Tatyana Merlo – Natalie Stalder – Nadja Stadelmann

Starbucks Manager

M120 – Benutzerschnittstelle entwickeln

# Einleitung

In einem vorherigen Projekt haben wir für Starbucks ein Menümanager geschrieben. Die Mitarbeiter können Produkte zum Menu hinzufügen, bearbeiten und wieder löschen. Weiter wird das Menü natürlich auch für die Kunden angezeigt.

Bisher wurde kein GUI eingesetzt, weshalb für die Klienten von Starbucks ein eher unschönes Menü angezeigt wurde. Weiter konnte das Programm nur über die Konsole bearbeitet werden, sodass eine spezielle Mitarbeiterschulung für den Gebrauch des Starbucks Managers nötig war.

Dies wollen wir in diesem Folgeprojekt nun ändern.

## Dokument

### Aufbau

* Analyse
  + Beschrieb und erste Aufsplittung in einzelne Teilbereiche
* Vorgehen
  + Abhängigkeiten, Aufgabenübersicht
* Design
  + Detaillierte Analyse DB-Design
  + Detaillierte Analyse Programmstruktur
* Implementation
* Testing
  + Testcases basieren auf UseCases

Inhalt

[Einleitung 1](#_Toc497240867)

[Dokument 1](#_Toc497240868)

[Aufbau 1](#_Toc497240869)

[Vision 4](#_Toc497240870)

[Analyse 4](#_Toc497240871)

[Anforderungen 4](#_Toc497240872)

[Muss-Kriterien 4](#_Toc497240873)

[Kann-Kriterien 5](#_Toc497240874)

[Use Cases 5](#_Toc497240875)

[Story-Bord 7](#_Toc497240876)

[Design 9](#_Toc497240877)

[Filestruktur 9](#_Toc497240878)

[Programm 9](#_Toc497240879)

[Model 9](#_Toc497240880)

[View 9](#_Toc497240881)

[Controller 10](#_Toc497240882)

[Klassendiagramm 10](#_Toc497240883)

[GUI 10](#_Toc497240884)

[Interaktion mit Model 10](#_Toc497240885)

[Ergonomie 11](#_Toc497240886)

[Klärung der Ebenen 11](#_Toc497240887)

[Aufgaben-Ebene 11](#_Toc497240888)

[Semantische Ebene 11](#_Toc497240889)

[Syntaktische Ebene 11](#_Toc497240890)

[Interaktionsebene 11](#_Toc497240891)

[Benutzer-Arten 12](#_Toc497240892)

[Weitere Aspekte 12](#_Toc497240893)

[Aufbau 12](#_Toc497240894)

[Dialoge 12](#_Toc497240895)

[Grundsätze der Dialoggestaltung 13](#_Toc497240896)

[Bedienung Funktionsorientiert / Objektorientiert 13](#_Toc497240897)

[Menü 13](#_Toc497240898)

[Styleguides 14](#_Toc497240899)

[Implementation 15](#_Toc497240900)

[Implementierung 15](#_Toc497240901)

[Verantwortlichkeiten 15](#_Toc497240902)

[Technischer Aufbau 15](#_Toc497240903)

[GUI 15](#_Toc497240904)

[DataHolder 15](#_Toc497240905)

[Verbindung zum Model 15](#_Toc497240906)

[Entwicklungsprozess 15](#_Toc497240907)

[Testing 17](#_Toc497240908)

[Testfallmatrix 17](#_Toc497240909)

[Anhang 22](#_Toc497240910)

[Projektauftrag – WebApplikation realisieren 22](#_Toc497240911)

# Vision

Im Modul 326 haben wir ein Starbucks-Menümanager mit Java erstellt. Bisher konnte der jedoch nur textbasiert verwendet werden (mittels Konsole).

Unser Ziel ist nun, den Mitarbeitern das Mutieren des Menüs mit einem GUI zu erleichtern. Das GUI soll alle bisherigen Userinteraktionen abdecken, sprich:

* Menü anzeigen
* Produkt hinzufügen
* Produkt bearbeiten
* Produkt löschen
* Programm beenden

Weiter möchten wir in einem weiteren Schritt eine Login- sowie Logoutfunktion einbauen, um die Sicherheit der Daten zu gewährleisten.

# Analyse

## Anforderungen

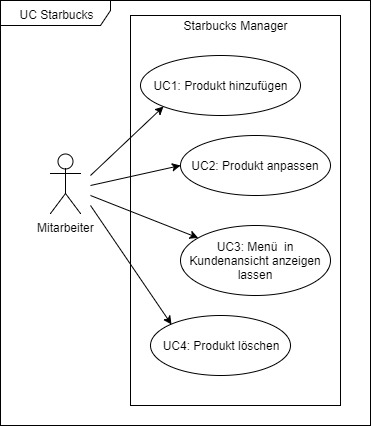
### Muss-Kriterien

1. Menüanzeige für die Kunden
   1. an Starbucks-Design angepasst
   2. gegliedert nach Produktkategorien
2. Menüanzeige für den Mitarbeiter
   1. Edit-Funktion zu jedem Produkt
   2. Delete-Funktion zu jedem Produkt
   3. selbe Gliederung wie bei der Kundenansicht
   4. Überbegriffe der einzelnen Felder sichtbar (bspw. Price)
3. Add-Funktion
   1. eigene Ansicht
   2. Auswahl der Produktkategorie
   3. Eingabe der Produktinformationen
      1. nur die relevanten Infos müssen eingegeben werden
4. Edit-Funktion
   1. Item bearbeiten
5. Delete-Funktion
   1. Item löschen
6. Layout
   1. Einheitlich
   2. Schlicht
   3. Gemäss Kapitel sowieso
7. Error-Handling
   1. genaue Informationen für den User

### Kann-Kriterien

1. Preis-Eingabe
   1. Dialogfenster mit Taschenrechnerlayout
2. Add-Funktion
   1. Speicherung relevante Daten beim Wechsel der Produktkategorie
3. Edit-Funktion
   1. Produktkategorie anpassen können
4. Help-Funktion
   1. Beschrieb der Funktionen
   2. Beschrieb des Files product.txt
5. Error-Handling
   1. Dialogfenster
      1. Message
      2. OK-Button als Userbestätigung

## Use Cases



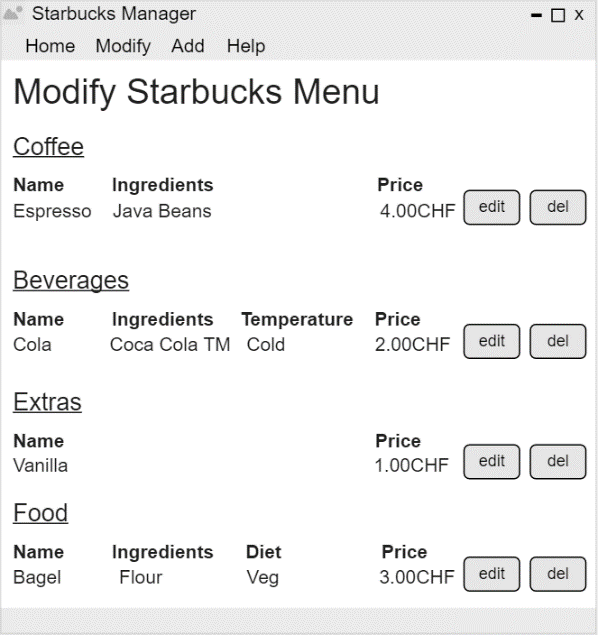
|  |  |
| --- | --- |
| **Identifikation** | **UC1: Produkt hinzufügen** |
| Beschreibung | Ein Produkt wird hinzugefügt. |
| Primärer Actor | Mitarbeiter |
| Vorbedingungen | - |
| Trigger | Mitarbeiter wählt Menü „Add“ |
| Standardablauf | 1. Wählt Menü „Add“  2. Wählt Produktkategorie  3. Gibt Produktdaten ein  4. Produkt wird hinzugefügt |
| Alternative Abläufe | 2a. falsche Produktkategorie gewählt 🡪 User kann Kategorie anpassen ohne Datenverlust  3a. Produkt bereits vorhanden 🡪 Fehlermeldung ausgeben  3b. Falsche Zeichen eingegeben 🡪 Fehlermeldung ausgeben |

|  |  |
| --- | --- |
| **Identifikation** | **UC2: Produkt anpassen** |
| Beschreibung | Ein bestehendes Produkt wird editiert |
| Primärer Actor | Mitarbeiter |
| Vorbedingungen | Produkt vorhanden  Menü „Modify“ ausgewählt |
| Trigger | Mitarbeiter wählt Option „edit“ |
| Standardablauf | 1. Wählt Option „edit“ beim zu editierenden Produkt  2. Produktinformationen werden in einem neuen Fenster in editierbaren Feldern angezeigt  3. Der User ändert die Produkteinformationen  4 User bestätigt seine Änderungen mit Klick auf "edit"  5. Das Produkt wird angepasst  6. Das Edit-Fenster wird geschlossen  7. Eine Erfolgsmeldung wird ausgegeben  8. Die Modify-Scene wird neu geladen und angezeigt |
| Alternative Abläufe | 3a. Falsche Zeichen eingegeben 🡪 Fehlermeldung ausgeben  4a. Der User schliesst das Fenster ohne Klick auf "edit" 🡪 das Produkt bleibt im alten Zustand  4a. Das Produkt kann nicht editiert werden 🡪 Das Fenster bleibt offen und eine Fehlermeldung wird ausgegeben |

|  |  |
| --- | --- |
| **Identifikation** | **UC3: Menü in Kundensicht anzeigen** |
| Beschreibung | Das Menü wird strukturiert und im Starbucks-Design ausgegeben |
| Primärer Actor | Mitarbeiter |
| Vorbedingungen | - |
| Trigger | Mitarbeiter wählt Menü „Home“ / Programmstart |
| Standardablauf | 1. Wählt Menü „Home“ / startet Programm  2. Produkte werden nach Typ sortiert ausgegeben |
| Alternative Abläufe | - |

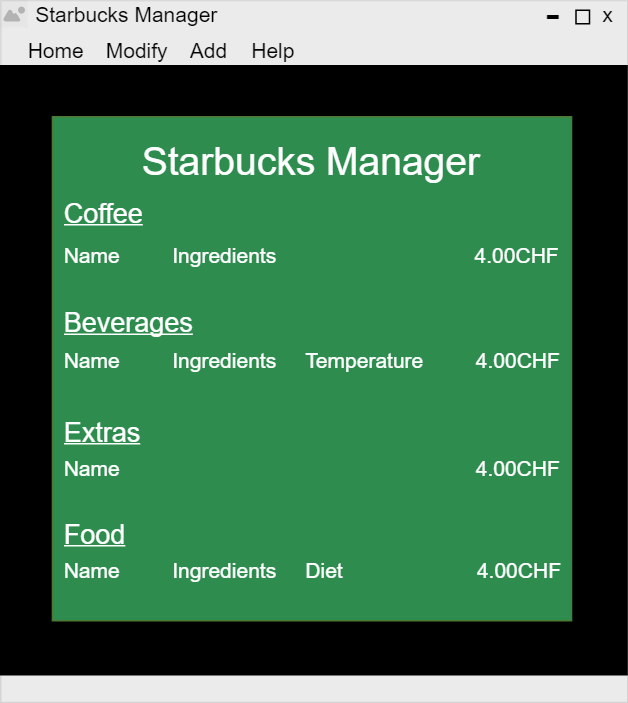
|  |  |
| --- | --- |
| **Identifikation** | **UC4: Produkt löschen** |
| Beschreibung | Ein bestehendes Produkt wird entfernt |
| Primärer Actor | Mitarbeiter |
| Vorbedingungen | Produkt vorhanden  Menü „Modify“ ausgewählt |
| Trigger | Mitarbeiter wählt Option „delete“ |
| Standardablauf | 1. Wählt Option „delete“ bei zu löschendem Produkt  2. Produkt wird entfernt  3. Eine Erfolgsmeldung wird ausgegeben  3. Die Modify-Scene wird neu geladen und angezeigt |
| Alternative Abläufe | 2a. Das Produkt kann nicht entfernt werden 🡪 eine Fehlermeldung wird ausgegeben |

## Story-Bord



Menü Modify

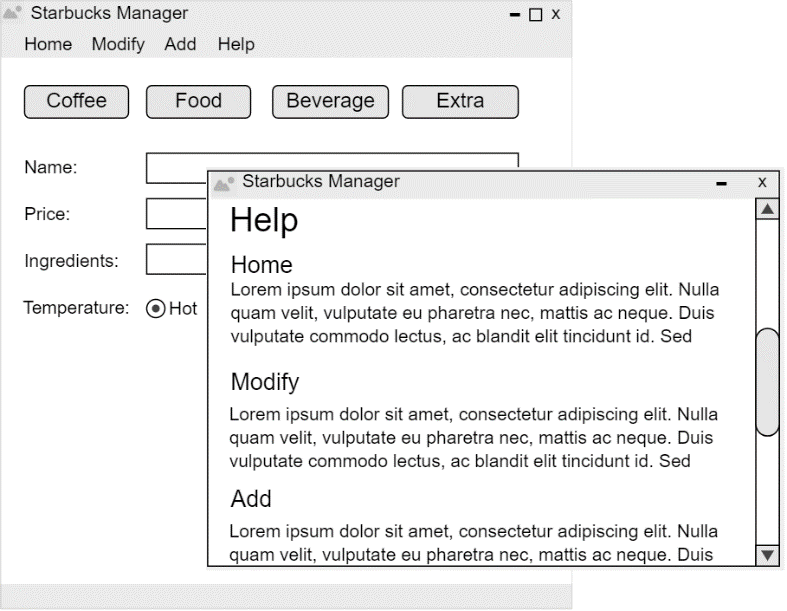
Klick



Programmstart

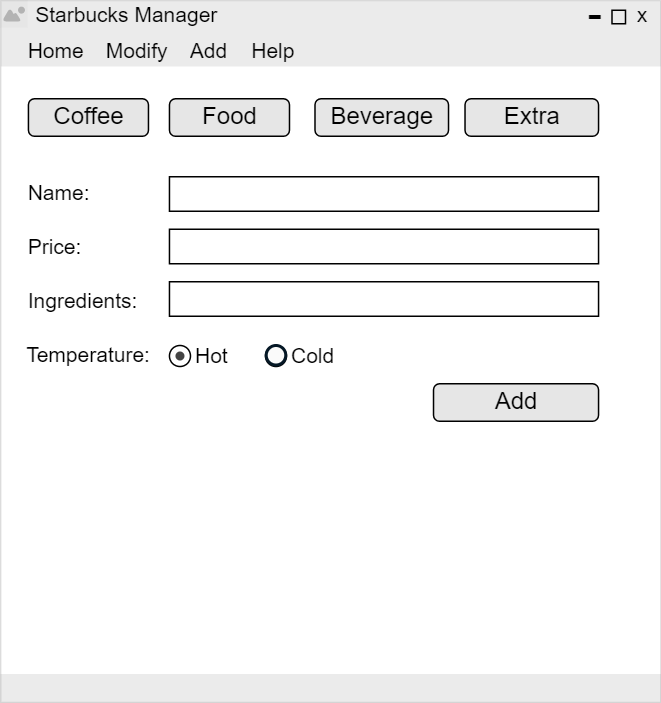
Menü Home

Klick / Start



Menü Help

Klick



Menü Add

Klick

Programmende

Home

**Eingaben** a - / Menü ‘Home’

**Ausgaben** a Menüanzeige aus Kundensicht

**Dialoge** -

Modify

**Eingaben** a Menü ‘Modify’  
b Button ‘edit’  
c Button ‘delete’

**Ausgaben** a Menüanzeige aus Mitarbeitersicht  
b Toast ‘Edit was successful’  
c Toast ‘Delete was successful’

**Dialoge** b Edit

Add

**Eingaben** a Menü ‘Add’  
b Textfelder befüllen mit Produktdaten  
c Button ‘add Item’

**Ausgaben** b Textfelder mit Labels zum Befüllen  
c Toast ‘Add was successful’

**Dialoge** b Price  
c Error bei Falsch- / fehlenden Eeingaben

Help

**Eingaben** a Menü ‘Help’

**Ausgaben** a Anzeige Help-Text  
b Nicht-modaler Dialog

**Dialoge** -

Edit

**Eingaben** a Textfelder befüllen mit Produktdaten  
b Button ‘edit’

**Ausgaben** a Textfelder mit Labels zum Befüllen  
b Modaler Dialog  
c Toast ‘Edit was successful’

**Dialoge** a Price  
c Error bei Falsch- / fehlenden Eingaben

Price

**Eingaben** a Buttons zur Zahleneingabe

**Ausgaben** a Textfeld mit der eingegebenen Zahl  
b Modaler Dialog

**Dialoge** a Error bei Falsch- / fehlenden Eingaben

Error

**Eingaben** a Button ‘ok’

**Ausgaben** a Button zur Bestätigung – danach wird Dialog   
geschlossen  
b Fehlermeldung als modaler Dialog

**Dialoge** -

Toasts

**Eingaben** -

**Ausgaben** a Toastmessage nach erfolgreicher Aktion im  
 Hintergrund

**Dialoge** -

# Design

## Filestruktur

Der bisherige Starbucks Manager hat keine Datenbank, welche er verwaltet. Die Daten werden in einem einfachen File abgespeichert. Dennoch ist die Struktur des Files natürlich extrem wichtig für die Datenverarbeitung, weshalb sie hier kurz deklariert ist.

Jede Zeile steht für einen Menü-Eintrag, der in einzelne Komponenten unterteilt ist. Je nachdem, welche Attribute der Eintrag mitführt, wird er in eine andere Kategorie unterteilt.

**Kategorie Recordstruktur**

Beverage Name | Preis | Zutaten | true (hot) / false (cold)  
beverage¦2.3¦Zutaten¦true

Coffee Name | Preis |Zutaten  
coffee¦1.4¦ingr

Extra Name | Preis  
extra¦2.0

Food Name | Preis |Zutaten | Vegi / Vegan / Glutenfrei / usw.  
food¦1.5¦Ingr¦Vegi

## Programm

### Model

Wie bereits erwähnt werden die Daten in einem einzigen File gehandhabt. Das File und sein Handling besteht bereits. Dennoch müssen am Datenmodel noch einiges angepasst werden, um die korrekte Zusammenarbeit mit dem GUI sicherzustellen.

### View

Das GUI ist komplett neu umzusetzen. Bisher waren nur Kommandozeileneingaben zur Steuerung des Programmes da. Dies soll neu über ein GUI geregelt werden, um den Mitarbeitern einen leichteren Einstieg zu ermöglichen.

So stellen wir uns das Ergebnis des GUIs vor:

Die Kundenansicht soll dem Starbucks-Design angepasst sein. In ihrem Logo ist ein spezielles Grün zu finden. Weiter zeigt es eine Weisse Figur auf schwarzem Hintergrund. Um den Kunden eine möglichst angenehme Ansicht bieten zu können, wird für das GUI das Grün als Hintergrund eingesetzt. Darauf positionieren werden die Artikel in weisser, serifenloser Schrift. Diese garantiert einen guten Lesefluss. Um dem Kunden die Auswahl möglichst zu vereinfachen, werden die Produkte in vier Kategorien unterteilt. Die Kaffees sind zu Oberst zu finden, da die meisten Kunden Starbucks wegen diesen Produkten besuchen. Danach folgen die Beverages (weitere Getränke), die Extras und zum Schluss noch die Muffins und Cookies unter der Kategorie Food.

Für die Mitarbeiteransicht wollen wir ein einheitliches Layout erstellen, damit der User sich schnell auf allen Seiten zurechtfinden. Es soll ausserdem übersichtlich sein, das heisst für uns, dass der User immer nur das angezeigt bekommt, was er wirklich braucht. Als Beispiel ist hier die Funktion ‘Produkt hinzufügen’ super geeignet. Ein Kaffee oder ein Extra benötigen weniger Angaben als ein neues Produkt der Kategorie Food. Also sollen auch nur diese Angaben abgefragt werden, welche notwendig sind.

Für all diese Views verwenden wir CSS für JavaFX, um die verschiedenen Teile des GUIs einheitlich gestalten zu können.

### Controller

Die Controller bestehen teilweise, es wurden bereits Usereingaben verarbeitet. Dennoch müssen sie an das GUI angepasst werden, da das Programm bisher nur über die Konsole bedient werden konnte. Teils müssen sie ganz ausgewechselt werden.

Einige Überprüfungen der Usereingaben werden neu vom GUI bereits abgefangen. So wird die Businesslogik entlastet, da nicht mehr alle Eingaben auf ihre Richtigkeit überprüft werden müssen. Ein Preis, welcher vom GUI weitergeliefert wird, ist beispielsweise immer ein double wenn er im Controllerbereich ankommt.

## Klassendiagramm

### GUI

### Interaktion mit Model

# Ergonomie

Da sich dieses Modul vor allem um die GUI-Implementierung und das Design des GUIs dreht, gehen wir in diesem Kapitel genauer auf unsere Überlegungen dazu ein.

## Klärung der Ebenen

Als erstes müssen einige Aspekte geklärt werden, um genau eruieren zu können, was das GUI alles können muss. Dies geschieht mittels verschiedenen Ebenen, welche hier genauer erläutert und auf unser GUI bezogen bearbeitet werden.

### Aufgaben-Ebene

Als erstes stellen wir uns die Frage, welches nun die primären Aufgaben des Benutzers sind. Nur wenn wir wissen, für was der Benutzer das Programm im Endeffekt benützen wird, können wir es auch entsprechend gestalten. In unserem Fall sind dies folgende Aufgabe:

* Starbucks-Menü kundenfreundlich und aktuell anzeigen lassen

Um diese Endaufgabe erreichen zu können, muss der Benutzer auch folgende Aufgaben bewältigen können:

* Produkte hinzufügen
* Produkte bearbeiten
* Produkte löschen

### Semantische Ebene

Weiter ist es wichtig zu wissen, welche Werkzeuge dem Benutzer bereits zur Verfügung stehen und was noch bereitgestellt werden muss.

In unserem Fall besteht bereits das Model und die Businesslogik vom Programm, welches auf einem handelsüblichen PC laufen wird. Ein grosser Fernseher wird als Anzeige für die Kundenansicht verwendet.

Wir müssen nun noch das GUI zur Verfügung stellen.

### Syntaktische Ebene

Auf dieser Ebene ist die Frage nach dem Wissen des Benutzers wichtig. Dies kann von Benutzer-Art zu Benutzer-Art unterschiedlich ausgeprägt sein. Je nach Komplexität des Programmes und nach Arbeitsteilung muss das Programm dementsprechend angepasst werden. Allenfalls müssen die Benutzer sogar geschult werden, damit das Programm im Arbeitsalltag (oder auch im Privatgebrauch) korrekt und effizient eingesetzt werden kann.

In unserem Fall handelt es sich um durchschnittliche End-User-Anwender. Einige Personen haben bereits Erfahrung mit dem bestehenden Programm, andere werden neu mit dem Programm arbeiten. Die Benutzer wissen, welche Produkte welcher Kategorie zuzuordnen sind und kennen ihre jeweiligen Eigenschaften.

### Interaktionsebene

Wie der Benutzer mit dem System interagieren kann, sprich wie er mit dem Programm kommunizieren kann ist für den Benutzer sehr zentral. Beim Starbucks Manager kann er dies per Maus und Tastatur machen.

## Benutzer-Arten

Grundsätzlich können verschiedene Benutzer sehr verschiedene Fähigkeiten und Wünsche mitbringen. Diese können sich im Laufe der Zeit verändern. Man unterscheidet meist Neulinge, geübte Benutzer und Profis.

Unser GUI wird sehr übersichtlich aufgebaut sein und für falsche Eingaben werden direkt mit Fehlermeldungen ausgegeben. So sollen die Intuition und der Lerneffekt des Benutzers ihn schnell zu einem geübten Benutzer verändern. Daher legen wir das GUI auf diese Benutzergruppe aus (Reduktion). Für unerfahrene Benutzer stellen wir dafür den Menüpunkt ‘Help’ zur Verfügung, der die Anwendung kurz beschreibt.

Für Profis wäre allenfalls eine Möglichkeit, die Fehlerdialoge bei falschen Eingaben auszuschalten, um den User bei Tippfehlern nicht zu stören. Dies wird jedoch erst in einem weiteren Schritt implementiert – je nach Wunsch der User.

## Weitere Aspekte

### Aufbau

Damit sich ein neuer Benutzer in unserem Programm schnell zurechtfindet, halten wir uns an gewisse Standards. Die Menüleiste ist zuoberst zu finden, unten wird das Programm mit einem Footer abgeschlossen. Ganz links wird die Home-Seite eingeblendet, sie ist gleichzeitig der Einstiegspunkt. Die Kundenansicht wird nämlich jeden Morgen aufgestartet, Anpassungen am Menü werden tendenziell seltener gebraucht. Die Funktion ‘Help’ befindet sich wie bei den meisten Programmen ganz rechts in der Menüleiste.

Weiter verwenden wir ein Icon, damit der User in der Taskleiste auf den ersten Blick sehen kann, wo er den Starbucks Manager findet. Dies kann hilfreich sein, wenn er noch weitere Programme geöffnet hat und zwischen den Programmen wechseln will.

Bei Fehler poppt ein modaler Dialog mit einer detaillierten Fehlermeldung auf, den er mittels eines Buttons bestätigen muss. Wenn jedoch eine Funktion erwartungsgemäss funktioniert hat, wird links unten in der Stage für eine kurze Zeitdauer (ca. 2.5 sec) eine kurze Meldung ausgegeben. So weiss der User immer, ob seine Aufgabe erfüllt wurde oder ob und was falsch gelaufen ist. Dies ist wichtig für die Usability.

### Dialoge

Meldungen werden immer in einem Dialog geöffnet, welche der Benutzer mit einem Button bestätigen und gleichzeitig beenden kann. Dies ist er sich von den meisten bestehenden Programmen ebenfalls schon gewöhnt. Diese Dialoge sind modal, das bedeutet, der User muss zuerst den Dialog schliessen um weiterarbeiten zu können. Dies stellt sicher, dass die Fehlermeldung zur Kenntnis genommen wird.

Für die Eingabe des Preises wird eine Metapher verwendet. Eine Art Taschenrechner, welcher auf die benötigten Funktionen begrenzt ist, öffnet sich und mittels Klicks auf die Buttons füllt die Zahlen korrekt dem double-Format entsprechend ein. Auch dieser ist modal eingestellt, um keinen Datenverlust zu erleiden.

Die Menü-Option ‘Help’ öffnet einen nicht-modalen Dialog, in dem die Funktionen des Programmes erklärt sind. So kann der User im Help-Dialog nachschauen wie etwas funktioniert und es gleichzeitig im GUI ausführen. Das ist für den Benutzer viel angenehmer als wenn er zwischen allen Schritten den Help-Dialog schliessen und neu öffnen müsste.

### Grundsätze der Dialoggestaltung

Wir haben uns vermehrt auf die Grundsätze vom ISO 9241-110 geachtet. Unsere Applikation ist aufgabenangemessen, da sich die Applikation nur auf die 4 Anforderungen beschränkt, die wir in unseren UseCases abgebildet sind. Es sind keine unnötigen Fenster vorhanden.

Weiter sind dank der Textwahl der Menüs, der Buttons und der Labels die Steuerbarkeit und die Selbstbeschreibungsfähigkeit unseres GUIs gegeben. Mittels dem ‘Help’-Dialoges werden diese nochmals verstärkt, da darin das Vorgehen für jede Funktion exziplit beschrieben ist.

Der Erwartungen der User werden wir ebenfalls gerecht, weil die Applikation über einige üblichen Konzepte verfügt. Dies ist im Kapitel ‘Aufbau’ genauer beschrieben. Weiter wurden wie bereits erwähnt Metaphern verwendet, beispielsweise für die Preiseingabe (Taschenrechner)

Fehlertoleranz kann in unserem Programm nicht eingebaut werden, da die Daten zur Weiterverarbeitung korrekt sein müssen. Dafür wurde das Errorhandling entsprechend ausgebaut, sodass der User seine Fehler nach den Fehlermeldungen immer korrigieren kann. Teilweise wird der Fehler gar nicht angenommen, beispielsweise wenn der User irgendwo den Filedelimiter ‘|’ eingibt. Der einzige Punkt, an dem eine Art Fehlertoleranz betrieben wird, ist die Anzeige der Preise. Jeder Preis wird nämlich vor der Ausgabe auf das Format 0.00 umgewandelt. So kann der User den Preis gegebenenfalls auch im Format 0 oder 0.0 eingeben.

Die Individualisierbarkeit ist ebenfalls ein Thema, welches sich für unser Programm eher unprofessionell als nützlich erweist. Natürlich ist es schön, wenn der Benutzer sein Programm nach seinen Gutdünken anpassen kann. Der Einsatzbereich des Programmes ist jedoch auf den geschäftlichen Rahmen innerhalb der Starbucksfilialen eingegrenzt. Dort verwenden alle Mitarbeiter dasselbe Programm und den Kunden soll ein einheitliches Menü angezeigt werden, egal an welchem Ort sie sich in Starbucks einfinden. Die Mitarbeiter können ausserdem die Filialen wechseln, ohne sich zuerst an neue Einstellungen gewöhnen zu müssen. Deshalb haben wir die Individualisierung komplett weggelassen.

Unsere Applikation ist einfach und übersichtlich aufgebaut. Was zu Beginn eventuell falsch verstanden werden könnte wird im Help-Dialog aufgezeigt. Eventuelle Falscheingaben eines neuen Benutzers werden stets mittels Fehlermeldungen abgefangen. Dies ist wichtig für die Lernförderlichkeit des Programmes. Beim nächsten Mal wird der User zum Beispiel nicht wieder versuchen ‘|’ in einem Textfeld einzugeben, nachdem ihm der Fehler aufgezeigt wurde.

### Bedienung Funktionsorientiert / Objektorientiert

Grundsätzlich ist unser GUI sehr funktionsorientiert bedienbar. Zuerst wählt man aus, ob man etwas hinzufügen, löschen oder editieren möchte. Erst dann kommt das eigentliche Objekt zum Zug. Die einzigen Orte, an denen die Orientierung am Objekt wichtig ist, ist der Edit-Dialog und das Hinzufügen eines Produktes. Hier muss das Objekt überprüft werden, bevor die korrekten Textfelder angezeigt werden.

### Menü

Für die Menüleiste haben wir die altbekannte Platzierung links oben ausgewählt und darauf geachtet, dass sie ständig sichtbar ist. Da die Applikation nur wenige Funktionen besitzt, haben wir auf ein DropDown verzichtet.

### Styleguides

Für die Kundenansicht werden die Starbucks-Farben verwendet. So wird das Corporate Design eingehalten, der Kunde merkt sofort, dass dieses Menü zu einer Starbucks-Filiale gehören muss.

Als Icon verwenden wir das Starbucks-Logo, so findet der Mitarbeiter sofort das korrekte Programm, wenn er den Starbucks Manager sucht.

Allgemein ist die Applikation wie bereits mehrheitlich erwähnt einheitlich gestaltet. Die Farben sind mit diversen Grautönen schlicht gehalten. Einzig die Kundenansicht und die Anzeige, wenn eine Funktion ausgeführt wurde sind farbig hervorgehoben. Dabei wurde das Corporate Design verwendet und auf die Kontraste geachtet. Weisse Schrift auf dunkelgrünem Grund ist gut zu lesen, weshalb dies kein Problem darstellen sollte.

Damit sich Mitarbeiter und Kunden wohlfühlen, haben wir die Fluchtlinien genauer angeschaut und einige eliminiert. So sehen unsere Views ordentlicher und harmonischer aus.

Alle Buttons (inkl. Menüs) besitzen einen kurzen, prägnanten Text, sodass der Benutzer sich schnell zurechtfindet.

# Implementation

## Implementierung

### Verantwortlichkeiten

Für die Implementierung haben wir unser GUI in drei Verantwortlichkeitsbereiche unterteilt. Für den jeweiligen Verantwortlichen bedeutet dies, dass die Implementation sowie das Testing der jeweiligen Funktionen seine Aufgabe ist. Natürlich bestehen einige Abhängigkeiten zwischen den einzelnen Bereichen, die wir miteinander besprechen werden. So wird auch unsere Zusammenarbeit gewährleistet.

Die Kundenansicht, welche mittels WebView gestaltet wird, ist unter Tatyana Merlo’s Verantwortlichkeit. Auch das Grundlayout gehört in ihren Bereich. Das Hinzufügen der Daten und die Anzeige der ‘Help’-Seite liegt unter Nadja Stadelmann’s Bereich. Natalie Stalder ist für die Mitarbeiteransicht zuständig, welche gleichzeitig die Funktionen ‘Produkt bearbeiten’ und ‘Produkt löschen’ beinhaltet.

### Technischer Aufbau

#### GUI

Unser Template beinhaltet das Fenster der Applikation inklusive Titel, Menüleiste und Footer. Je nach Auswahl der Menübuttons durch den User werden nun die eigentlichen Fensterinhalte ausgetauscht. Der Inhalt der Kundenansicht wird betreffend des Layouts speziell behandelt. Alle Mitarbeiteransichten werden mittels demselben CSS-Stylesheet formatiert. Dadurch stellen wir sicher, dass der User bei der Anwendung eine einheitliche Ansicht zu sehen bekommt.

#### DataHolder

Um die Datenintegrität sicherzustellen, wird ein DataHolder verwendet. Er beinhaltet die möglichen Daten eines einzelnen Produktes.

Die Views AddItem und ChangeItem verwenden diesen, um für den User sicherzustellen, dass seine Daten zwischengespeichert sind. So klickt der User beispielsweise auf Add > Coffee und gibt den Namen eines neuen Produktes der Kategorie Beverage ein. Um ihm den Kategorienwechsel zu vereinfachen, werden die bereits eingegebenen Daten aus dem DataHolder in die neue Anzeige eingespeist.

#### Verbindung zum Model

Die Anbindung des GUIs an das Model wird über die Klasse ‘Menu’ gestaltet. In jener Klasse war zuvor die Konsoleneingaben und -überprüfungen zu finden. Sie hat bereits vorher schon mit dem Model kommuniziert, weshalb einige bestehende Methoden nun nur noch umgebaut werden müssen.

### Entwicklungsprozess

Bei der Implementation haben wir als erstes das Template erstellt. Parallel dazu wurden die Grundstrukturen der Menüoptionen ‘Modify’ und ‘Add’ festgelegt. Anschliessend folgte die Ausarbeitung dieser beider Menüs, sowie das Erstellen unserer Kundenansicht.

Zuerst wurden die notwendigen Elemente wie Labels, Textfelder und Buttons erstellt und ungefähr in die Grundstrukturen eingefügt. Dann ging es vor allem um das Datenhandling. Es folgte die Implementation der notwendigen Überprüfungen, sodass die Daten bereits im richtigen Format an die Businesslogik eingespeist werden.

Die Überprüfung des Preises wurde speziell implementiert. Wir bauten einen bestehenden Taschenrechner um, sodass er unseren Ansprüchen genügt. So kann nun mittels einiger Überprüfungen sichergestellt werden, dass der Preis immer als double gespeichert wird.

Je nach Useraktion müssen die Daten in verschiedenen Feldern beibehalten werden (bspw. genau bei unserer Preiseingabe: der eingegebene Preis muss auf das Preis-Textfeld in der jeweiligen Ansicht übertragen werden. Dafür haben wir eine eigene Klasse erstellt, den ‘DataHolder’. Diese ist zuständig für die Daten eines einzelnen Produktes und wurde zunächst static implementiert. Die Überlegung dafür war, dass der DataHolder grundsätzlich nicht als einzelne Instanz benutzt wird, sondern nur die Daten darin relevant sind. Dies hat jedoch anschliessend nicht so funktioniert wie gedacht. Deshalb bauten wir für diesen Datenaustausch das Observer-Pattern ein. Der DataHolder ist observable, die Views sind seine jeweiligen Observer. So wird die Datenintegrität zwischen dem DataHolder und den Views sichergestellt.

Am Schluss folgte die definitive Anbindung an das Model, sodass das Programm am Ende auch wirklich funktioniert und die verschiedenen Elemente hinzugefügt, angezeigt, bearbeitet und gelöscht werden können.

Parallel zu diesem Prozess schrieben wir das Error-Handling mittels des Errors-Dialogs, welcher überall verwendet werden kann. So wird die Einheitlichkeit für die Meldungen sichergestellt. Das Layout haben wir ebenfalls parallel zum Prozess verbessert.

Ganz am Ende fand die Implementation des ‘Help’-Dialoges statt für unerfahrene User, welche eventuell etwas nicht verstehen.

# Testing

## Testfallmatrix

Die Testfälle 1-4 gehen davon aus, dass das Programm bereits gestartet ist.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nr** | **UC** | **Testfall** | **Testbeschrieb** | **Erwarteter Output** | **Ergebnis** | **Status** |
| **1** | **UC1** | **Produkt hinzufügen** |  |  |  |  |
| 1.1 |  |  | Menü “Add” wählen, Produktkategorie auswählen, Daten eingeben, auf “add item” klicken. | Produkt wird hinzugefügt, eine Erfolgsmeldung poppt auf und verschwindet wieder. | Produkt wird hinzugefügt, eine Erfolgsmeldung poppt auf und verschwindet wieder. | OK |
| 1.2 |  |  | Menu "Add" wählen, Produktkategorie auswählen und Daten einfüllen. Produktkategorie wechseln. | Die eingegebenen Daten, welche bei beiden Produktkategorien gleich sind, bleiben ausgefüllt. | Die eingegebenen Daten, welche bei beiden Produktkategorien gleich sind, bleiben ausgefüllt. | OK |
| 1.3 |  |  | Menu "Add" wählen, ein Produkt hinzufügen, mit demselben Namen, wie ein bereits existierendes Produkt. "Add item" klicken. | Eine Fehlermeldung wird ausgeben, dass bereits ein Produkt mit demselben Namen existiert. | Eine Fehlermeldung wird ausgeben, dass bereits ein Produkt mit demselben Namen existiert. | OK |
| 1.4 |  |  | Menu "Add" wählen, ein verbotenes Zeichen (z.B. "|") in einem der Felder eingeben. | Eine Fehlermeldung wird ausgegeben, dass dieses Zeichen nicht erlaubt ist, Zeichen wird nicht geschrieben. | Eine Fehlermeldung wird ausgegeben, dass dieses Zeichen nicht erlaubt ist, Zeichen wird nicht geschrieben. | OK |
| **Nr** | **UC** | **Testfall** | **Testbeschrieb** | **Erwarteter Output** | **Ergebnis** | **Status** |
| **2** | **UC2** | **Produkt anpassen** |  |  |  |  |
| 2.1 |  |  | Menu "Modify" wählen, bei dem zu editiernden Produkt "edit" auswählen, die Daten anpassen und mit Klick auf "edit" die Änderungen abspeichern. | Das Produkt wird angepasst, das Editier-Fenster geschlossen, eine Erfolgsmeldung poppt für kurze Zeit auf, die Modify-Scene mit allen Produkten wird neu geladen und angezeigt. | Das Produkt wird angepasst, das Editier-Fenster geschlossen, , eine Erfolgsmeldung poppt für kurze Zeit auf, die Modify-Scene mit allen Produkten wird neu geladen und angezeigt. | OK |
| 2.2 |  |  | Menu "Modify" wählen, bei dem zu editiernden Produkt "edit" auswählen, die Daten anpassen und mit Klick auf "edit" die Änderungen abspeichern ohne Erfolg. | Das Editier-Fenster bleibt offen und eine Fehlermeldung wird angezeigt, dass das Produkt nicht editiert werden kann. | Das Editier-Fenster bleibt offen und eine Fehlermeldung wird angezeigt, dass das Produkt nicht editiert werden kann. | OK |
| 2.3 |  |  | Menu "Modify" wählen, bei dem zu editiernden Produkt "edit" auswählen, die Daten anpassen und das Editier-Fenster schliessen (nicht über "edit"-Button). | Das Fenster schliesst sich, das Produkt wird nicht angepasst und die Modify-Scene der Produkte wird nicht neu geladen. | Das Fenster schliesst sich, das Produkt wird nicht angepasst und die Modify-Scene der Produkte wird nicht neu geladen. | OK |
|  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nr** | **UC** | **Testfall** | **Testbeschrieb** | **Erwarteter Output** | **Ergebnis** | **Status** |
| 2.4 |  |  | Menu "Modify" wählen, bei dem zu editiernden Produkt "edit" auswählen, beim Daten-Anpassen ein verbotenes Zeichen eingeben (z.B. "|"). | Eine Fehlermeldung erscheint, das unzulässige Zeichen wird nicht zum Text hinzugefügt. | Eine Fehlermeldung erscheint, das unzulässige Zeichen wird nicht zum Text hinzugefügt. | OK |
| 2.5 |  |  | Menu "Modify" wählen, bei dem zu editiernden Produkt "edit" auswählen, den Namen des Produktes auf einen Namen eines anderen, existierenden Produktes ändern. "Edit"-Button klicken. | Eine Fehlermeldung wird ausgegeben, dass bereits ein Produkt mit demselben Namen existiert und das Produkt so nicht verändert werden kann. | Eine Fehlermeldung wird ausgegeben, dass bereits ein Produkt mit demselben Namen existiert und das Produkt so nicht verändert werden kann. | OK |
| **3** | **UC3** | **Menu in Kundensicht anzeigen** |  |  |  |  |
| 3.1 |  |  | Menu "Home" wählen oder Programm starten. | Kundensicht in HTML wird und sortiert nach Produkttyp ausgegeben. | Kundensicht in HTML wird und sortiert nach Produkttyp ausgegeben. | OK |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nr** | **UC** | **Testfall** | **Testbeschrieb** | **Erwarteter Output** | **Ergebnis** | **Status** |
| **4** | **UC4** | **Produkt löschen** |  |  |  |  |
| 4.1 |  |  | Menu "Modify" wählen und beim zu löschenden Produkt auf den "Delete"-Button klicken. | Das Produkt wird gelöscht. Eine Erfolgsmeldung ploppt für kurze Zeit auf und die Modify-Scene mit allen Produkten wird neu geladen. | Das Produkt wird gelöscht. Eine Erfolgsmeldung ploppt für kurze Zeit auf und die Modify-Scene mit allen Produkten wird neu geladen. | OK |
| 4.2 |  |  | Menu "Modify" wählen und beim zu löschenden Produkt auf den "Delete"-Button klicken ohne Erfolg. | Das Produkt kann nicht entfernt werden. Eine Fehlermeldung ploppt auf und die Modify-Scene der Produkte wird nicht gelöscht. | Das Produkt kann nicht entfernt werden. Eine Fehlermeldung ploppt auf und die Modify-Scene der Produkte wird nicht gelöscht. | OK |
| 5 |  | Leere Modify Scene anzeigen | "products.txt" leeren und das Programm starten. Menu "Modify" wählen. | Eine Meldung wird ausgegeben, dass das Menu leer ist und neue Produkte unter "Add" erfasst werden können. | Eine Meldung wird ausgegeben, dass das Menu leer ist und neue Produkte unter "Add" erfasst werden können. | OK |
| 6 |  | Leeres Menu anzeigen | "products.txt" leeren und das Programm starten. | Im Home werden die Kategorien leer ausgegeben. | Im Home werden die Kategorien leer ausgegeben. | OK |

Neben Tests auch ein Benutzer-Feedback verfassen

Benutzerfeedback

# Anhang

## Projektauftrag – WebApplikation realisieren

|  |  |  |
| --- | --- | --- |
| **Projekttitel:** | Individueller Reiseblog | |
| **Projektnummer:** | 00001 | |
| **Projektart:** | Software-Lösung  Daten: Serverbasierend  UI: Webbasierend | |
| **Projektleiter/in:** | Tatyana Merlo | |
| **Projektauftraggeber/in:** | Herr D. A. Waldvogel | |
| **Projektkunde(n):** | Offene Kundschaft | |
| **Projektdauer:** | Geplanter Beginn: 13.09.2017  Geplantes Ende: 01.11.2017 | |
| **Ausgangssituation / Problembeschreibung:** | Reiseziele auf einer Karte festhalten  Blogeinträge zu den Reisezielen erfassen  Login notwendig  Registrierung für neue Kunden | |
| **Projektgesamtziel:** | Jeder kann sich auf der Webseite registrieren. Wenn die Personen eingeloggt sind, können sie ihre bisherigen Reiseziele erfassen, dazu Blogeinträge schreiben und Fotos hochladen. Es sind nur ihre eigenen Einträge sichtbar, daher ist auch ein Login notwendig. | |
| **Projektteilziele und -ergebnisse:** | |  |  | | --- | --- | | **Teilziele:** | **Ergebnisse:** | | Datenbank Design und Erstellung | * Datenbank ist bereit für Laufzeit, DB-Server installiert | | Verzeichnissystem | * Installieren und Aufbauen des Verzeichnissystems auf dem Webserver. Bereit für Laufzeit. | | Entwicklung des serverbasierten Services | * Verknüpfung zu Datenbank und Verzeichnissystem | | User Interface | * Design und Verknüpfung zu internen Logik fertig gestellt | | |
| **Meilensteine:** | |  |  | | --- | --- | | **Meilensteine:** | **Datum:** | | Projektauftrag | 13.09.2017 | | Serverteil fertig gestellt (Datenbank, Service, Dateisystem) | 18.10.2017 | | Clientteil abgeschlossen (User Interface) | 18.10.2017 | | Testing | 25.10.2017 | | Abgabe | 01.11.2017 | | |
| **Projektorganisation:** | **Kernteam**:   * Tatyana Merlo, Projektleiterin * Natalie Stalder, Webentwickler, Backend * Nadja Stadelmann, Webentwickler, Frontend   **Sonstige Projektbeteiligte**:   * Web- und Server-Hoster | |
| **Projektressourcen:** | |  |  | | --- | --- | | **Ressourcen:** | **Menge:** | | Personal |  | | Entwicklungsumgebung | XAMPP | | -Programmierung php | 3 | | -Design CSS | 3 | |  |  | | |
| **Projektbudget:** | 200 CHF | |
| **Projektrisiken und -unsicherheiten:** | * Technische Umsetzungsprobleme * fehlendes Know-How * schlechtes Zeitmanagement | |
| **Unterschrift /**  **Abnahme** | Kunde:  Dominik Waldvogel  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Projektleiter:  Tatyana Merlo  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |