Projekt Alpha

Ein Adressbuch in Form einer Konsolenapplikation im Rahmen einer Projektarbeit von Medin Tafic

Inhalt

1 Management Summary

Wir, Studierende der TEKO erhielten im 3. Semester den Auftrag, im Rahmen einer Projektarbeit eine Konsolenapplikation zu entwickeln. Es geht dabei im Wesentlichen um die Anwendung des Grundlagenwissens des Faches «Projektarbeit» sowie dem vermittelten Unterrichtsstoff des Faches «Programmieren». Ich habe dieses Projekt «Alpha» getauft, weil dies mein allererstes Projekt im Programmieren ist und Alpha der erste Buchstabe im griechischen Alphabet ist.

Die folgenden Seiten beinhalten die Dokumentation der Projektarbeit in ihren einzelnen Schritten. Diese Schritte sehen wie folgt aus: Initialisierung, Planung, Realisierung und Abschluss.

In der Initialisierung geht es um den Auftrag und die dazu benötigten Ziele. Diese wurden uns von unserem Auftraggeber, welcher zugleich unser Dozent im Fach «Programmieren» ist, vorgegeben.

Da ich dieses Projekt nebenberuflich und fast ausschliesslich in meiner Freizeit erarbeitet habe, ist Planung ein wichtiger Punkt. In diesem Teil ist die Zeitplanung aufgeführt, zu welchem Zeitpunkt was erarbeitet wurde.

In der Realisierungsphase war eine Ideensammlung und Variantenauswertung von sinnloser Bedeutung, da für mich von Anfang an klar war welche Variante ich aussuchen würde. Die Varianten wurden uns vor Beginn der Projektarbeit bereits kommuniziert. Dementsprechend besteht dieser Teil fast ausschliesslich aus der Ausgestaltung der Konsolenapplikation.

Zum Schluss der Arbeit wurden die Ziele ausgewertet und anschliessend eine Reflexion in Form eines Fazits festgehalten.

2 Projektinitialisierung

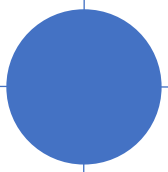
2.1 Zieldefinition (Coverdale Zielscheibe)

- Eine Dokumentation mit folgenden Inhalten:
  - Auftrag
  - Komplette Zielscheibe
  - Projektplanung
  - Vollständige Dokumentation der Projektarbeit
  - Reflexion Weg zum Ziel
- Lauffähiger Source Code

Die schweizerische Fachschule TEKO.

Endergebnis

Kunde



Sinn und Zweck

Erfolgskriterien

Im Fach Programmieren erhalten die Studierenden den Auftrag, das erforderliche Grundlagenwissen zu einem klar abgegrenzten Thema aufzubauen und anzuwenden. Dabei geht es für sie im Wesentlichen um die konsequente Zielorientierung, die Anwendung der Phasen des Projektmanagements auf der Grundlage des vermittelten Unterrichtsstoffes an der HF sowie die konsequente Umsetzung von

- Die Grundsätze des Projektmanagements (Initialisierung, Planung, Realisierung, Abschluss) werden berücksichtigt
- Die Dokumentation ist vollständig, verständlich und nachvollziehbar
- Qualität des abgegebenen Source Code (Sinnvolle Struktur, Dokumentation, ….)
- Der Gesamteindruck der Projektdokumentation (Aufbau, Gestaltung, Rechtschreibung, Verständlichkeit)

Abbildung 1 - Zieldefinition

3 Projektplanung

3.1 Projektstrukturplan

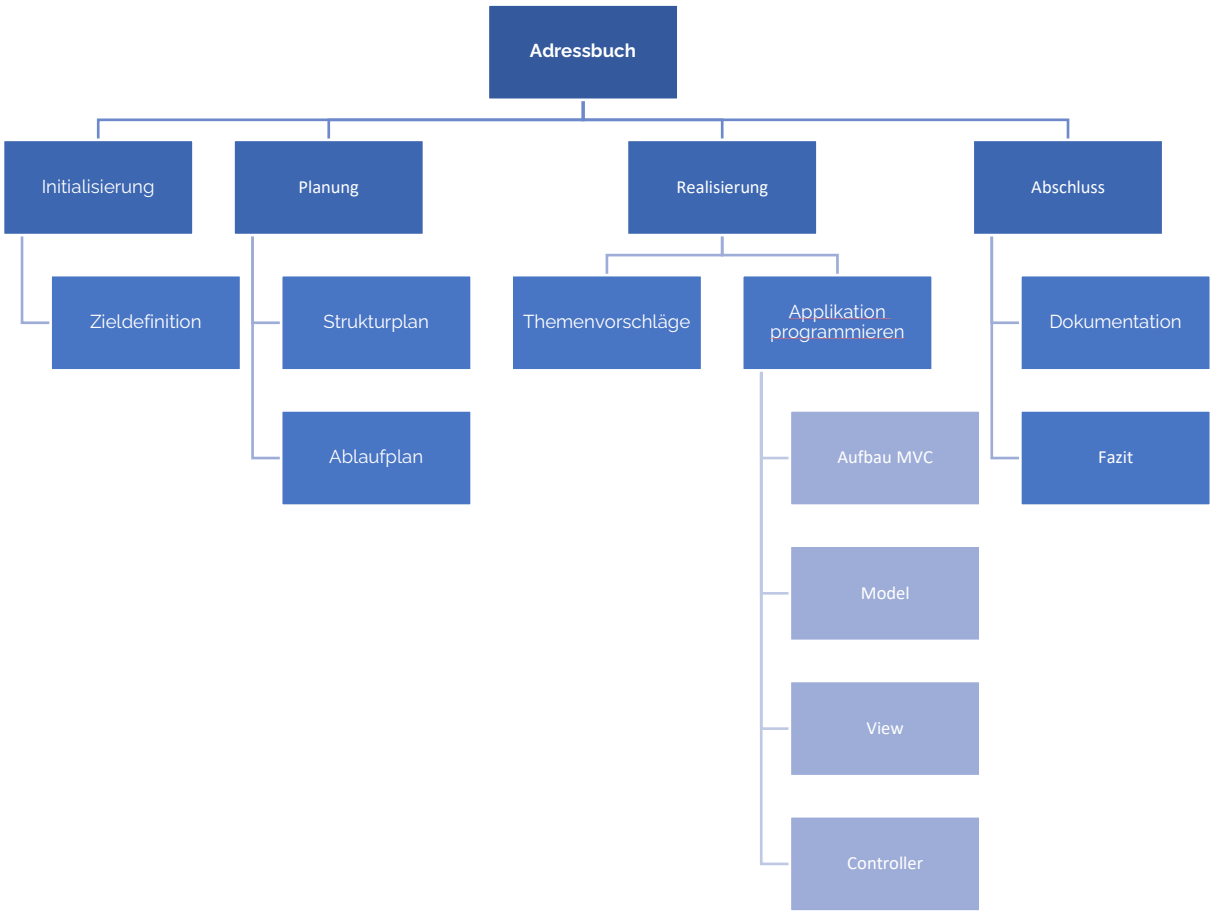


Abbildung 2 - Projektstrukturplan

3.2 Projektablaufplan

Arbeitspakete	Start	Dauer	Ende	Kalenderwoche							
		Wochen		5	6	7	8	9	10	11	12
Auftragsklärung	05.02.2022	1	12.02.2022								
Zieldefinition	05.02.2022	1	12.02.2022								
Abschluss Initialisierung	12.02.2022	1	19.02.2022								
Strukturplan	12.02.2022	1	19.02.2022								
Ablaufplan	12.05.2022	1	19.02.2022								
Abschluss Planung	12.05.2022	1	19.02.2022								

Tabelle 1 - Projektablaufplan 1

Arbeitspakete	Start	Dauer	Ende	Kalenderwoche							
		Wochen		5	6	7	8	9	10	11	12
Phasenplan für Umsetzung	19.02.2022	2	05.03.2022								
Programmieren	19.02.2022	5	19.03.2022								
Abschluss Realisierung	19.03.2022	1	19.03.2022								
Dokumentation	12.03.2022	3	26.03.2022								
Fazit	24.03.2022	-	26.03.2022								
Abschluss Projekt	24.03.2022	-	26.03.2022								

Tabelle 2 - Projektablaufplan 2

4 Projektrealisierung

4.1 Themenvorschläge

4.1.1 Vorschlag 1 – Adressbuch:

Interaktive Konsolenapplikation mit folgender Funktionalität:

- View all contacts (Alle Kontakte in der Konsole anzeigen)
- Add new contact (Neuen Kontakt hinzufügen)
- Edit existing contact (Kontakt anhand ~~des Namens~~ der Index-Nummer bearbeiten)
- Remove existing contact (Kontakt anhand ~~des Namens~~ der Index-Nummer löschen)
- Terminate application (Applikation beenden

Weiteres:

- Persistierung (Speicherung) der Daten in einer .csv Datei
- Empfehlung: Verwendung von MVP Pattern.

4.1.2 Vorschlag 2 – To-Do-Liste

Interaktive Konsolenapplikation mit folgender Funktionalität:

- View all tasks (Alle Aufgaben in der Konsole anzeigen
- Add new task (Eine Aufgabe hinzufügen)
- Remove a task (Eine Aufgabe entfernen)
- Mark task as completed (Aufgabe als erledigt markieren)
- Mark task as incomplete (Aufgabe als unvollständig markieren)
- Terminate application (Applikation beenden)

Weiteres:

- Persistierung (Speicherung) der Daten in einer .csv Datei
- Empfehlung: Verwendung von MVC Pattern

4.1.3 Vorschlag 3 – Berechnung von Statistik-Metriken von Sensormessdaten

Interaktive Konsolenapplikation mit folgender Funktionalität:

- Read data
  - Input Pfad zur .csv-Datei mit rohen Messdaten einer Messreihe
  - Lesen von Daten aus .csv-Datei
- Calculate statistic metrics
  - Berechnung der Statistik Metriken (Durchschnitt, Min, Max, Median)
- Show results
  - Darstellung von Ergebnis in Konsole
- Save results to file
  - Benutzer kann das Ergebnis sich noch speichern lassen in eine Output Datei (.txt)
- Terminate application

Weiteres:

- Testen mit min. drei zur Verfügung gestellten Dateien
- Empfehlung: Verwendung von MVC Pattern

4.2 Entscheidung

Für mich war sofort klar, dass ich mich für den Vorschlag 1 «Adressbuch» entscheiden werde. Ich habe mich für das Adressbuch entschieden, weil es etwas ist, was man täglich im Alltag braucht und auch noch später beliebig erweitert werden kann mit verschiedenen weiteren Funktionalitäten, einer GUI (Graphic user interface), etc. Jedoch habe ich in Absprache mit dem Auftraggeber/Dozenten eine Anpassung in der Funktionalität vornehmen müssen.

Die Funktionen «Edit existing contact» und «Remove existing contact» wurden in der Anforderung angepasst. Ursprünglich war bei beiden Funktionen eine Abfrage anhand des Namens vorgesehen. Hier fiel mir auf, dass eine Abfrage anhand des Namens wenig Sinn macht, weil es keine eindeutige Information ist. Es kann sein, dass man mehrere Kontakte mit demselben Namen hat. Was jedoch eindeutig ist, ist die Index-Nummer der einzelnen Kontakte (siehe Bild).

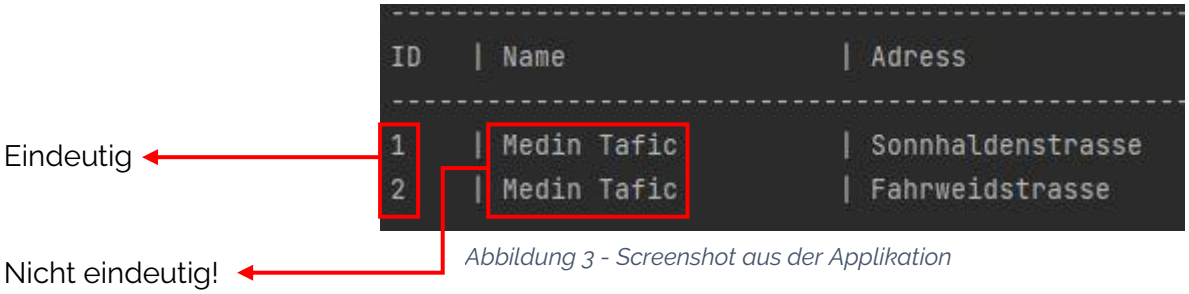


Abbildung 3 - Screenshot aus der Applikation

4.3 Ausarbeitung Konzept

4.3.1 Realisierungsplan zur praktischen Umsetzung der Applikation

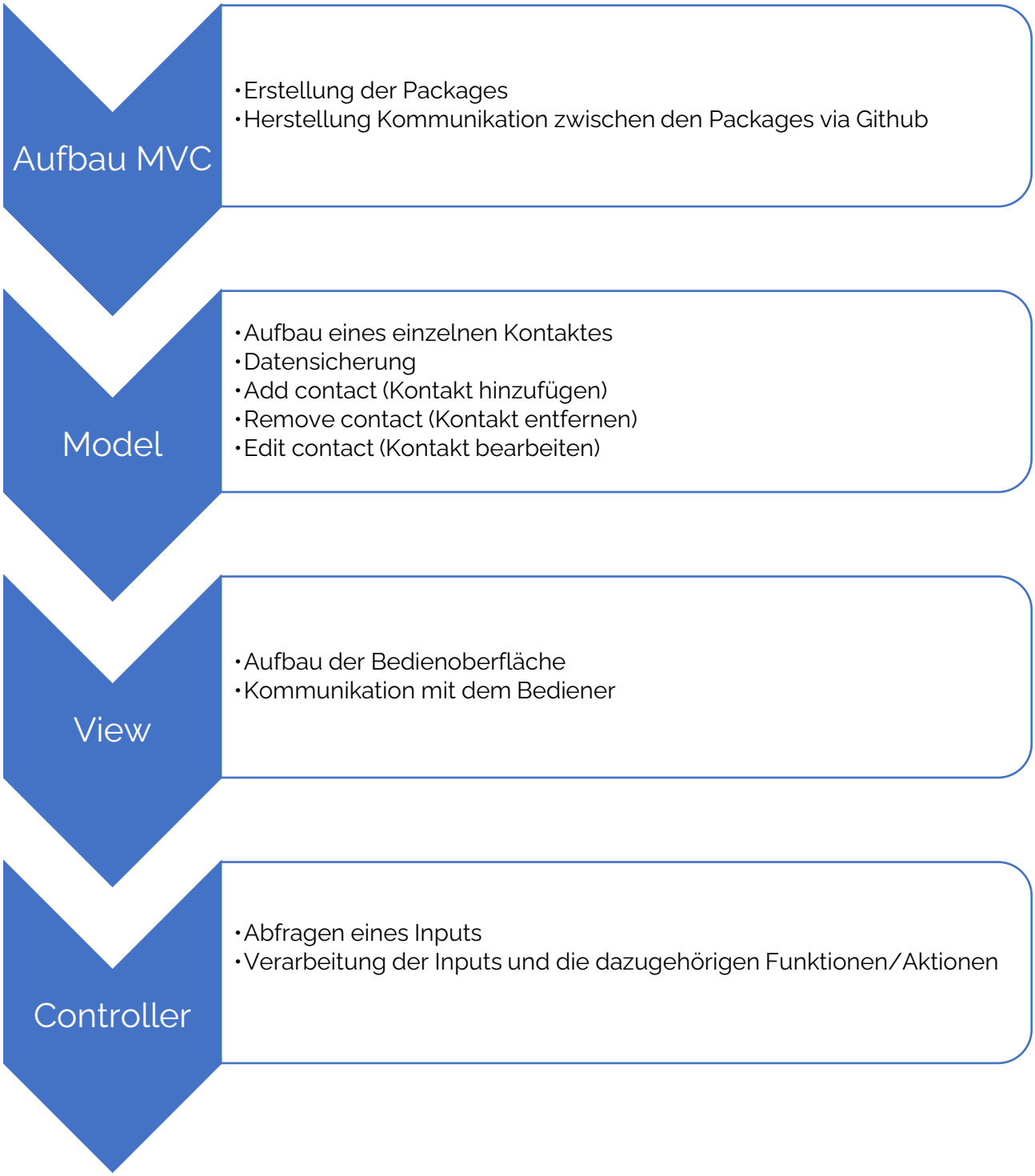


Abbildung 4 – Phasenplan

4.3.2 Aufbau MVC

Was ist MVC und wie funktioniert es?

Model View Controller ist ein Muster zur Unterteilung einer Software in die drei Komponenten: Datenmodell (Model), Präsentation (View) und Programmsteuerung (Controller). In der folgenden Darstellung ist der Aufbau des MVC grafisch dargestellt.

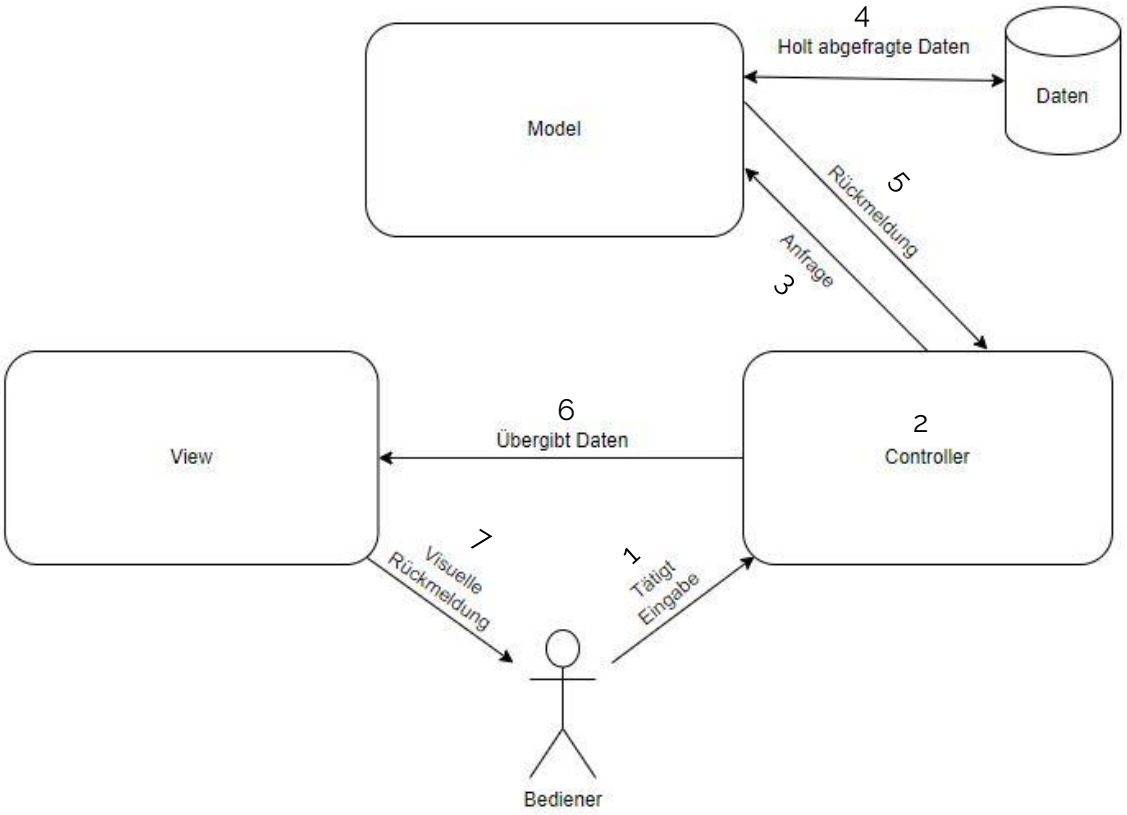


Abbildung 5 - MVC Pattern (selber erstellt)

Beispiel Adressbuch:

1. Der Bediener möchte die Kontaktliste öffnen.
2. Dieser Input wird von „Controller“ verarbeitet.
3. Der „Controller“ sendet eine Anfrage an „Model“.
4. „Model" holt sich die angefragten Daten aus der Datenbank
5. „Model“ gibt die Daten an „Controller“ zurück
6. „Controller“ gibt die Daten an „View“ weiter.
7. „View“ zeigt dem Bediener die angeforderte Kontaktliste an.

Das Projekt wird zugunsten der Übersicht genau gleich wie in der obigen Darstellung in die drei Komponenten aufgeteilt in Form von Ordnern. In den Ordnern befindet sich dann jeweils eine .go Datei mit dem Quellcode.

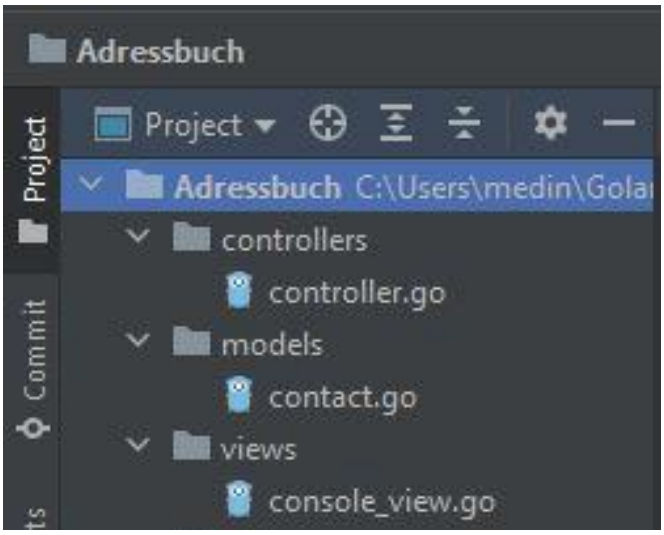


Abbildung 6 - Screenshot von der Ordnerstruktur

```
// Package controllers contains the contact list controllers
package controllers

import (
    "bufio"
    "github.com/Medinolas/Adressbuch/models"
    "github.com/Medinolas/Adressbuch/views"
    "log"
    "os"
    "strings"
    "time"
)
```

Damit Funktionen aus einem Paket in einem anderen Paket verwendet werden können, müssen die Funktionen importiert werden. Da ich vor habe dieses Projekt nach Abschluss des Studiums zu behalten und eventuell weiterzuentwickeln, habe ich GoLand mit Github verknüpft. Somit kann ich die Github URL «importieren» lassen von GoLand um auf die Funktionen zugreifen zu können.

4.3.3 Model

Im Paket «Model» dreht sich alles um den Aufbau des Kontaktes, der Datensicherung und der Kommunikation zur «Datenbank», in diesem Fall eine .csv Datei.

```
const FileName = "data.csv"

// Contact repository
var contacts []Contact

// Contact persistence
var filePersistence bool = false

// Contact as type
type Contact struct {
   Name        string
   StreetName  string
   HouseNumber string
   PostalCode  string
   CityName    string
   PhoneNumber string
   EMailAdress string
}

// EnableFilePersistence enables the file persistence
func EnableFilePersistence() {
   filePersistence = true
}

// DisableFilePersistence disables the file persistence
func DisableFilePersistence() {
   filePersistence = false
}
```

Zuerst wird die «Datenbank» erstellt «data.csv».

Der Kontakt wurde als Typ «struct» erstellt mit den Informationen wie Name, Adresse, etc als Attribute.

Zur Aktivierung und Deaktivierung der Persistierung wurden einzelne Funktionen erstellt.

```
// AddContact adds a contact to the list
func AddContact(contact Contact) bool {
   if contact.Name == "" || contact.StreetName == "" || contact.HouseNumber
== "" || contact.PostalCode == "" || contact.CityName == "" ||
contact.PhoneNumber == "" || contact.EMailAdress == "" {
      return false
   }

   contacts = append(contacts, contact)

   if filePersistence {
      updateDataInFile()
   }

   return true
}
```

AddContact überprüft, ob alle Felder mit dem richtigen Typ gefüllt werden und erweitert anschliessend den Slice (contacts) um den neu erstellten Kontakt.

```
// RemoveContact removes a contact from the list
func RemoveContact(rowIndex string) {
   var tempContacts []Contact
   idAsInt := ToInt(rowIndex) - 1

   for index, currentContact := range contacts {
      if index != idAsInt {
         tempContacts = append(tempContacts, currentContact)
      }
   }

   contacts = tempContacts

   if filePersistence {
      updateDataInFile()
   }
}
```

RemoveContact orientiert sich, wie in der Variantenentscheidung erwähnt, nicht an den Namen des Kontaktes, sondern an der Index Nummer (index). Hier iteriert das Programm durch die Kontaktliste (contacts) und vergleicht die Index Nummer jedes Kontaktes mit der vom Benutzer eingegeben zu löschenden Kontakt. Danach wird die Kontaktliste neu erstellt ohne den zu löschenden Kontakt.

Anschliessend die .csv Datei aktualisiert.

```
// EditContact chooses the contact you want to edit
func EditContact(i string) {
   idAsInt := ToInt(i) - 1

   for index, currentContact := range contacts {
      if index == idAsInt {
         fmt.Println(currentContact)
      }
   }

}

// EditFieldContact changes a field in the contact information
func EditFieldContact(rowIndex, field, newInfo string) {
   idAsInt := ToInt(rowIndex) - 1

   for index, _ := range contacts {
      if index == idAsInt {
         switch {
         case field == "name":
            contacts[index].Name = newInfo
         case field == "adress":
            contacts[index].StreetName = newInfo
         case field == "house number":
            contacts[index].HouseNumber = newInfo
         case field == "postal code":
            contacts[index].PostalCode = newInfo
         case field == "city name":
            contacts[index].CityName = newInfo
         case field == "phone number":
            contacts[index].PhoneNumber = newInfo
         case field == "e-mail adress":
            contacts[index].EMailAdress = newInfo
         }
      }
   }
   if filePersistence {
      updateDataInFile()
   }
   fmt.Println("Contact updated.")
}
```

Mit EditContact wählt man den Kontakt aus, den man bearbeiten möchte. Diese Funktion wurde um eine bessere Übersicht zu gewährleisten als Zwischenschritt erstellt.

EditFieldContact verlangt drei Informationen: die Index Nummer (rowIndex), welches Feld man bearbeiten möchte (field) und die neue Information, welche die zu ändernde Information ersetzen soll (newInfo).

4.3.4 View

Im «View» Paket geht es um die visuelle Darstellung von Daten sowie der Kommunikation zum Bediener. Da es sich hier um eine Konsolenapplikation handelt, ist man in der Gestaltung der Benutzeroberfläche stark eingeschränkt.

```
// PrintMenu prints the menu to the console
func PrintMenu() {
   fmt.Println(`
   ################################################
   #                                              #
   #          WELCOME TO YOUR CONTACT LIST        #
   #           CHOOSE YOUR OPTION BELOW           #
   # -------------------------------------------- #
   #  1. ADD ANOTHER CONTACT IN CONTACT LIST      #
   #  2. REMOVE A CONTACT FROM THE LIST           #
   #  3. VIEW ALL CONTACTS                        #
   #  4. EDIT CONTACT                             #
   #  c. CLEAR VIEW AND PRINT MENU                #
   #  q. TERMINATE CONTACT LIST APP               #
   #                                              #
   ################################################
   `)
}
```

Der grüne Teil des Codes soll das Hauptmenü der Applikation darstellen, womit der Benutzer hauptsächlich das Programm bedient.

```
// PrintContactList prints all the contacts in the contact list to the
console
func PrintContactList(contactsToPrint []models.Contact) {

   fmt.Println("-------------------------------------------------------------
-------------------------------------------------------------------------
")
   fmt.Printf("%-4s | %-20s | %-20s | %-11s | %-11s | %-15s | %-12s | %-
25s\n", "ID", "Name", "Adress", "House number", "Postal code", "City name",
"Phone number", "E-Mail adress")
   fmt.Println("-------------------------------------------------------------
-------------------------------------------------------------------------
")

   for i, contact := range contactsToPrint {

      fmt.Printf("%-4d | %-20s | %-20s | %-12s | %-11s | %-15s | %-12s | %-
25s\n", i+1, contact.Name, contact.StreetName, contact.HouseNumber,
contact.PostalCode, contact.CityName, contact.PhoneNumber,
contact.EMailAdress)
   }
}
```

PrintContactList iteriert durch die Kontaktliste und gibt jeden Kontakt mit dessen Daten an den Benutzer weiter.

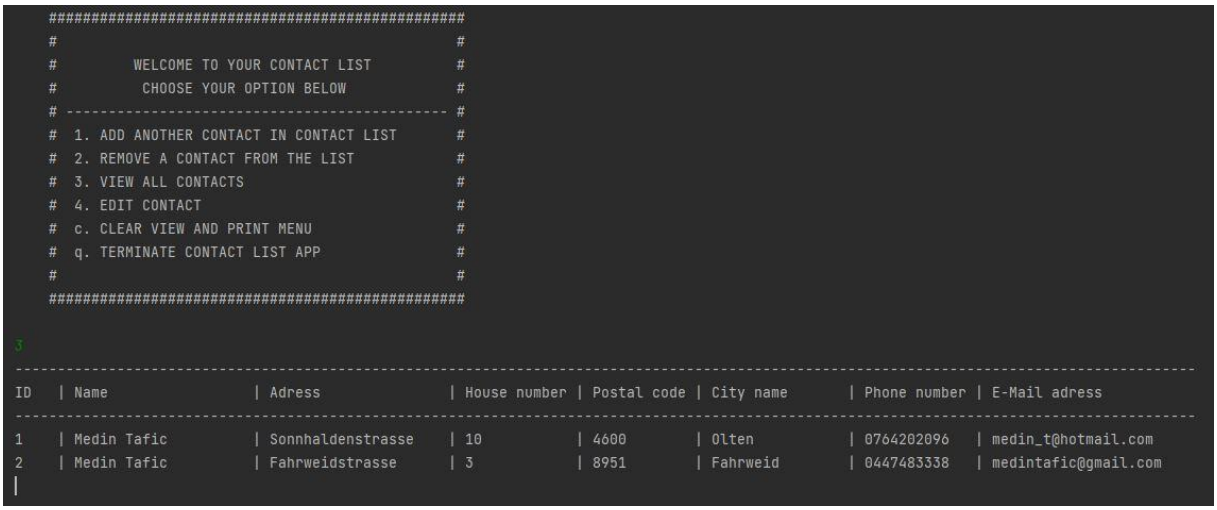


Abbildung 7 - Screenshot aus der Konsole

Im obenstehenden Bild sieht man wie eine Interaktion mit dem Programm stattfindet. Zuerst erscheint das Hauptmenü, anschliessend gibt man in der Konsole die gewünschte Funktion an. In diesem Fall wurde die Kontaktliste aufgerufen und unterhalb dargestellt.

```
// PrintContactInformation asks you to enter the new contact information
func PrintContactInformation() {
   fmt.Println("Please enter the new contact with the following format:
Full name, Adress, House Number, Postal Code, City Name, Phone Number, E-
Mail Adress")
}

// PrintRemoveInformation asks for the name of the contact you want to
remove from the list
func PrintRemoveInformation() {
   fmt.Println("Please enter the index number of the contact you want to
remove.")
}

// EditContactInformation asks which contact you want to edit
func EditContactInformation() {
   fmt.Println("Please enter the index number of the contact you want to
edit.")
}
```

Hier noch einige Beispiele von Funktionen welche zur Kommunikation mit dem Benutzer dienen.

4.3.5 Controller

Im Controller findet die ganze Verarbeitung der eingegeben Werte statt, sowie das Zusammenspiel aller Komponenten.

```
func askForInput() string {
        reader := bufio.NewReader(os.Stdin)
        response, err := reader.ReadString('\n')
        if err != nil {
        log.Fatal(err)
   }
   response = strings.TrimSpace(response)
   return response
}
```

askForInput wie es der Name schon sagt ist die Funktion, welche eine Eingabe des Bedieners liest und in eine Variable (response) steckt mit der dann gearbeitet werden kann.

```
func parseCommand(input string) {
      switch {
   case input == "1":
      // Add another contact
      //
      views.Clear()
      views.PrintContactInformation()
      response := askForInput()
      contact := createContact(response)
      models.AddContact(contact)
      views.Clear()
      views.PrintMenu()
      break
   case input == "2":
      // Remove a contact
      //
      views.Clear()
      contacts := models.FindAllContacts()
      views.PrintContactList(contacts)
      views.PrintRemoveInformation()
      response := askForInput()
      models.RemoveContact(response)
      views.Clear()
      views.PrintMenu()
      break
```

Da der switch case sechs verschiedene Fälle hat, wird der Quelltext sehr lang und nimmt viel zu viel Platz in der Dokumentation. Darum hier ein Ausschnitt der ersten beiden Fälle um den Ablauf und die Zusammenarbeit der verschiedenen Komponenten aufzuzeigen.

Beispiel «case input == «2»: Der Bediener will einen Kontakt entfernen.

Alle Kontakte werden an contacts übergeben, contacts wird ausgegeben und der Bediener wird gefragt welchen Kontakt er entfernen möchte.

Danach erfolgt eine Eingabe des Bedieners.

Anschliessend wird diese Eingabe verarbeitet, der Kontakt gelöscht und das Programm springt zurück ins Hauptmenü.

5 Projektabschluss

5.1 Auswertung Zielerreichung

Endergebnisse	Erfolgskriterien	Ziel erreicht?
1. Der Source Code (als ZIP-File) und die vollständige Dokumentation sind abzugeben.	Die Dateien sind bis am 26.03.2022 – 23:59 Uhr abgegeben.	**Durch Auftraggeber zu beurteilen**
2. Der Code im Goland ist dokumentiert via Kommentare	Die Kommentare beschreiben den nachfolgenden Code	**JA**
3. Die wichtigsten Funktionen und Codezeilen sind in der Projektdokumentation beschrieben.	Ein Ausschnitt des Codes ist im Text eingefügt und in eigenen Worten beschrieben.	**JA**
4. View all contacts (Alle Kontakte in der Konsole anzeigen) ist implementiert	Alle Kontakte werden angezeigt	**JA**
5. Add new contact (Neuen Kontakt hinzufügen)	Ein neuer Kontakt wird hinzugefügt.	**JA**
6. Edit existing contact (Kontakt anhand der Index Nummer bearbeiten)	Ein bestehender Kontakt kann bearbeitet werden.	**JA**
7. Remove existing contact (Kontakt anhand der Index Nummer löschen)	Ein bestehender Kontakt kann entfernt werden.	**JA**
8. Persistierung (Speicherung) der Daten in einer .csv Datei	Die Kontakte sind in einer .csv Datei gesichert.	**JA**

Tabelle 3 - Auswertung Zieldefinition

5.2 Fazit / Lessons Learned

Diese Projektarbeit hat mich gefordert, jedoch im guten Sinne. Man lernt, den gelernten Unterrichtsstoff zu kombinieren und korrekt anzuwenden. Endlich finden die einst auf den ersten Blick sinnlos scheinenden Funktionen einen Sinn und Platz.

Anfangs hatte ich Schwierigkeiten bei der Implementierung der einzelnen Funktionen. Die schwierigste Frage, die ich mir immer gestellt habe, ist: «Wo fange ich an?». Wenn man jedoch einmal begonnen hat, fallen einem die folgenden Implementierungen viel einfacher. Bei der Erstellung der Funktionen musste ich mich auch ab und zu Fragen, ob dies in «Model», «View» oder «Controller» gehört. Hier hat die Abbildung des MVC Patterns sehr geholfen, solche Gedankengänge kurz zu halten. Gerne hätte ich mir noch gewünscht, dieser Applikation eine GUI hinzuzufügen.

Was ich als angehender Applikationsentwickler schon gelernt habe ist: «Das Programm ist nie fertig.». Obwohl die Ziele erreicht sind, findet man immer etwas was man noch besser, noch einfacher oder noch effizienter gelöst haben möchte. Diese Projektarbeit hat mir grossen Spass gemacht und ich freue mich auf die nächsten Projekte.

6 Anhang

6.1 Abbildungsverzeichnis

6.2 Tabellenverzeichnis