|  |
| --- |
|  |
| Bash – Getränkeautomat  Luca Fabian Burger  Neba dr Rüfi 14  **7203 Trimmis**  **076 306 72 03**  Photofuel.tech |

Ein Bild, das Spielautomat, Automat, Softdrink, Maschine enthält.

Automatisch generierte Beschreibung

Bild generiert von Microsoft Copilot

# Inhaltsverzeichnis

[Inhaltsverzeichnis 2](#_Toc167865371)

[Versionierung 4](#_Toc167865372)

[Kontaktdaten 4](#_Toc167865373)

[Planung 5](#_Toc167865374)

[Lastenheft 5](#_Toc167865375)

[1. Einleitung 5](#_Toc167865376)

[1.1 Projektbezeichnung 5](#_Toc167865377)

[1.2 Ziel des Projekts 5](#_Toc167865378)

[2. Projektbeschreibung 5](#_Toc167865379)

[2.1 Anforderungen an das System 5](#_Toc167865380)

[2.2 Funktionsweise des Skripts 5](#_Toc167865381)

[3. Technische Anforderungen 6](#_Toc167865382)

[3.2 Strukturierung 6](#_Toc167865383)

[3.3 Dokumentation 6](#_Toc167865384)

[4. Akzeptanzkriterien 6](#_Toc167865385)

[4.1 Funktionalität 6](#_Toc167865386)

[4.2 Benutzerfreundlichkeit 6](#_Toc167865387)

[4.3 Codequalität 6](#_Toc167865388)

[5. Einschränkungen 6](#_Toc167865389)

[5.1 Eingabemethoden 6](#_Toc167865390)

[5.2 Simulationsgrad 7](#_Toc167865391)

[6. Dealines 7](#_Toc167865392)

[6.2 Abgabedatum 7](#_Toc167865393)

[Pflichtenheft 7](#_Toc167865394)

[1. Einleitung 7](#_Toc167865395)

[2. Projektziele 7](#_Toc167865396)

[3. Anforderungen 7](#_Toc167865397)

[3.1. Funktionale Anforderungen 7](#_Toc167865398)

[3.2. Nicht-funktionale Anforderungen 7](#_Toc167865399)

[4. Systemübersicht 8](#_Toc167865400)

[5. Technische Details 8](#_Toc167865401)

[6. Betriebs- und Wartungsvorgaben 8](#_Toc167865402)

[7. Test und Abnahme 8](#_Toc167865403)

[8. Anhang 8](#_Toc167865404)

[Struktogramm 9](#_Toc167865405)

[Funktionsdefinition 10](#_Toc167865406)

[Funktionsbeschreib 11](#_Toc167865407)

[Beispielausgabe 12](#_Toc167865408)

[Debugging 13](#_Toc167865409)

[Fazit 14](#_Toc167865410)

[Anhang 15](#_Toc167865411)

[Code 15](#_Toc167865412)

[Readme.mnd 16](#_Toc167865413)

# Versionierung

|  |  |  |
| --- | --- | --- |
| Version | Anpassung | Datum |
| 0.1 | Initial Commit | 14.5.24 |
| 0.2 | Grundstruktur | 14.5.24 |
| 0.3 | Zahlprozesse | 14.5.24 |
| 0.4 | Switch Cases | 14.5.24 |
| 0.5 | Konsolen Ausgaben | 14.5.24 |
| 0.6 | Problembehebung | 14.5.24 |
| 0.7 | Input Funktionalität Update | 14.5.24 |
| 0.8 | README.md erstellt | 14.5.24 |
| 0.9 | Ordnerstruktur update | 14.5.24 |
| 1.0 | Erste Funktionierende Version | 14.5.24 |
| 1.1 | Auslagerungen auf Methoden | 21.5.24 |
| 1.2 | Aufsplitten von Switch Case | 21.5.24 |
| 1.3 | Input handling verbessert | 21.5.24 |
| 1.4 | Grammatik und Codestruktur angepasst | 21.5.24 |
| 1.5 | Ordnerstruktur Update | 21.5.24 |
| 1.6 | Hot-fixes | 28.5.24 |
| 1.7 | «Finale» Version vollendet | 28.5.24 |
| 1.8 | Dateistrukturen angepasst | 29.5.24 |
| 1.9 | Kommentare überarbeitet | 19.5.24 |

## Kontaktdaten

Name Luca Fabian Burger

Adresse Neba dr Rüfi 14, 7203 Trimmis

Telefonnummer 076 306 72 03

E-Mail-adresse info@photofuel.tech

# Planung

Zuerst habe mich darangesetzt, was ich überhaupt mit dem Projekt umsetzen will., wie ich vorgehen will, wie lange ich ca. dafür brauche und wann die Deadlines sind. Zuerst habe ich mich daran gesetzt eine ToDo-Liste zu kreieren. Anhand der ToDO-Liste habe ich mir den groben Ablauf des Programmes ausgearbeitet und habe das Struktogramm erstellt. Anschliessend habe ich das Programm fertig programmiert und mir dann 1 Woche für die Erstellung der Dokumentation gegeben.

## Lastenheft

## 1. Einleitung

### 1.1 Projektbezeichnung

Getränkeautomat

### 1.2 Ziel des Projekts

Ziel dieser Aufgabe ist es, ein automatisiertes Skript zu entwickeln, das einen Getränkeautomaten simuliert. Der Automat soll verschiedene Getränke anbieten und den gesamten Prozess der Getränkewahl, Bezahlung und Zubereitung simulieren.

## 2. Projektbeschreibung

### 2.1 Anforderungen an das System

Das Skript soll folgende Anforderungen erfüllen:

#### 2.1.1 Auswahl der Getränke

Der Benutzer soll die Möglichkeit haben, folgende Getränke auszuwählen:

* Kaffee
  + Latte Macchiato
  + Cappuccino
  + Espresso
  + Normaler Kaffee
* Tee
* Coca-Cola (Zero, Light, Normal)
* Sprite
* Fanta
* Mineralwasser (mit Kohlensäure/ohne Kohlensäure)

#### 2.1.2 Zusatzauswahl bei Heissgetränken

Bei Heissgetränken (Kaffee und Tee) soll zusätzlich abgefragt werden:

* Mit Zucker / Ohne Zucker
* Mit Milch / Ohne Milch

#### 2.1.3 Bezahlung

Bevor das Getränk erstellt werden kann, muss das nötige Geld eingeworfen werden. Der Automat soll keine Aktion ausführen, bevor der Zahlungsprozess abgeschlossen ist.

### 2.2 Funktionsweise des Skripts

Das Skript muss funktionsorientiert gestaltet sein.

Die richtigen Kontrollstrukturen (wie Schleifen und Verzweigungen) müssen verwendet werden, um den Ablauf zu steuern.

Benutzereingaben müssen in Variablen oder Arrays gespeichert werden.

## 3. Technische Anforderungen

3.1 Programmiersprache

Das Skript soll in [bash] entwickelt werden.

### 3.2 Strukturierung

Vor der Programmierung muss ein Struktogramm oder Programmablaufplan erstellt werden, um die Logik und den Ablauf des Skripts zu visualisieren.

### 3.3 Dokumentation

Das abgegebene Dokument muss folgende Bestandteile enthalten:

* Titelblatt (mit Kopf- und Fusszeile)
* Inhaltsverzeichnis
* Versionierung
* Kontaktdaten
* Lastenheft
* Pflichtenheft
* Struktogramm/Programmablaufplan
* Code
* Testing
* Fazit

## 4. Akzeptanzkriterien

### 4.1 Funktionalität

* Der Benutzer kann alle aufgeführten Getränke auswählen und die erforderlichen Zusatzauswahlen treffen.
* Der Automat akzeptiert nur gültige Zahlungen und verweigert die Ausgabe des Getränks bei unzureichender Zahlung.
* Die Zubereitung des Getränks wird simuliert.

### 4.2 Benutzerfreundlichkeit

* Klare und verständliche Abfragen an den Benutzer.
* Eindeutige Rückmeldungen zu den Benutzereingaben und zum Status des Automaten.

### 4.3 Codequalität

* Sauberer und kommentierter Code.
* Verwendung von sinnvollen Variablen- und Funktionsnamen.
* Modularisierung durch funktionsorientiertes Design.

## 5. Einschränkungen

### 5.1 Eingabemethoden

Das Skript wird nur über die Kommandozeile bedient.

### 5.2 Simulationsgrad

Die Ausgabe der Zubereitung erfolgt nur als Textsimulation und nicht als tatsächliche Herstellung eines Getränks.

## 6. Dealines

### 6.2 Abgabedatum

Das fertige Dokument und das Skript müssen bis 31.05.2024 eingereicht werden.

## Pflichtenheft

## 1. Einleitung

Dieses Pflichtenheft beschreibt die Umsetzung der Anforderungen für das Projekt "Getränkeautomat". Das Ziel ist die Bereitstellung eines Automaten, der verschiedene Getränke und Zigaretten verkauft und dabei Nutzerinteraktionen sowie Zahlungsvorgänge effizient abwickelt.

## 2. Projektziele

* Bereitstellung einer benutzerfreundlichen Schnittstelle für die Auswahl und den Kauf von Getränken und Zigaretten.
* Sicherstellung zuverlässiger Zahlungsabwicklungen.
* Integration von Anpassungsoptionen für bestimmte Getränke (z.B. Auswahl von Milch- und Süssungsoptionen für Kaffee und Tee).
* Aufzeichnung und Anzeige der täglichen Verkäufe.

## 3. Anforderungen

### 3.1. Funktionale Anforderungen

* Starten des Automaten: Der Automat muss vom Benutzer gestartet werden können.
* Getränke- und Zigarettenauswahl: Der Benutzer kann aus einer Liste von Getränken und Zigaretten auswählen.
* Anpassungsoptionen für Getränke: Der Benutzer kann Milch- und Süssungsoptionen für Kaffee und Tee auswählen.
* Zahlungsabwicklung: Der Automat akzeptiert Münzen und Scheine und gibt Wechselgeld zurück.
* Tägliche Verkaufsaufzeichnung: Der Automat speichert und zeigt die täglichen Verkäufe an.
* Apple-Fans Liste: Der Automat fragt den Benutzer, ob er ein Apple-Fan ist, und fügt den Namen zur "Apple-Fans"-Liste hinzu, falls gewünscht.

### 3.2. Nicht-funktionale Anforderungen

* Benutzerfreundlichkeit: Die Benutzeroberfläche soll intuitiv und einfach zu bedienen sein.
* Zuverlässigkeit: Der Automat soll zuverlässig funktionieren und Transaktionen sicher abwickeln.
* Performance: Der Automat soll innerhalb von 2 Sekunden auf Benutzereingaben reagieren.
* Sicherheit: Der Automat muss sicherstellen, dass nur gültige Zahlungen akzeptiert werden und persönliche Daten geschützt sind.

## 4. Systemübersicht

Der Getränkeautomat besteht aus den folgenden Hauptkomponenten:

* Benutzerschnittstelle: Ermöglicht die Auswahl von Produkten und die Eingabe von Anpassungsoptionen.
* Zahlungssystem: Akzeptiert Münzen und Scheine, berechnet den Gesamtpreis und gibt Wechselgeld zurück.
* Verwaltungssystem: Speichert tägliche Verkäufe und führt die "Apple-Fans"-Liste.

## 5. Technische Details

Das Projekt wurde unter Linux (Arch) entwickelt und läuft als Bash-Skript. Das Projekt ist 100% in der Shell programmiert. Der Getränkeautomat wird über eine andere Datei gestartet. Die Start-bash-datei lädt den Code und die Dokumentation aus meinem GitHub Account runter und startet den Getränkeautomaten. Voraussetzungen für das Programm sind das es eine .sh ausführen kann und, dass das System ein Verzeichnis Desktop/ besitzt

## 6. Betriebs- und Wartungsvorgaben

* Betriebssystem: Linux (Arch)
* Starten des Programms:
  + Zum Starten des Automaten: `./getraenke-automat.sh`
  + Zum Starten aus dem Root-Verzeichnis: `./startGetraenke.sh`
* Wartung: Regelmässige Überprüfung und Aktualisierung des Skripts zur Behebung von Fehlern und zur Implementierung neuer Funktionen.

## 7. Test und Abnahme

* Testfälle:
  + Starten des Automaten
  + Auswahl und Anpassung von Getränken
  + Zahlungsvorgänge und Wechselgeldrückgabe
  + Speicherung und Anzeige der täglichen Verkäufe
  + Hinzufügen von Benutzern zur "Apple-Fans"-Liste
* Abnahme: Die Abnahme erfolgt durch den Auftraggeber nach erfolgreichem Abschluss der definierten Testfälle. Alle Funktionen müssen fehlerfrei arbeiten und die Anforderungen des Lastenhefts erfüllen.

## 8. Anhang

* Autor: Luca Fabian Burger
* Version: 1.9
* Letzte Änderung: Verbesserung der Dateistruktur am 29.05.2024

## Ein Bild, das Text, Diagramm, parallel, technische Zeichnung enthält. Automatisch generierte BeschreibungStruktogramm

# Funktionenliste

Liste der Funktionen im Bash-Skript für den Getränkeautomaten:

1. start\_machine()

Startet den Automaten, bis der Benutzer "1" eingibt.

2. get\_beverage()

Zeigt das Getränke- und Zigarettenmenü zur Auswahl an.

3. get\_cafe()

Zeigt das Kaffeemenu zur Auswahl an.

4. get\_tea()

Zeigt das Teemenu zur Auswahl an.

5. get\_cigarettes()

Zeigt das Zigarettenmenu zur Auswahl an.

6. get\_mineral\_water()

Ermöglicht die Auswahl von Mineralwasser mit oder ohne Kohlensäure.

7. get\_cola()

Zeigt das Cola-Menu zur Auswahl an.

8. get\_y\_n()

Fragt den Benutzer nach einer Ja- oder Nein-Antwort.

9. get\_milk()

Ermöglicht die Auswahl einer Milchart oder keine Milch.

10. get\_sugar()

Ermöglicht die Auswahl einer Süssungsoption oder keine Süssung.

11. get\_modifications()

Ruft get\_milk() und get\_sugar() auf, um Getränkemodifikationen zu erhalten.

12. select\_cafe()

Verarbeitet die Kaffeeauswahl und setzt Getränk und Preis.

13. select\_tea()

Verarbeitet die Teeauswahl und setzt Getränk und Preis.

14. select\_cigarettes()

Verarbeitet die Zigarettenauswahl und setzt Marke und Preis.

15. handle\_payment()

Verarbeitet die Zahlung und berechnet das Wechselgeld.

16. add\_to\_apple\_fans()

Fügt den Benutzer zur "Apple-Fans"-Liste hinzu.

17. display\_order\_message()

Zeigt eine Zusammenfassung der Bestellung an.

18. simulate\_loading\_bar()

Simuliert eine Ladeanzeige während der Zubereitung.

19. simulate\_shutdown\_loading\_bar()

Simuliert eine Ladeanzeige während des Herunterfahrens.

20. (main)

Das main kontrolliert den Code, hier wird der Ablauf des Programms definiert und die «functions» aufgerufen. Der Code selbst werde ich nicht genau Linie für Linie erklären der Code ist mit Kommentaren versehen und sollte verständlich geschrieben sein.

# Beispielausgabe

Ein Bild, das Text, Screenshot enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Display, Software enthält.

Automatisch generierte Beschreibung

# Debugging

Hier habe ich meine Testfälle aufgelistet. Es gab viel mehr Tests, ob man das Programm brechen kann. An dieser Stelle möchte ich Patrick Weidmann danken welcher mir geholfen hat die Schwächen in meinem Code zu finden und auszumerzen. Ohne ihn könnte man das Programm mit leeren Strings bis jetzt noch zum Überspringen von Code-segmenten oder dem Abbruch des Programms zu erzwingen. Ein kurzes Beispiel, welches kein eigener Testfall bekam, wenn man die Integer value gebrochen hat das System horrende Summen an Rückgeld gegeben, dies wurde behoben und ist nicht mehr. Aber es gibt keinen spezifischen Testfall dazu.

Die Testfälle sind wie folgt aufgebaut:

* Testfall
  + Testfall:
  + Geplante Ausgabe:
  + Ausgabe:
  + Ist die Ausgabe wie geplant:

Disclaimer: Die Testfälle stehen stellvertretend für alle Test welche durchgeführt wurden und dienen nicht dazu alle aufzulisten!

* Testfall
  + Kann der Automat mit 1 gestartet werden
  + Startscreen der Auswahl
  + Startscreen der Auswahl
  + Ja, alles ist wie geplant
* Testfall
  + Kann der Automat mit -1 beendet werden
  + Daily\_sales werden angezeigt, danach wird das System runtergefahren
  + Daily\_sales werden angezeigt, danach wird das System runtergefahren, anschliessend wird der Start screen der Maschine wieder gestartet
  + Ja, sogar besser als ursprünglich geplant
* Testfall
  + Wähle "Cafe"
    - Wähle "Espresso"
    - Wähle keine Milch und keinen Zucker
    - Bezahle den Betrag korrekt
  + Anzeige des Getränke-Menüs
    - Anzeige des Cafe-Menüs
    - Auswahl der Modifikationen (Milch und Zucker)
    - Aufforderung zur Zahlung mit korrekter Bestätigung
    - Bestätigung der Bestellung und Zubereitungsstatus
  + Getränke-Menü wird angezeigt
    - Cafe-Menü wird angezeigt
    - Modifikationen werden abgefragt (keine Milch, kein Zucker)
    - Zahlung wird korrekt bestätigt
    - Bestellung wird bestätigt und zubereitet
  + Ja, alles ist wie geplant

# Fazit

Das Projekt war für mich eine sehr lehrreiche und interessante Erfahrung. Es hat mir ermöglicht, mein Wissen und Verständnis von Bash deutlich zu erweitern. Während der Entwicklung des Projekts habe ich viele praktische Kenntnisse erworben, die weit über die theoretischen Grundlagen hinausgehen. Einige der fortgeschritteneren Funktionen sind mir zwar immer noch etwas schwammig, aber insgesamt habe ich durch dieses Projekt ein solides Verständnis von Bash erlangt.

Die Umsetzung meines Projekts ist für mich nicht perfekt, dennoch bin ich stolz auf das Ergebnis. Natürlich gibt es Verbesserungsmöglichkeiten, besonders was die Anzahl der Codezeilen und die Struktur der Funktionen betrifft. Ich neige dazu, mehr Funktionen zu schreiben als nötig, weil ich es bevorzuge, den Code so zu gestalten, dass er gut strukturiert und leicht lesbar ist. Meiner Meinung nach ist dies ein wichtiger Aspekt der Softwareentwicklung, der die Wartbarkeit und Erweiterbarkeit des Codes verbessert.

Die Dokumentation des Projekts war für mich der schwierigste Teil. Es fiel mir schwer, meine Gedanken und die Funktionsweise des Codes klar und präzise zu dokumentieren. Um sicherzustellen, dass die Dokumentation fehlerfrei und verständlich ist, habe ich sie mehrfach von ChatGPT, Llama 3 und DeepL auf Rechtschreibfehler und Satzstellung überprüfen lassen. Trotz dieser Unterstützung bin ich mir bewusst, dass die Dokumentation der schwächste Teil meiner Arbeit ist.

Insgesamt bin ich jedoch zufrieden mit dem, was ich erreicht habe. Das Projekt hat mir nicht nur technisches Wissen vermittelt, sondern auch meine Fähigkeiten im Projektmanagement und in der Problemlösung verbessert. Es hat mir gezeigt, wie wichtig es ist, kontinuierlich zu lernen und offen für neue Technologien und Methoden zu sein. Auch wenn das Projekt seine Herausforderungen hatte, betrachte ich es als wichtigen Schritt in meiner Entwicklung als Programmierer.

# Anhang

## Code - Getränkeautomat



## Code – Start Getränkeautomat



## Readme.mnd

