Erarbeitung der Hausaufgabe 1 im Fach “Einführung in die Programmierung”
Abgabe von Paul Friedrich Vierkorn.

Aufgabe 1

F1	Patienten sollen eine Wartenummer zugewiesen werden
F2	Patienten sollten mit dieser Wartenummer aufgerufen werden
F3	Patienten sollen nach dem Aufruf aus der Warteschlage entfernt werden.
F4	Patienten sollen auch außerhalb der eingegebenen Reihenfolge aufrufbar sein.
F5	Das Tool erlaubt Patienten zu erkennen, wann sie Aufgerufen werden.
F6	Die Wartenummern von allen wartenden Patienten sollen im Wartezimmer für alle angezeigt werden. Dies erlaubt Patienten zu erkennen, ob die Ihnen genannte Nummer tatsächlich registriert wurde.
N1	Das Tool soll das Management von Wartenummern vereinfachen.
N2	Das Tool läuft auf mehreren Betriebssystemen.
N3	Das Tool erwartet keine Eingabe von Patienten, ausschließlich vom Personal.
N4	Das Tool übermittelt alle Daten als Plaintext, um das Management von Daten auch für die Endnutzer zu vereinfachen.
N5	Das Tool läuft mit Fehlerbehandlungen, sodass eventuelle Programm- und Eingabefehler nicht zum Absturz führen.
N6	Das Tool arbeitet DSGVO-Konform.
N7	Der Quellcode des Programmes soll verfügbar gemacht werden können.
N8	Das Tool ist leicht erweiterbar, um Anbindungen an eventuell bestehende Daten in das Tool zu integrieren.

Aufgabe 2

Die Wartezimmersoftware (im folgenden Text abgekürzt als WZS) besteht aus 2 verschiedenen Programmkomponenten.

Die erste Komponente ist das Tool “WZS-Manager”, die Eingaben vom Arzt und der Anmeldung entgegennimmt und die Daten in eine CSV-Datei schreibt.
Die zweite Komponente ist das Tool “WZS-Viewer”, welche die vom Manager geschrieben Daten aus der CSV-Datei ausliest und in einem benutzerfreundlichen Format für das Wartezimmer darstellt.

Das CSV-Format wurde gewählt, da es Plaintext verwendet und damit im vergleich zu anderen Binärformaten leicht für Endnutzer zu bearbeiten ist. Sollte die Software also irgendwann abstürzen, ist eine Datenwiederherstellung einfach.
Die Daten, die in der CSV-Datei abgelegt werden, sehen in etwa folgendermaßen aus:

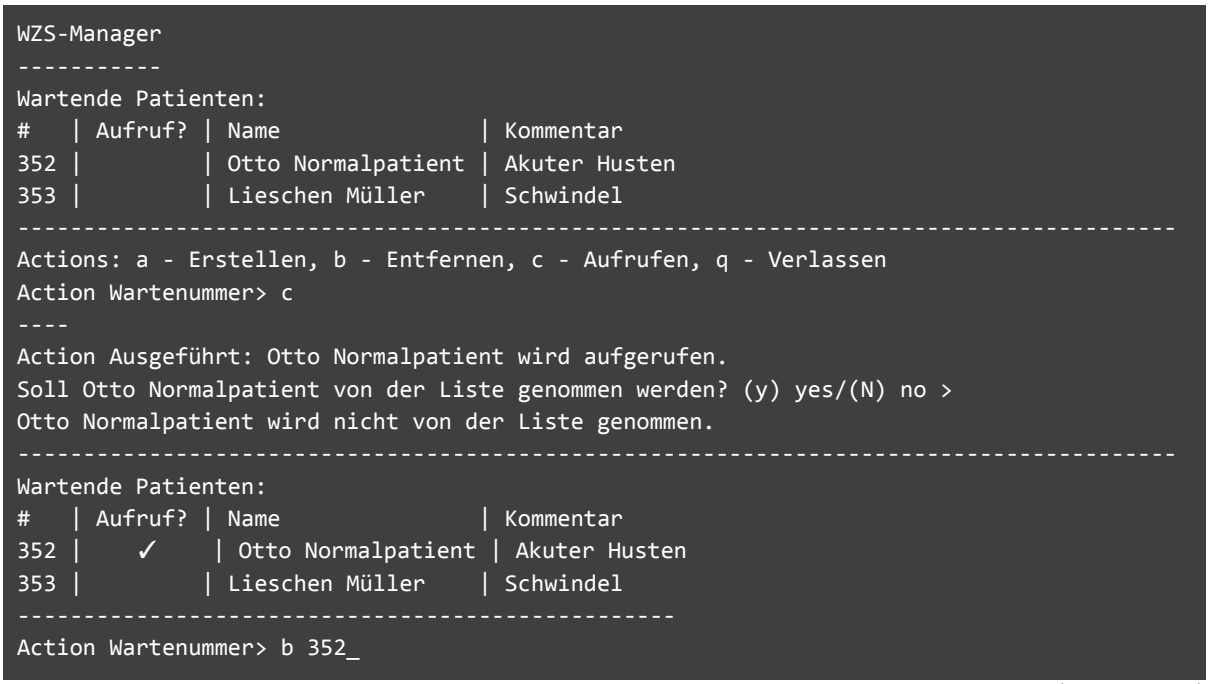
```
PatientName,Wartenummer,IsCalled,CommentB64,DataB64
Otto Normalpatient,352,True,YWt1dGVyIEh1c3Rlbg==,null
Lieschen Müller,353,False,RWFzdGVyZWdn,null
```

(Abbildung 1)

Hierbei ist PatientName der Name des Patienten, um diesen im WZS-Manager der Wartenummer zuzuordnen.
Wartenummer ist die Nummer, welche zum Anwählen von Einträgen im WZS-Manager und ebenfalls die Nummer, welche vom WZS-Viewer im Wartezimmer angezeigt wird.
Da die Anforderung N2 vorsieht, dass das Tool DSGVO-Konform arbeitet, ist es hier wichtig, dass das Name des Patienten im Wartezimmer nicht angezeigt wird.
Der Boolean IsCalled wird dann auf `true` gesetzt, wenn der Patient vom Arzt aufgerufen wird.
Im Feld CommentB64 wird ein Kommentar gespeichert, welcher mithilfe des WZS-Managers beim Erstellen einer Wartenummer gesetzt werden kann.
Da Eingabedaten in Textform manchmal Sonderzeichen enthalten, welche möglicherweise im CSV-Format ungültig sind, wird der Kommentar mit Base64 enkodiert, um eventuelle Schwierigkeiten beim Einlesen der Datei zu vermeiden.
DataB64 ist ein Feld, welches erlaubt zusätzliche Daten für einen Patient zu speichern, welche z.B. von anderen Softwarelösungen in der Arztpraxis verwendet werden.
Da Daten von fremder Software keinem genormten Standard folgen, sollten auch diese Daten mithilfe des Base64-Encodings gespeichert werden, um eventuelle Konflikte mit ungültigen Zeichen in der Datei zu vermeiden.

Die CSV-Datei wird in einem Netzwerkspeicherort hinterlegt, um die Datei über mehrere Computer zu synchronisieren.

Mithilfe des WZS-Managers werden die Einträge von der Anmeldung in die CSV-Datei eingefügt.
Die Wartenummer wird daraufhin vom WZS-Viewer im Wartezimmer angezeigt.
Der Arzt kann dann im Behandlungszimmer eine Wartenummer auswählen.
Im Manager wird nicht nur die Wartenummer angezeigt, sondern auch der Name des Patienten sowie der Kommentar, welcher von der Anmeldung eingegeben wurde als Plaintext.
Die Konsole sieht hierbei folgendermaßen aus:



(Abbildung 2)

Mithilfe des Managers können Arzt und Anmeldung in der Arztpraxis das komplette Anmeldesystem steuern.
Dies geschieht über “Actions”. Die verfügbaren Actions sind in der Abbildung 2 dargestellt.

Bei Action a (Erstellen) wird vom Manager eine Wartenummer generiert, der Name des Patienten im Format “Vorname Nachname”, sowie ein Kommentar mit Freitext abgefragt. Eventuelle ungültige Zeichen (zum Beispiel Kommata) werden aus den Eingaben entfernt. Anschließend wird die Wartenummer an das Ende der Liste angehangen.
Sollten weitere Daten für fremde Software in der CSV-Datei abgelegt werden, muss die Eintragung über ein Plugin für die fremde Software geschehen.

Bei Action b (Entfernen) wird die nach der Action eingegebene Wartenummer aus der CSV-Datei entfernt.
Wird keine Nummer eingegeben, wird die erste Wartenummer in der Liste entfernt.

Bei Action c (Aufrufen) wird der IsCalled Boolean des ersten Patienten auf true gesetzt, was im WZS-Viewer die Wartenummer hervorhebt.
Wird hinter der Action eine Wartenummer eingetragen, können Patienten auch außerhalb der Reihenfolge aufgerufen werden.
Es wird außerdem abgefragt, ob der Patient gleich aus der Liste genommen werden soll. Dies erlaubt dem Arzt, einen Patienten aufzurufen, auf dessen Ankunft zu warten und ihn dann sofort von der Liste zu nehmen, ohne die Wartenummer auch nur ein einziges mal zu tippen.
Die Anfrage wird standardmäßig mit Nein beantwortet, sodass der Arzt ein “y” eintragen muss, um den Eintrag zu entfernen.

Wird die Action q (Verlassen) ausgeführt, wird das WZS-Manager-Programm beendet.

Der WZS-Viewer kann in vielen verschiedenen Varianten entwickelt werden.
Dazu muss zuerst ein Programm entwickelt werden, welches auf Veränderungen der CSV-Datei wartet. Hier könnte in Golang zum Beispiel das Package fsnotify mit einem WRITE-Event genutzt werden.
Wird die CSV-Datei aktualisiert, wird die bisherige Ausgabe des Programmes geleert und mit den Inhalten der CSV-Datei aktualisiert.

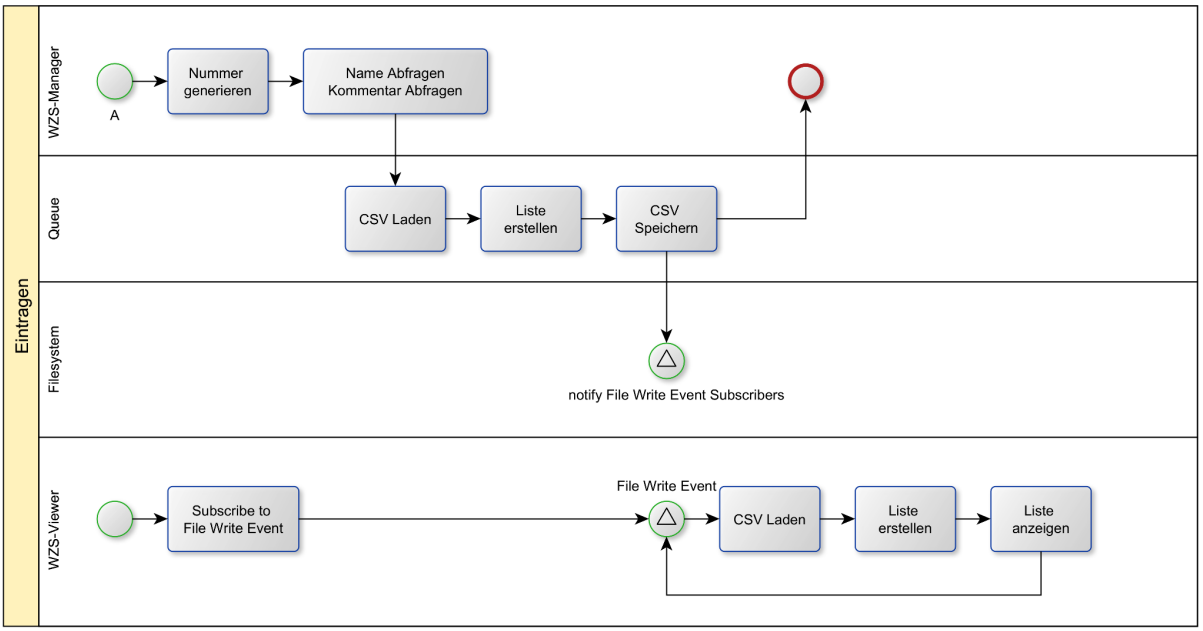
```
Wartezimmer
-----------
Aufgerufene Patienten:
352
-----------
Wartend:
353
```

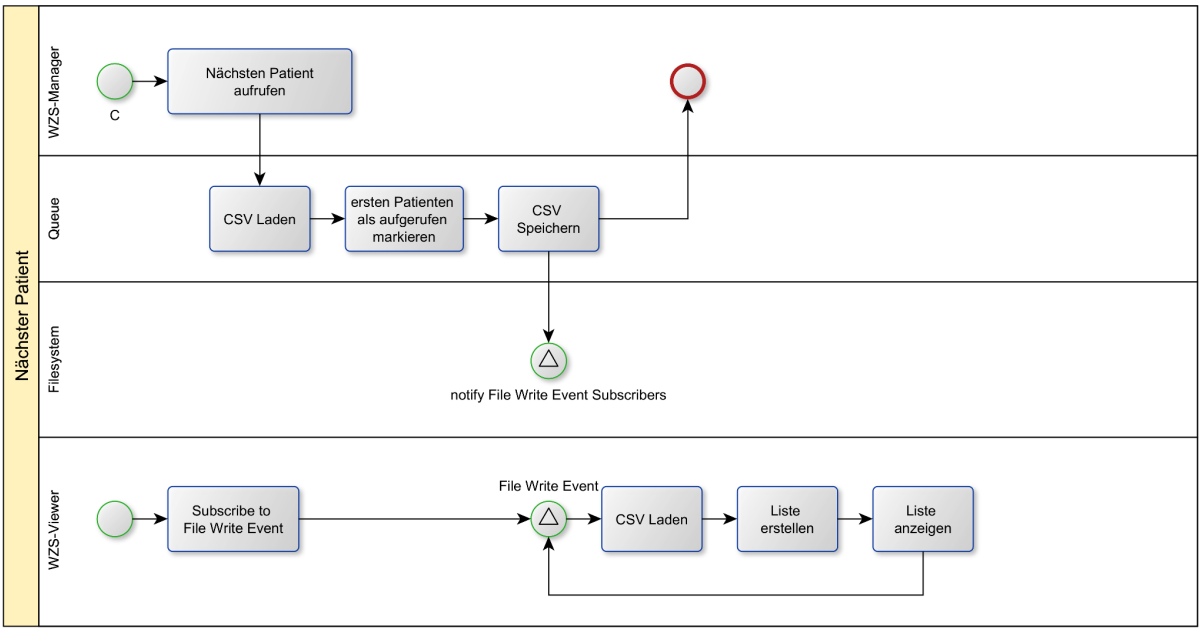
Aufgabe 3)

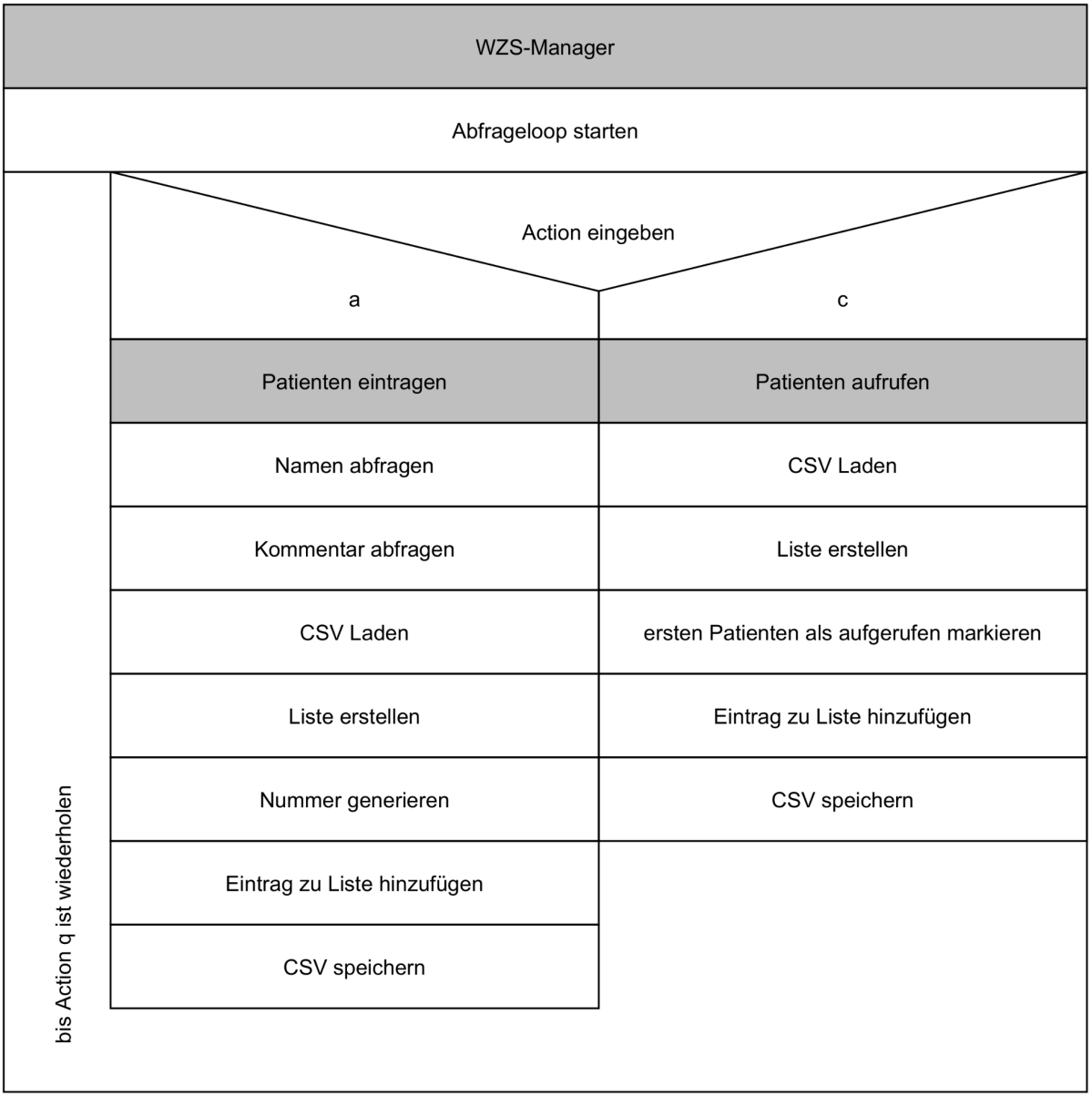
```
struct Record:
  name string
  number int
  comment string
  data string

from localproject load module QueueModule

global queue QueueModule.QueueType contains Record

function init:
  queue.init()

function eintragen ( name, comment ):
  record := new Record{name, queue.last.number + 1, comment, null}
  queue.enqueue(record)

function naechsterPatient ( number default null ) returns ( Record ):
  should number be null:
    return queue.popFirstElement()
  else:
    return queue.popElementAt(number)

should lang.runtimeMode not be “imported”:
  init()
```

QueueModule

```
struct Record:
  name string
  number int
  comment string
  data string

local queue lang.ListType contains Record

from lang load module CsvModule

function init:
  queue := lang.ListType.new()

public function enqueue( record ):
  rows := file_read()
  rows.append(record)
  file_write(rows)

public function popFirstElement() returns ( Record ):
  rows := file_read()
  first := rows[0]
  rows.deleteAtIndex(0)
  file_write(rows)
  return first

public function popElementAt(number) returns ( Record ):
  rows := file_read()
  for element, index in rows:
    should element.number be number:
      record := element
      rows.deleteAtIndex(index)
      break
  return record

public function last() returns ( Record ):
  rows := file_read()
  return rows[length_of(rows) - 1]

function file_read() returns (lang.ListType):
    rows := CsvModule.loadFile(
            lang.openFile(
              "\\\\AP-NAS\\WZS\\records.csv",
              “read”
            )
          ) as lang.ListType
    return rows

function file_write(list):
  file := lang.openFile(
            “\\\\AP-NAS\\WZS\\records.csv”,
            “overwrite”
          )
  file.write(CsvModule.dumpCsv(rows))
  file.flush()
  file.close()
  return first
```


Eintragen
WZS-Manager
Queue
Filesystem
WZS-Viewer
A
Nummer generieren
Name Abfragen Kommentar Abfragen
CSV Laden
Liste erstellen
CSV Speichern
notify File Write Event Subscribers
Subscribe to File Write Event
File Write Event
CSV Laden
Liste erstellen
Liste anzeigen


Nächster Patient
WZS-Manager
Queue
Filesystem
WZS-Viewer
C
Nächsten Patient aufrufen
CSV Laden
ersten Patienten als aufgerufen markieren
CSV Speichern
notify File Write Event Subscribers
Subscribe to File Write Event
File Write Event
CSV Laden
Liste erstellen
Liste anzeigen


WZS-Manager
Abfrageloop starten
Action eingeben
a
c
Patienten eintragen
Patienten aufrufen
Namen abfragen
CSV Laden
Kommentar abfragen
Liste erstellen
CSV Laden
ersten Patienten als aufgerufen markieren
Liste erstellen
Eintrag zu Liste hinzufügen
Nummer generieren
CSV speichern
Eintrag zu Liste hinzufügen
CSV speichern
bis Action q ist wiederholen