeitsdatum wird standardmässig auf das aktuelle Datum plus sieben Tage festgelegt (Code: 3.2.1). Dies kann jedoch vom jedem Benutzer individuell angepasst werden.

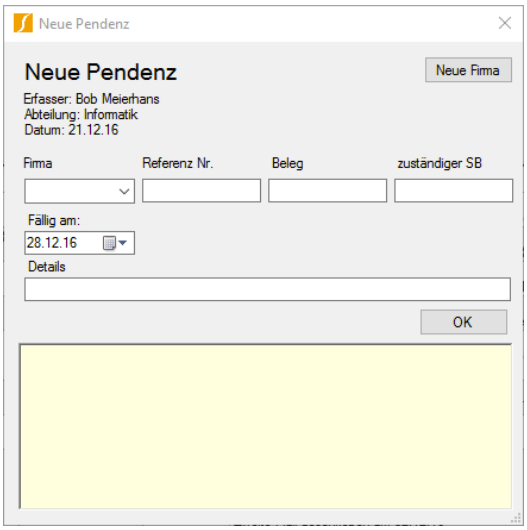

Abbildung 3.2.1 Neue Pendenz anlegen

```
31 public addIssue() {
32   setCompany(); // Füllt das Dropdown-Feld "Firma"
33   creator.Text = $"Erfasser: {person.getInfo()[1]} {person.
         getInfo()[2]}";
34   abteilung.Text = $"Abteilung: {person.getInfo()[4]}";
35   due.Value = DateTime.Now.AddDays(7);
36 }
```

Quellcode 3.2.1 Pendenz anlegen

Falls eine Firma noch nicht in der Datenbank erfasst ist und somit nicht im Dropdown “Firma” erscheint, kann diese direkt aus diesem Fenster angelegt werden (Siehe dazu Kapitel 4).

In der gelben Box werden zu dem jeweilig aktiven Textfeld Hilfetexte eingeblendet. Dies kann zum Beispiel der folgende Text für das Feld Beleg sein: “Die Belegnummer entspricht der Dokumentennummer der Lieferfirma. Auftragsbestätigung (AB), Lieferschein (LS), Gutschrift (GU) oder Rechnung (RG).”

Beim Senden der neuen Pendenz (Code: 3.2.3) wird zunächst die Funktion checkEntries() ausgeführt (Code: 3.2.2). Diese Funktion prüft die Eingaben und gibt eine Rückmeldung, falls ein oder mehrere Feld nicht ausgefüllt wurde. Zudem wird der Bool-Wert **_error** gesetzt.

```
37 private void checkEntries() {
38   if (company.Text == "") {
39     _error = true;
40     error.Text += "Firmenangabe";
41   }
42   if ((reference.Text == "") && (document.Text == "")) {
43     if (_error) error.Text += ", ";
44     _error = true;
45     error.Text += "Referenz oder Beleg";
46   }
47   if (_error) error.Text += " ist zwingend einzugeben.";
48 }
```

Quellcode 3.2.2 Errorcheck

Nach dem die Textfelder geprüft wurden, werden deren Inhalte als lokale Variablen zwischengespeichert um daraus die SQL Query zu formen. Anschliessend wird die Anfrage an die Datenbank abgeschickt.

```
49 private void submit(object sender, EventArgs e) {
50   checkEntries();
51   string details = $"{DateTime.Now} {person.getInfo()[1]} {
         person.getInfo()[2]}\n{details.Text}";
52   var lieferant = company.Text;
53   if (_error == false) {
54   string query = $"INSERT INTO pendenz (lieferant, referenz,
         document, erfasst_am, erfasst_von, due, sachbearbeiter,
         detail, department) VALUES('{lieferant}', '{referenz}',
         '{document}', '{erfasstAm}', '{erfasstVon}', '{due}', '{
         bearbeiter}', '{details}', '{department}')";
55   db.Insert(query);
56   Close();
57   }
58 }
```

Quellcode 3.2.3 Pendenz senden

3.3 PENDENZ BEARBEITEN

Zum Bearbeiten einer Penzenz, kann diese durch anklicken in der Gesamtansicht geöffnet werden. Es öffnet sich nun das Bearbeitungsfenster im welchem Änderungen und Updates eingetragen werden. Im Feld “Unternommene Schritte” wird dabei der Verlauf aller Änderungen aufgelistet.

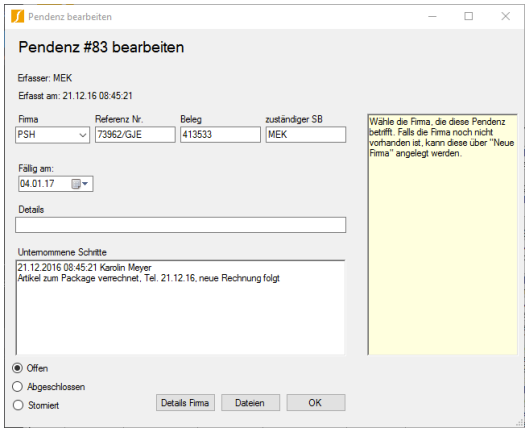

Abbildung 3.3.1 Pendenz bearbeiten

```
59 public modifyIssue(int id) {
60   string query = $"SELECT * FROM pendenz WHERE id = {id}";
61   var dataTable = db.Select(query);
62   var i = 0;
63   foreach (DataRow row in dataTable.Rows)
64   foreach (DataColumn column in dataTable.Columns) {
65     list.Add(row.ItemArray[i].ToString()); i++;
66   }
67   // Daten aus Query an die Textfelder verteilen
68   changeIssue.Text = $"Pendenz #{id} bearbeiten";
69   company.Text = list[1];
70   switch (list[10]) {
71     case "cancelled": cancelButton.Checked = true; break;
72     case "done": finalizedButton.Checked = true; break;
73     default: openButton.Checked = true; break;
74   }
75   abteilung = list[11];
76   p = Path.getPath(company.Length == 3, company, abteilung);
77   if (p == "-1") openFilesButton.Visible = false;
78 }
```

Quellcode 3.3.1 Pendenz bearbeiten

KAPITEL 4

ADRESSENMANAGEMENT

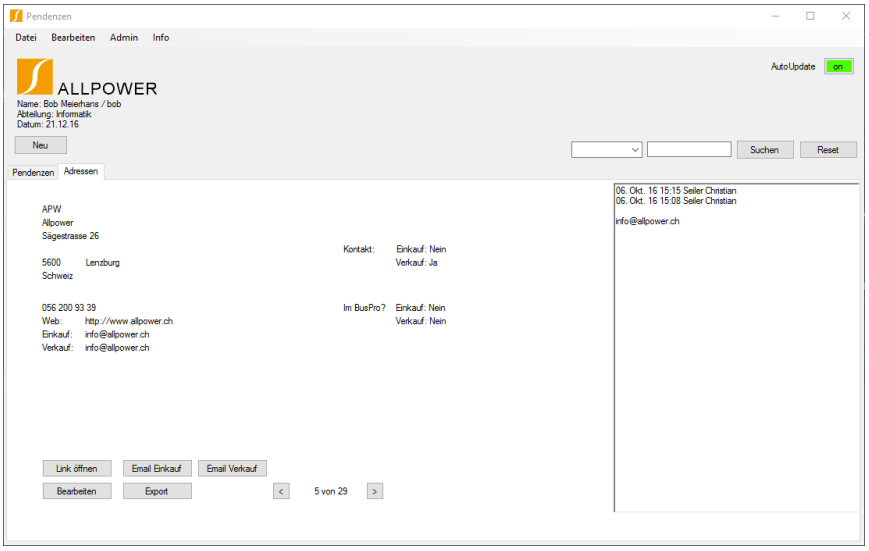

Abbildung 4.0.1 User Interface Adressen

Wie bereits im Kapitel 3 angedeutet, beinhaltet die Applikation neben dem Pendenzenmanagement auch eine Adressdatenbank. Beim Aktivieren des Adressentabs wird die Funktion loadContact(**string** id) ausgeführt.

```
79 private void loadContact(string id) {
80   t = db.Select($"SELECT * FROM company WHERE id = '{id}'");
81   makeData(t);
82 }
```

Quellcode 4.0.1 Reload Data

Die Funktion loadContact(**string** id) generiert eine Datentabelle, welche anschliessend an die Funktion nameData(**DataTable** t) geleitet wird. Diese beiden Vorgänge sind in zwei separate Funktionen aufgeteilt, damit diese für die Suche nach einer Firma nutzen zu können. Dazu besteht eine die erweiterte Funktion loadContact(**string** k, **string** id) welche mit einem zweiten Parameter k die Query zusätzlich ändert (z.B. {" ... WHERE city = '{id}'"} statt {" ... WHERE id = '{id}'"}).

```
83  private void loadContact(string k, string id) {
84    t = db.Select($"SELECT * FROM company WHERE {k} = '{id}'");
85    makeData(t);
86  }
```

Quellcode 4.0.2 Reload Data mit Suche

Nachdem die Datentabelle für den Kontakt erstellt wurde, wird diese in der Funktion makeData(**DataTable** t) an die verschiedenen Textfelder verteilt.

```
87  private void makeData(DataTable t) {
88    var c = new ArrayList(t.Columns.Count);
89    foreach (DataRow row in t.Rows)
90    foreach (DataColumn column in t.Columns)
91    contact.Add(row[column]);
92    id.Text = c[0].ToString();
93    company.Text = c[1].ToString();
94    street.Text = c[2].ToString();
95    poBox.Text = c[3].ToString();
96    plz.Text = c[4].ToString();
97    city.Text = c[5].ToString();
98    country.Text = c[6].ToString();
99    phone.Text = c[7].ToString();
100   url.Text = c[8].ToString();
101   email.Text = c[9].ToString();
102   getCompanyIndex();
103 }
```

Quellcode 4.0.3 Make Data

Mit der Funktion getCompanyIndex() werden die Vor- und Zurückbuttons sowie der dazugehörige Zähler gesteuert. Die Buttons werden deaktiviert wenn die Datenbank an das eine oder andere Ende der Liste gelangt ist. Der Zähler wird im Format “ID von TOTAL” dargestellt, wobei ID der Zählerstand der aktuellen Adresse und TOTAL die Gesamtzahl aller Adressen darstellt.

```
104 private void getCompanyIndex() {
105   var t = db.Select("SELECT id FROM company");
106   c = new ArrayList(t.Rows.Count);
107   foreach (DataRow row in t.Rows) c.Add(row.ItemArray[0]);
108   var currentID = c.IndexOf(id.Text) + 1;
109   count.Text = $"{currentID} von {c.Count}";
110   if (currentID == 1) previous.Enabled = false;
111     else previous.Enabled = true;
112   if (currentID == c.Count) next.Enabled = false;
113     else next.Enabled = true;
114 }
```

Quellcode 4.0.4 Firmenindex

Wird eine neue Firma angelegt oder eine bestehende bearbeitet, öffnet sich das Bearbeitenfenster. Dieses ist vom Aussehen her dem für das Neuerstellen identisch. Von der Funktion sind diese jedoch leicht verschieden. Der haubtsächliche Unterschied dabei ist die Interaktion mit der Datenbank - Neu einfügen gegenüber von Aktualisieren der Daten. Da das Firmenkürzel eine zwingende Angabe ist, wird bei der Prüfung dessen Vorhandensein gecheckt.

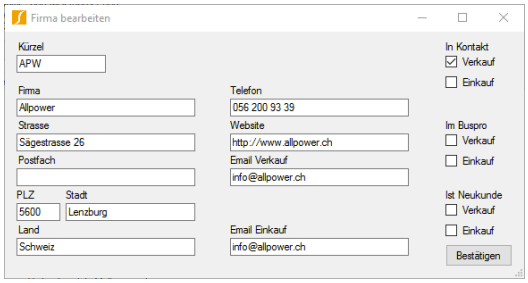

Abbildung 4.0.2 Firma bearbeiten

```
115 private void submitadd(object sender, EventArgs e) {
116   if (id.TextLength > 0) {
117   string query = $"INSERT INTO company VALUES ('{id.Text}', '{
        company.Text}', '{street.Text}', '{box.Text}', '{plz.
```

```
        Text}', '{city.Text}', '{country.Text}', '{phone.Text}',
         '{url.Text}', '{mail.Text}', '{history}')";
118     db.Insert(query); Close();
119     } else {
120     var result = MessageBox.Show(Resources.idRequired, "Fehlende
         Eingabe", MessageBoxButtons.OKCancel, MessageBoxIcon.
        Question);
121     }
122 }
```

Quellcode 4.0.5 Neue Firma

Bei einem Update einer Firma wird der id-Wert nicht überschrieben, daher findet keine Prüfung mehr statt.

```
123 private void submitChange(object sender, EventArgs e) {
124     string query = $"UPDATE company SET name='{company.Text}',
        street='{street.Text}', box='{box.Text}', plz='{plz.Text
        }', city='{city.Text}', country='{country.Text}', phone
        ='{phone.Text}', url='{url.Text}', mail='{mail.Text}',
        history = '{history}' WHERE id = '{id}'";
125     db.Update(query); Close();
126 }
```

Quellcode 4.0.6 Firma bearbeiten

Wenn eine in einer Pendenz verknüpfte Firmenkartei geöffnet wird, erscheint die jeweilige Adresskartei. Diese enthält die wichtigsten Infos über diejenige Firma. Diese Informateionen beinhalten unter anderem die komplette Postadresse, die Telefonnummer, Email und Webadresse. Die Kartei kann direkt aus diesem Fenster bearbeitet werden (Siehe Abbildung 4.0.2)

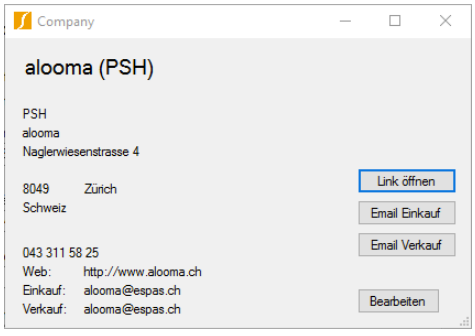

Abbildung 4.0.3 Adressenkartei

KAPITEL 5

ADMINISTRATION

5.1 ÜBERGABEN

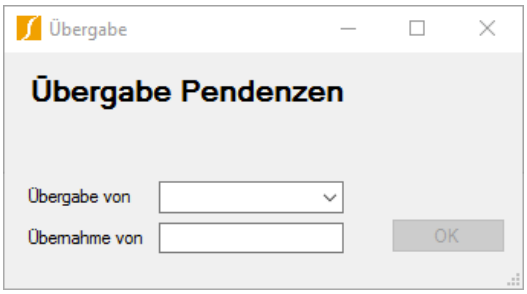

Abbildung 5.1.1 Übergabe Pendenzen

Durch die hohe Fluktuation kann es immer wieder vorkommen, dass Pendenzen von einer ausgetretenen Person noch nicht fertig bearbeitet wurden. Im Falle, dass es sich dabei nur um eine oder zwei Pendenzen handelt, kann dies einfach von Hand in den jeweiligen Pendenzen überschrieben werden. Falls dies aber mehrere Pendenzen betrifft, kann dies recht mühsam werden. Darum übernimmt das Modul "Übergabe Pendenzen" diese Arbeit. Ein einfacher Mitarbeiter kann dabei lediglich seine eigenen Pendenzen an eine andere Person übergeben. Die Geschäftsleitung besitzt allerdings die Berechtigung sämtliche Pendenzen von einem beliebigen Mitarbeiter weiterzugeben.

Sobald im Textfeld "Übergabe von" das Kürzel des Mitarbeiters eingetragen wurde, wird die Anzahl der Pendenzen, welche übergeben werden, im OK-Button als "OK (3)" angezeigt, wobei bei einem Mitarbeiter dieses Feld auf Grund der Benutzerrechte automatisch eingetragen wird (Siehe Funktion validate(), Code 5.1.1. Solange eine der beiden Eingabefelder leer ist, bleibt der OK-Button gesperrt.

```
127 private void validate() {
128   var count = db.Count($"SELECT COUNT(*) FROM pendenz WHERE
          sachbearbeiter = '{uebergabe.Text}'");
129   if (count > 0) {
130     okButton.Text = $"OK ({count})";
131     if (uebername.TextLength >= 3) {
132       okButton.Enabled = true;
133     } else { okButton.Enabled = false; }
134   } else {
135     okButton.Text = "OK";
136     okButton.Enabled = false;
137   }
138 }
```

Quellcode 5.1.1 Validierung zur Freigabe des OK Button

5.2 REORGANISATION

Um die Datenbank zu entlasten, respektive die Grösse der Datenbank zu kontrollieren, hat die Geschäftsleitung die Möglichkeit, alte Pendenzen zu löschen. Dabei wird zunächst ausgewählt, wie weit zurück die Pendenzen für den Löschvorgang berücksichtigt werden.

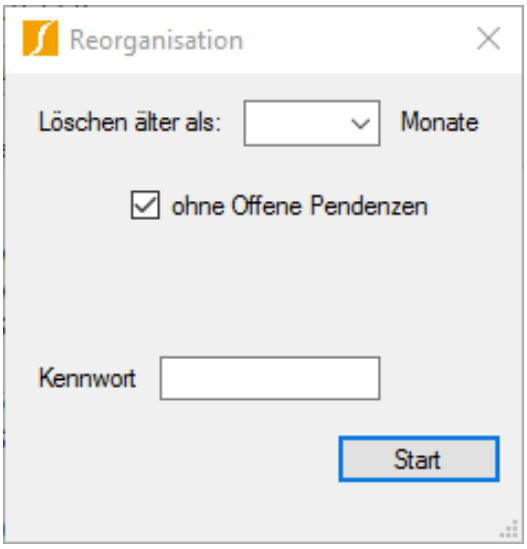

Abbildung 5.2.1 Reorganisation

Anschliessend kann entschieden werden, ob noch offene Pendenzen ebenfalls gelöscht werden sollten wobei dies standardmässig nicht geschieht. Bevor die Reorganisation gestartet werden kann, muss noch ein Kennwort eingegeben werden.

```
139 private void start(object sender, EventArgs e) {
140   if (authenticate()) {
141     if (openOnly.Checked) {
142     query = $"DELETE FROM pendenz WHERE erfasstAm < '{date}'
          AND state = 'done'";
143     count = $"SELECT COUNT(*) FROM pendenz WHERE erfasstAm <
          '{date}' AND state = 'done'";
144     } else {
145     query = $"DELETE FROM pendenz WHERE erfasstAm < '{date}'";
146     count = $"SELECT COUNT(*) FROM pendenz WHERE erfasstAm <
          '{date}'";
147     }
148     if (db.Count(count) > 0) {
149     info.Text = $"{db.Count(count)} Datensätze gelöscht.";
150     db.Delete(query);
151     } else { info.Text = $"Keide Datensätze zu löschen."; }
152   }
153 }
```

Quellcode 5.2.1 Löschvorgang

Das Kennwort wird mit SHA256 Verschlüsselt und mit dem in der Datenbank gespeicherten Hash des Kennworts verglichen.

```
154 private bool authenticate() {
155   query = $"SELECT AdminOption FROM admin WHERE idadmin = '
        password'";
156   var passTable = db.Select(query);
157   var passRow = passTable.Rows[0];
158   var passCell = passRow.ItemArray;
159   var pass = passCell[0].ToString();
160   if (crypt.Encrypt(password.Text) == pass)  return true;
161   return false;
162 }
```

Quellcode 5.2.2 Ableich des Password-Hash

5.3 STATISTIK

Die Statistik erlaubt es, mit einem kurzen Blick, die wichtigsten Zahlen zu betrachten. Die folgenden Zahlen werden angezeigt:

- Anzahl offener Pendenzen
- Anzahl abgeschlossener Pendenzen
- Anzahl stornierter Pendenzen
- Anzahl neu eröffneter Pendenzen

Diese Zahlen werden, mit Ausnahme der offenen Pendenzen, sowohl für den aktuellen wie auch für den vergangenen Monat angezeigt. Dabei werden jeweils nur die Zahlen der eigenen Abteilung abgebildet (ausgenommen Geschäftsleitung).

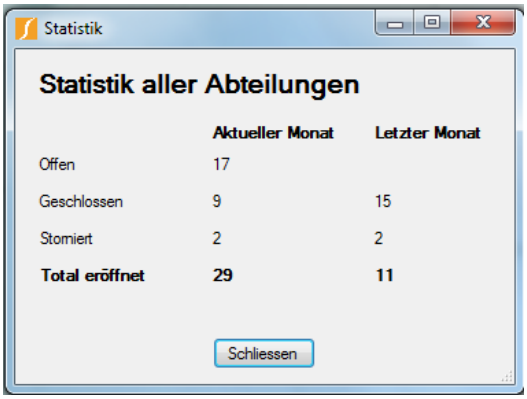

Abbildung 5.3.1 Statistik

KAPITEL 6

GLOBALE FUNKTIONEN

6.1 BENUTZERRECHTE

Die Benutzerrechte sind an den Eintrag im AD geknüpft (Siehe dazu Abschnitt 6.2). Diese Information wird in der Funktion getInfo(**string** id) abgefragt und als fünftes Element (Index [4]) verfügbar. Die konkreten Rechte werden im Programm an die Abteilung geknüpft welche gegen das bezogene Element abgeglichen wird.

Ein Mitarbeiter der Abteilung Einkauf sieht also ausschliesslich die Pendenzen dieser Abteilung. Das gleiche gilt für jede andere Abteilung. Die Geschäftsleitung hat jedoch Zugriff auf die Pendenzen sämtlicher Abteilungen.

6.2 BENUTZERINFORMATIONEN

Die Applikation holt sich den Benutzername des aktuell angemelden Benutzer (getID()). Mit dieser Information holt sich die Funktion getInfo(**string** id) die dazugehörigen Daten aus dem AD.

```
163 public static string getID() {
164   return WindowsIdentity.GetCurrent().Name.Split('\\').Last();
165 }
```

Quellcode 6.2.1 Benutzernamen auslesen

```
166 public static List<string> getInfo(string id) {
167   DirectoryEntry e = new DirectoryEntry("LDAP://server.local",
        "username", "password");
```

```
168     DirectorySearcher s = new DirectorySearcher(e);
169     s.Filter = $"(&(objectClass=user)(samAccountName={id}))"
170     s.SearchScope = SearchScope.Subtree;
171     List<string> list = new List<string>();
172     SearchResult r = s.FindOne();
173     // Benutzerkürzel
174     list.Add(r.Properties["samAccountName"][0].toString());
175     // Vorname
176     list.Add(r.Properties["givenName"][0].toString());
177     // Familienname
178     list.Add(r.Properties["sn"][0].toString());
179     // Email-Adresse
180     list.Add(r.Properties["mail"][0].toString());
181     // LDAP Pfadnamen -> Abteilung
182     list.Add(r.Properties["distinguishedname"][0].toString().
            Split(',')[1].Split('=').Last();
183     return List
184 }
```

Quellcode 6.2.2 Personeninfos

Damit nicht jedes Mal zuerst getID()) und anschliessend getInfo(**string** id) aufgerufen werden muss, kann die Funktion getInfo() (ohne Argument “id”) aufgerufen werden. Dies führt die beiden Funktionen kombiniert aus.

```
185 public static List<string> getInfo() {
186     return getInfo(getID());
187 }
```

Quellcode 6.2.3 Kombinierte Abfrage

KAPITEL 7

SCHLUSSFOLGERUNG

Die Produktion der Pendenzen Applikation stellte eine Herausforderung dar. Dies kann einerseits durch das relativ neue Arbeiten mit C# und SQL begründet werden. Anderseits aber auch durch die Komplexität der Module und der verschiedenen Berechtigungen.

Durch das schrittweise Vorantasten entstanden Modul um Modul sowie Funktion um Funktion. Verschiedene Male wurden Funktionen refakturiert um einerseits die Lesbarkeit zu verbessern aber andererseits auch um die Möglichkeit zubekommen die Funktionen, respektive Teile der Funktionen in mehreren Varianten anzusteuern. Dadurch wurde erreicht, dass Teile des Source Codes nicht mehrere Male geschrieben werden musste.

Das direkte Arbeite mit den Abteilungen brachte den Vorteil, dass die App von den zukünftigen Benutzern getestet und ausprobiert werden konnte. So konnten Fehler in Design und Funktion schnell gefunden und Wünsche, sofern möglich, schnell eingearbeitet werden.

GLOSSAR

Active Directory

Active Directory heißt der Verzeichnisdienst von Microsoft Windows Server. Active Directory ermöglicht es, ein Netzwerk entsprechend der realen Struktur des Unternehmens oder seiner räumlichen Verteilung zu gliedern. Dazu verwaltet es verschiedene Objekte in einem Netzwerk wie beispielsweise Benutzer, Gruppen, Computer, Dienste, Server, Dateifreigaben und andere Geräte wie Drucker und Scanner und deren Eigenschaften. Mit Hilfe von Active Directory kann ein Administrator die Informationen der Objekte organisieren, bereitstellen und überwachen. Den Benutzern des Netzwerkes können Zugriffsbeschränkungen erteilt werden. So darf zum Beispiel nicht jeder Benutzer jede Datei ansehen oder jeden Drucker verwenden. 13, 29

AD

Active Directory. 13, 25, 29, *Siehe:* Active Directory

C#

C# (C Sharp) ist eine von Microsoft entwickelte Programmiersprache. 6, 7, 27

CI

Corporate Identity. 6, 29, *Siehe:* Corporate Identity

Corporate Identity

Ist die Gesamtheit der Merkmale, die ein Unternehmen kennzeichnen und es von anderen Unternehmen unterscheiden. 6, 29

Hash

Eine Hashfunktion ist eine Funktion, die eine Zeichenfolge beliebiger Länge auf eine Zeichenfolge mit fester Länge, der Hash, abbildet.. i, 23

IDE

Integrierte Entwicklungsumgebung. 5, 30, *Siehe:* Integrierte Entwicklungsumgebung

Integrierte Entwicklungsumgebung

Eine integrierte Entwicklungsumgebung (Abkürzung IDE, von englisch integrated development environment) ist eine Sammlung von Anwendungsprogrammen, mit denen die Aufgaben der Softwareentwicklung möglichst ohne Medienbrüche bearbeitet werden können. 5, 30

JavaScript

eine Skriptsprache, die ursprünglich für dynamisches HTML in Webbrowsern entwickelt wurde, um Benutzerinteraktionen auszuwerten, Inhalte zu verändern, nachzuladen oder zu generieren und so die Möglichkeiten von HTML und CSS zu erweitern. 5

Multithreading

bezeichnet das gleichzeitige (oder quasi-gleichzeitige) Abarbeiten mehrerer Threads (Ausführungsstränge) innerhalb eines einzelnen Prozesses. 11

PHP

ist eine Skriptsprache, die hauptsächlich zur Erstellung dynamischer Webseiten oder Webanwendungen verwendet wird. 5, 6

Query

Ein präziser Antrag auf Informationsabfrage mit Datenbank- und Informationssystemen. 13, 15, 18

relationale Datenbank

digitale Datenbank die Daten in einer oder mehreren Tabellen speichert. Jeder Datensatz besitzt einen eindeutigen Schlüssel. Jede relationale Datenbank verwendet SQL als Sprache zur Abfrage und Pflege. 5, 7

SHA256

engl. secure hash algorithm; kryptologischen Hashfunktion zum Verschlüsseln digitaler Daten. 23

Source Code

engl. Quelltext; in einer Programmiersprache geschriebene Text eines Computerprogrammes. 6, 27

SQL

ist eine Datenbanksprache zur Definition von Datenstrukturen in relationalen Datenbanken sowie zum Bearbeiten (Einfügen, Verändern, Löschen) und Abfragen von darauf basierenden Datenbeständen. 5, 6, 13, 15, 27, 31

String

engl. Zeichenkette; eine Folge von Zeichen (z. B. Buchstaben, Ziffern, Sonderzeichen und Steuerzeichen). 8

Workstation

engl. Arbeitsstation; ein Arbeitsplatzrechner. 6

Weitere Publikationen

Project: UserSetup Allpower (2016)

Project: Bestellformular Allpower (2016)

Project: Intranet Allpower (2016)

Copyright © 2017 Christian Seiler

Alle Rechte vorbehalten.

www.christianseiler.ch

LaTeX