IPA BERICHT

AUTOR(EN) : Bosshard Fabrice
VERSION : 1.0
STATUS : Draft
QUELLE : Atos
DOKUMENTENDATUM : 09. November 2018
ANZAHL DER SEITEN : 91

Änderungshistorie

Version	Datum	Beschreibung	Autor(en)
0.1	30.10.2018	Initiale Dokumentationsstruktur erstellt.	Fabrice Bosshard
0.2	31.10.2018	Kapitel «Informieren» abgeschlossen	Fabrice Bosshard
0.3	01.11.2018	Kapitel «Planen» abgeschlossen	Fabrice Bosshard
0.4	02.11.2018	Kapitel «Entscheiden» abgeschlossen	Fabrice Bosshard
0.5	07.11.2018	Kapitel «Realisieren» abgeschlossen	Fabrice Bosshard
0.6	09.11.2018	Kapitel «Kontrollieren» abgeschlossen	Fabrice Bosshard
0.7	09.11.2018	Kapitel «Auswerten» abgeschlossen	Fabrice Bosshard
0.8	09.11.2018	Dokumentationsabschluss	Fabrice Bosshard

Tabelle 1: Änderungshistorie

Inhalt

Teil 1: Umfeld und Ablauf

Teil 2: Projekt

1 Aufgabenstellung

Aufgabenstellung Modul 223 nach PkOrg. Die Aufgabenstellung wird vom Auftraggeber festgelegt und ist anbei dokumentiert.

1.1 Projekt

Projekttitel:	Just Muesli
Im Auftrag von:	Remo Steinmann Modul 223
Auftragnehmer:	Fabrice-Ronny Bosshard
Starttermin:	31. Oktober 2018
Abgabetermin:	09. November 2018
Geplanter Projektaufwand:	5 Arbeitstage à 6 Stunden (30h)

Erstellung einer Multi-User-Applikation, mit Frontend, Backend und Anbindung an eine relationale Datenbank.

1.2 Ausgangslage

Die Aufgabenstellung existiert bereits aus den Informatik-Schweizermeisterschaften vom Jahr 2018 (ICT Skills 2018). Sie wurde für dieses Modul angepasst, um den Anforderungen an die Applikation zu entsprechen (siehe 2.8 Anforderungen an die Applikation).

Das Projekt wird von Grund auf aufgebaut und implementiert. Der Auftragnehmer muss neben den Standard .NET-Kenntnissen auch solide Kenntnisse in Entity Framework und WPF (Windows Presentation Foundation) haben.

Die Struktur der Dokumentation soll bereits als Vorlage existieren, damit bei Projektbeginn direkt mit dem Schreiben angefangen werden kann.

1.3 Detaillierte Aufgabenstellung

Im Anhang finden sie die detaillierte Aufgabenstellung der ICT-Skills Schweiz. Sie beschreibt die funktionalen Anforderungen an die Applikation und gibt durch Wire Frames vor, wie das Layout der Benutzeroberfläche aussehen muss. Es existieren neben Benutzerfreundlichkeit keine weiteren Kriterien für das Styling der Applikation.

1.4 Management Summary

Kurze Repräsentation der Aufgabenstellung von ICT-Skills Schweiz.

„Just Muesli" ist eine neu gegründete Firma welche sich als Ziel gesetzt hat, allen Kunden ein perfektes, konfigurierbares Muesli bieten zu können. Die Kunden können sich selber, aus einer Auswahl von Zutaten, Muesli online erstellen. So wird ermöglicht, dass man unzählige einzigartigen Muesli für jeden Benutzer erstellen kann.

1.5 Applikationsstruktur

Das folgende Diagramm zeigt die verschiedenen Teile der Applikation. Jeder Knoten repräsentiert ein Fenster, welche jeweils im Anhang (Aufgabenstellung ICT-Skills) detailliert beschrieben werden.

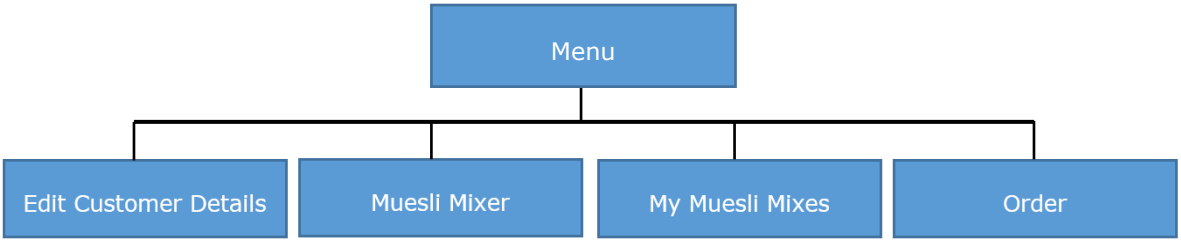


Abbildung 1: Applikationstruktur

1.6 Technologie-Stack

IDE:	Visual Studio 2017 (mit ReSharper)
Database:	SQLite mit Entity Framework
Front- / Backend:	C# 7.0 mit Windows Presentation Foundation (WPF)
Testing:	NUnit
Versionsverwaltung:	Git mit TortoiseGit

Programmarchitektur wird nach MVVM (Model View ViewModel) umgesetzt.

1.7 Firmenstandards

Für Qualitätssicherheit und Vulnerabilitätsüberprüfung wird SonarQube verwendet.
Als Dokumentationsstandard gibt es bei der Atos ein spezielles Template für Reports. Dieses Template gibt Layout, Schriftarten, Farben, Kopf- / Fußzeilen und Absätze vor.

1.8 Anforderungen an die Applikation

Folgende Anforderungen müssen von der Applikation erfüllt werden:

- Multi-User-Applikation
- Relationale Datenbank
- Objektorientierte Programmierung
- Mehrere Clients müssen gleichzeitig auf den gleichen Datenbestand zugreifen
- Transaktionssicherheit muss gewährleistet werden
- Zentrale Datenbank

1.8.1 Zusätzliche Kriterien

Während der Arbeit müssen weiterhin folgende Kriterien beachtet werden:

- 225 Versionsverwaltung mit Verwaltungs-SW
- 235 Entwurf mit UML
- 166 Codingstyle – lesbarer Code
- 250 Schichtentrennung
- 130 Vollständiges ERM bzw. Datenmodell
- 125 Gliederung des Programms

- 164 Codierung: Fehlerbehandlung

1.9 Vorkenntnisse

In diesem Abschnitt werden die bereits erlangten Kenntnisse aufgelistet.

Projektkenntnisse:

- C# (2 Jahre)
- WPF (2 Jahre)
- Visual Studio (2 Jahre)
- Angular *TypeScript* (1 Jahr)
- Git (2 Jahre)
- CI (1 Jahr)
- DI (1 Jahr)

1.10 Vorarbeiten

In der IPA muss eine vorgegebene Arbeit umgesetzt und dokumentiert werden. Oftmals sprengen alle Tätigkeiten in dieser Arbeit den Zeitrahmen und deshalb werden gewisse Arbeiten bereits zuvor erledigt. In diesem Kapitel werden alle Vorbereitungen für die Probe-IPA deklariert.

- Initiale Dokumentationsstruktur aufbauen nach IPERKA.
  - Struktur
  - Layout
  - Verzeichnisse
- Wissen für Entity Framework aufgebaut.
- Transaktionssicherheit implementiert in Lernprojekt

2 Projektorganisation

Dieses Kapitel gibt einen Überblick über den Aufbau der Organisation, informiert über die beteiligten Personen und zeigt die Vorgehensweise im Allgemeinen.

2.1 Beteiligte Personen

Person	Rolle	Verantwortung	Kontakt
Torben Dziuk	Auftraggeber & Verantwortliche Fachkraft	Aufgabe definieren & Unterstützung Kandidat	torben.dziuk@atos.net
Remo Steinmann	Hauptexperte	Rahmenbedingungen setzen & Bewertung der IPA	remo.steinmann@siemens.com
Fabrice Bosshard	Kandidat	Umsetzung der Arbeit	fabrice.bosshard.external@atos.net fabrice.bosshard@siemens.com

Tabelle 2: Beteiligte Personen

2.2 Projektmanagementmethode

Im Betrieb arbeiten wir mit der Projektmanagementmethode Scrum, welche sich sehr gut für die agile Prozessentwicklung eignet. Bei Scrum werden die verschiedenen Arbeitsschritte in Zyklen (Sprints) abgehandelt, was das Anpassen von Vorgaben zwischen den Zyklen und die transparente Kommunikation innerhalb des Entwicklerteams ermöglicht.
Scrum ist jedoch für eine IPA nicht gut geeignet, da der Mehraufwand für die Formalitäten (Daily, Retro etc.) zu hoch wäre.
Ich habe mich deshalb für die Projektmanagementmethode IPERKA (**I**nformieren, **P**lanen, **E**ntscheiden, **R**ealisieren, **K**ontrollieren, **A**uswerten) entschieden, da ich damit schon viel Erfahrungen in der Schule sammeln konnte. Diese klassische Projektmanagementmethode eignet sich für die IPA perfekt, da der Ablauf respektive die Struktur mit den Anforderungen der Arbeit übereinstimmt.

2.3 Backup-Konzept

Bei der Umsetzung der Arbeit ist es essentiell, dass man stets auf eine valide Version der Arbeit zurückgreifen kann. Wenn man zum Beispiel einen groben Fehler macht und die ganze Applikation nicht mehr zum Laufen kriegt, ist es wichtig, dass man nicht von Neuem beginnen muss, sondern auf den Stand vor dem Fehler zurückgreifen kann.
Die Dokumentation und der Quellcode werden in ein Git-Repository geladen und sind somit zentral in einer Cloud gespeichert. Dazu wird das externe Tool GitHub verwendet. GitHub macht bei sich selber immer wieder Backups und so muss ich mich nicht um ein Backup-System kümmern. Das Repository muss stets aktualisiert werden, damit möglichst immer der neuste Stand verfügbar ist. Um dies zu ermöglichen, sollte für jede neue Funktionalität ein Commit erstellt werden, welcher den neuen Stand darstellt (Daumenregel: 2-4 Mal am Tag).
Als weitere Absicherung wird ein USB-Stick genutzt, auf welchem mindestens ein Mal am Tag ein komplettes Backup erstellt wird. Im Ernstfall würde man als erstes versuchen auf das Git-Repository zurückzugreifen, doch wenn dies ebenfalls fehlschlagen sollte, gibt es noch eine Sicherung auf dem Stick.

3 Zeitplanung

In diesem Kapitel werden die verschiedenen Aufwände geschätzt und aufgelistet. Die Zeitplanung dient als Leitfaden für den Projektablauf und hilft bei der Selbstkontrolle.

3.1 Meilensteine

Meilensteine sind wichtige Teilpunkte im Projektverlauf. Sie werden als Prüfpunkte verwendet und wirken sich positiv auf die Qualitätssicherung aus. Man kann somit die erledigten Ergebnisse mit den erwarteten Vorgaben abgleichen.

Meilenstein	Erfüllungskriterien	Datum
Projektbeginn	Kick-off der Probe IPA	31.10.2018 08:30 Uhr
Informieren	Auftragsanalyse durchgeführt und dokumentiert	31.11.2018 17:00 Uhr
Planen	Anwendungsfälle, Architektur, Testfälle planen, Logging-konzept geplant	01.11.2018 17:00 Uhr
Entscheidung	Tools / Framework & Datenbankentscheidungen definiert	02.11.2018 11:00 Uhr
Realisieren	Implementation der Arbeit & Dokumentation realisiert	07.11.2018 17:00 Uhr
Kontrollieren	Die zuvor definierten Kontrollkriterien überprüft und Fazit daraus gezogen	09.11.2018 11:00 Uhr
Auswerten	Reflexion der Arbeit und des Arbeitsverhalten erstellt & Kurzfassung geschrieben	09.11.2018 16:00 Uhr
Projektabschluss	Projektarbeit abgeschlossen	09.11.2018 17:00 Uhr

Tabelle 3: Meilensteine

3.2 Massnahmen bei Verzug

Sobald ein Meilenstein oder eine Tätigkeit nicht zur definierten Zeit erledigt werden konnte, muss eine Massnahme ergriffen werden, damit weitere wichtige Tätigkeiten nicht in Gefahr geraten. Dazu würden die kleineren Teil-Tätigkeiten, welche nicht grossen Einfluss auf den Fertigstellungsgrad haben, ignoriert und als optional für nach der Fertigstellung markiert werden. Es werden jedoch nur Teil-Tätigkeiten aus der Realisierungs-Phase weggenommen, damit der allgemeine Projektablauf nicht manipuliert wird. Das heisst, wenn eine Massnahme ergriffen werden müsste, würde als Erstes die letzte Ansicht der Benutzeroberfläche ignoriert werden (In dem Beispiel => «Order Confirmation» würde als optional gesetzt werden (Siehe Anhang Aufgabenstellung)).

3.3 Gantt-Diagramm

Zeitplan		Datum	Mittwoch 31.10.2018			Donnerstag 01.11.2018			Freitag 02.11.2018			Mittwoch 07.11.2018			Freitag 09.11.2018		
		Stunden	2	4	6	2	4	6	2	4	6	2	4	6	2	4	6
I	Projektorganisation definieren	Soll															
		Ist															
	Aufwandschätzung und Meilensteine definieren	Soll															
		Ist															
	Ist- / Sollanalyse beschreiben	Soll															
		Ist															
	Systemübersicht definieren.	Soll															
		Ist															
P	Use Cases erstellen	Soll															
		Ist															
	Testkonzept erstellen	Soll															
		Ist															
	Software-Architektur beschreiben	Soll															
		Ist															
	Datenbank-Architekur beschreiben	Soll															
		Ist															
	Logging-Konzept	Soll															
		Ist															
E	Tools und Framework entscheide dokumentieren	Soll															
		Ist															
	Datenbankimplementation dokumentieren	Soll															
		Ist															
R	Umgebung aufsetzen (Projektbeginn)	Soll															
		Ist															
	Verknüpfung zu SonarQube	Soll															
		Ist															
	Logger aufsetzen	Soll															
		Ist															
	Datenbankstruktur erstellen (Entity Framework)	Soll															
		Ist															
	Applikationsstruktur erstellen (View, Viewmodel & Model)	Soll															
		Ist															
	Benutzer authentifikation implementieren	Soll															
		Ist															
	Business-Logik mit View und DatabaseClient verknüpfen	Soll															
		Ist															
K	Test durchführen	Soll															
		Ist															
	Testprotokoll & Fazit erstellen	Soll															
		Ist															
A	Reflexion & Fazit schreiben	Soll															
		Ist															
	Erweiterbarkeit	Soll															
		Ist															
	Kurzfassung schreiben	Soll															
		Ist															
	Gantt-Diagramm erstellen	Soll															
		Ist															
	Arbeitsjournal führen	Soll															
		Ist															
Meilensteine			Projektbeginn		Informieren			Planen	Entscheiden					Realisieren	Kontrollieren		Auswerten

Tabelle 4: Gantt-Diagramm

4 Arbeitsprotokoll

Das Arbeitsprotokoll dient zur Repräsentation des Tagesablaufs und der Fortschrittsbelegung. An jedem Arbeitstag wird das Arbeitsprotokoll erweitert und gepflegt. Es wird für jeden Tag mindestens eine Stunde für das Protokoll eingerechnet, damit die Qualität des Produkts sichergestellt werden kann. Es zeigt die geplanten Aufgaben pro Tag und stellt dar, ob es von der Tagesplanung Abweichungen gegeben hat. Zudem enthält das Protokoll jeweils eine Reflexion für jeden Tag, bei der man auf Erfolge, Misserfolge und den daraus zu schliessenden Massnahmen achtet.
Zur Unterstützung der Reflexion wird mit Continious Integration (CI) gearbeitet. Die CI nimmt den eingecheckten Code und prüft ihn auf folgende Kriterien: Code Coverage, Check-Style, Vulnerabilities.

4.1 Mittwoch, 31. Oktober 2018

Tag 1 (31.10.2018)			
Phase: Informieren			
Tätigkeit	**Soll-Zeit (Stunden)**	**Ist-Zeit (Stunden)**	**Bemerkung**
Aufgabenstellung mit Hauptexperte festlegen	0.25	0.25	Änderung der Aufgabenstellung wird im Arbeitsjournal und nicht in der konkreten Aufgabenstellung festgehalten
Gantt-Diagramm erstellen	0.75	0.75	-
Projektorganisation definieren	0.75	0.5	-
Aufwandschätzung und Meilensteine definieren	1	1	-
Ist- / Sollanalyse beschreiben	1.5	2	-
Systemübersicht definieren	0.75	1	-
Use Cases beginnen	0.5	0	Zeit war zu knapp / Am nächsten nachholen.
Arbeitsjournal schreiben	1	0	Zeit war zu knapp / Am nächsten Morgen geschrieben.
Total	6.5 Stunden	5.5 Stunden	Wir hatten an diesem Tag weniger Zeit

Tabelle 5: Arbeitsprotokoll 31.10.2018

Heute war der erste Tag an dem wir an der IPA arbeiten konnten. Zu Beginn des Tages hatten wir noch einen Kurs über das designen von Flip-Charts. Aus diesem Grund haben wir bereits ein bisschen später mit der Arbeit angefangen. Diese fehlende Zeit wird am nächsten Tag mit Mehrarbeit kompensiert. Ich hatte mit dem Hauptexperten anschliessend eine kurze Besprechung in der wir über den Zeitplan und die Aufgabenstellung gesprochen haben. Wie ich einen Tag vor Projektbeginn gemerkt habe, kann ich nicht ganz mit der Aufgabenstellung fortfahren, welche wir definiert hatten. Ich wollte eigentlich die Datenbank mit SQLite erstellen, jedoch ist die Anbindung an das Entity Framework ein zu grosser Mehraufwand und deshalb habe ich mich für MSSQL umentschieden. Diese Entscheidung wurde mit dem Hauptexperten abgesprochen und akzeptiert.

4.1.1 Erfolge

Trotz der fehlenden Zeit für den heutigen Tag, konnte ich sehr viele Arbeiten im Teil «Informieren» erledigen. Ich habe mich vor allem intensiv mit der Ist-/Sollanalyse beschäftigt und diese ausführlich dokumentiert. Es hat zwar etwas länger als geplant gedauert, dafür habe ich eine hohe Qualität sichergestellt.

Obwohl der ganze Raum relativ laut war konnte ich mich sehr gut auf meine Arbeit konzentrieren und war stets voll dabei. Ich denke ich habe trotz verschiedenen Zwischenfälle, für den heutigen Tag, das volle Potenzial mögliche herausgeholt.

4.1.1 Misserfolge

Der Einstieg in die Arbeit war etwas holprig, da wir sowieso schon weniger Zeit als geplant erhielten, und ich etwas nervös wegen der Arbeit war. Deshalb war ich bei den ersten Tätigkeiten, wie Zeitplan oder Projektorganisation, ziemlich unsicher und wusste nicht, ob das richtig ist was ich mache. Auch mein Zeitplan konnte ich nicht so ausführlich machen wie ich es eigentlich wollte, da der Zeitplan (Gantt-Diagramm) eigentlich ein grobes Planungstool ist und wir nicht genügend Zeit hatten dafür hatten. Aus diesem Grund habe ich schnell das Gefühl bekommen, dass ich zu sehr vom Zeitplan abweiche. Mit diesem Stress konnte ich nicht sehr gut arbeiten und hatte auch mühe mich zu konzentrieren am Anfang. Mit der Zeit wurde mir bewusst, dass es nicht bringt mich selber zu stressen, da meine Arbeitsleistung dann nur abnimmt.

4.1.1 Massnahmen

Der Tag ist nicht so ganz gelaufen, wie ich es mir erhofft habe. Ich kam in Stress und das hat mich schlussendlich nur blockiert. Ich muss für die restlichen Tage unbedingt versuchen einfach ruhig zu bleiben. Ich darf mich selber nicht stressen. Auch wenn ich mit dem Zeitplan hinterher hange, muss ich versuchen «cool» zu bleiben, um die Qualität und vor allem auch die Arbeitsmotivation nicht in den Keller zu hauen.
Ich habe die fehlende Zeit des Tages auch nicht wirklich in meinen Zeitplan mitaufgenommen und deshalb kam ich dann sehr rasch in Verzug. Ich muss versuchen eine genauere Tagesplanung zu machen, denn ich bin eigentlich ein sehr schneller Arbeiter, jedoch hatte ich mir gestern einfach zu viel vorgenommen.

4.1.1 Fazit

Trotz den erlebten Missstände finde ich der erste Tag war ein Erfolg. Es ist eine Probe-IPA (zum Glück) und da dürfen gewisse Sachen schieflaufen und für mich war es halt ein Erfolg, weil ich daraus eine wichtige Erkenntnis erlangt habe. Ich habe gelernt, dass ich einfach unter Stress nicht gut arbeite und das ist etwas, dass ich in den letzten Jahren nie wirklich gehabt habe. Ich war eigentlich immer sehr ruhig und davon überzeugt, dass ich gut abschliessen werde in einem Test oder ähnlichem. Nun war es heute mal anders und ich fand es spannend zu sehen wie ich darauf reagiere. Diese Kenntnis werde ich sicherlich für die restlichen tage nutzen. Es ist für mich auch wichtig, dass ich eigentlich möglichst viele Fehler oder Fehlüberlegungen während der Probe-IPA mache, denn nur so kann ich mich auf die richtige IPA vorbereiten. Ich sehe die heutigen Fehler nicht als schlimm an, sondern konzentriere mich darauf, dass sie nicht mehr passieren und dass ich daraus lernen kann.

4.2 Donnerstag, 01. November 2018

Tag 2 (01.11.2018)			
Phase: Planen			
Tätigkeit	**Soll-Zeit (Stunden)**	**Ist-Zeit (Stunden)**	**Bemerkung**
Use Cases erstellen	1.5	2	
Testkonzept erstellen	1	1.5	Probleme mit Word-Template
Software-Architektur beschreiben	1.5	1	
Datenbank-Architektur beschreiben	1	1	
Logging-Konzept	1	0.5	
Arbeitsjournal führen	1	1	
Total	7 Stunden	7 Stunden	

Tabelle 6: Arbeitsprotokoll 01.11.2018

Heute war definitiv besser als Gestern. Ich konnte den ganzen Tag durcharbeiten und habe so ziemlich alles abgeschlossen, was ich mir heute vorgenommen habe. Man muss dazu auch sagen, dass ich eigentlich noch ein Gespräch mit dem Hauptexperten gehabt hätte und ich deshalb wie 45 Minuten mehr arbeiten konnte. Dieses Gespräch wird morgen nachgeholt, weshalb ich befürchte, dass es morgen etwas enger wird mit dem Zeitplan.

4.2.1 Erfolge

Das Beste was ich aus dem Tag berichten kann ist, dass ich meine Massnahme des letzten Tages umgesetzt habe. Ich war heute viel bequemer unterwegs und habe direkt gemerkt, dass die Information viel flüssiger aus mir herauskommen. Gestern hatte ich Mühe, da ich durch den Stress blockiert war. Ich habe als Hilfe praktisch den ganzen Tag Musik gehört. Die Musik konnte mich sehr gut beruhigen und ich konnte mich wesentlich besser konzentrieren. Natürlich ist es auch ein Erfolg, dass ich meine Tagesplanung vollständig umsetzen konnte, jedoch muss man eben auch dazu beachten, dass ich mein Expertengespräch nicht wie geplant hatte und deshalb mehr Zeit hatte.

4.2.1 Misserfolge

Das Planen ist etwas, was mir immer wieder Mühe macht. Ich brauche zum Teil etwas länger, bis ich bei solchen Themen mir eine Vorstellung geschaffen habe und deshalb musste ich dann einfach viel intensiver dokumentieren. Momentan spüre ich sehr stark meine beanspruchten Finger und hoffe das bessert sich morgen. Was mir ausserdem überhaupt nicht passt, ist der Arbeitsort. Ich kann nicht einen ganzen Tag auf einem «schäbigen» Stuhl sitzen, weil ich sehr schnell Rückenschmerzen bekomme.

4.2.1 Massnahmen

Auf jeden Fall muss ich mich heute etwas ausruhen. Ich sollte ein bisschen früher ins Bett, da ich gegen Ende des Tages ziemlich Mühe hatte mich noch zu Leistungen zu erzwingen. Ich trinke eigentlich keinen Kaffee mehr, doch ich denke für diese Arbeit könnte es was nutzen.

4.2.1 Fazit

Ich bin sehr zufrieden von den Leistungen, welche ich erbracht habe. Ich bin auf jeden Fall für das Expertengespräch morgen vorbereitet. Die ersten beiden Schritte von IPERKA sind nun endlich abgeschlossen und ich kann mit dem Implementieren beginnen.

4.3 Freitag, 02. November 2018

Tag 1 (31.10.2018)			
Phase: Realisieren			
Tätigkeit	**Soll-Zeit (Stunden)**	**Ist-Zeit (Stunden)**	**Bemerkung**
Umgebung aufsetzen	0.5	0.5	
SonarQube verknüpfen	0.5	0.5	
Expertengespräch	0.5	0.5	
Datenbankstruktur erstellen (Entity Framework)	1.5	1.5	
Applikationsstruktur erstellen (View, ViewModel & Model)	2.5	3	
Logger aufsetzen	0.5	0.5	
Arbeitsjournal schreiben	1	1	
Total	7 Stunden	7.5 Stunden	

Tabelle 7: Arbeitsprotokoll 02.11.2018

Heute Morgen hatte ich das erste Expertengespräch. Wir sind gemeinsam durch die Dokumentation gegangen und der Experte hat mir Feedback zu meiner Arbeit gegeben. Die zwei wichtigsten Inputs die er mir gegeben hat, waren bezüglich Zeitplanung und Entscheidungsfindung. Ich hatte zu Beginn ein Gantt-Diagramm, welches ziemlich grob geplant war, meine Meilensteine und eine Aufwandschätzung, welche ziemlich genau war. Es war für mich jedoch kaum möglich die Aufwandschätzung zu befolgen, da sie für diesen Auftrag viel zu genau war. Wir haben besprochen, dass es am meisten Sinn macht, wenn man eine gröbere Planung, wie das Gantt-Diagramm, über die gesamte Projektzeit macht und für jeden einzelnen Tag, im Arbeitsjournal, noch detaillierter plant.
Bei der Entscheidungsfindung war vor allem das Problem, dass ich nicht wirklich Entscheide getroffen habe, sondern bereits getroffene Entscheidung dokumentiert habe. Der Experte hat mir gezeigt, was in der Phase «Entscheiden» wirklich stehen sollte und ich werde diese Information sicherlich in meiner IPA nutzen.
Den restlichen Tag habe ich vollständig für das Implementieren der Arbeit genutzt. Ich habe die meisten Views aufgebaut und alle benötigten Klassen mit ihren Feldern und Eigenschaften erstellt. Ausserdem habe ich mir aus meinen Models die Datenbank generieren lassen.

4.3.1 Misserfolge

In der Mitte des Tages habe ich bemerkt, dass die Aufgabenstellung den Zeitrahmen dieser Arbeit sprengt. Die Teilnehmer der SwissSkills hatten zwar nur 8 Stunden Zeit für die Arbeit, mussten jedoch praktisch gar nicht auf die Qualität und Coderechtlinien achten. Also wenn ich meinen Zeitplan befolgen möchte, habe ich zwei ganze Tage à 6 Stunden Zeit. Heisst also nur 4 Stunden mehr als unser Schweizermeister. Da wurde mir schnell klar, dass ich nicht fertig werde ohne Überstunden zu machen.

4.3.1 Massnahmen

Da ich mit dem Zeitplan hinterher hänge, muss ich meine initiale Aufgabenstellung kürzen. Wie in meinem Massnahmen-Konzept im Kapitel «Zeitplanung» beschrieben, fange ich einfach von hinten an Funktionalität wegzuschneiden. Ich werde sicherlich die Bestellungsbestätigung (Order Confirmation) nicht mehr implementieren können und ich muss eventuell noch mehr kürzen, was ich aber in den nächsten Tagen entscheiden werde.

4.3.1 Fazit

Ich habe ganz klar die Aufgabenstellung unterschätzt und hinke nun hinter dem Zeitplan her. Zum Glück passiert mir das während der Probe, sodass ich daraus lernen kann und es nächstes Mal besser mache.

4.4 Mittwoch, 07. November 2018

Tag 4 (07.11.2018)			
Phase: Realisieren			
Tätigkeit	**Soll-Zeit (Stunden)**	**Ist-Zeit (Stunden)**	**Bemerkung**
ViewModels fertigstellen	1	1	
Business-Logik mit View und DatabaseClient verknüpfen	5	5	
Testfälle schreiben	1.5	1.5	
Arbeitsjournal führen	1	1	
Total	8.5 Stunden	8.5 Stunden	

Tabelle 8: Arbeitsprotokoll 07.11.2018

Heute hatte ich noch meinen Endspurt in der Phase «Realisieren». Ich habe den ganzen Tag mehrheitlich Business-Logik geschrieben und kam ganz gut voran. Nebenbei hatte ich immer wieder Testfälle erstellt, die die geschriebene Logik abdecken. Wir hatten gegen Ende des Tages noch eine kurze Einleitung zum Thema Präsentation und Fachgespräch. Remo hat uns ein wenig über den Ablauf erzählt und wir wurden einem Datum zugewiesen, für die Präsentation der Probe-IPA.

4.4.1 Erfolge

Ich empfand den heutigen Tag, im Gegensatz zu den bisherigen Arbeitstagen, als eher gemütlich. Ich konnte sehr still und konzentriert Arbeiten und habe mich fast nicht ablenken lassen. Mit dem Stand der Dokumentation bin ich sehr zufrieden. Sie hat soliden Inhalt und sollte die meisten Kriterien der IPA abdecken.

4.4.1 Misserfolge

Wie auch schon im vorherigen Arbeitsjournal beschrieben, konnte ich nicht ganz dem Zeitplan folgen. Es war mir trotz der Massnahme, dass ich eine Ansicht weniger mache, nicht möglich alle Kriterien in der Aufgabenstellung abzuhandeln. Mein logisches Verständnis empfinde ich als relativ stark, weshalb es mir auch einfach fällt Konzepte aufzustellen. Jedoch scheitert es bei mir meistens an der Umsetzung dieser Konzepte. Ich bin mir nie ganz sicher, welchen Datentyp ich nehmen soll oder mit welchem Pattern es jetzt am einfachsten wäre und so weiter. Ich brauche dann immer eine gute Planung, um dies doch noch umsetzen zu können. In dieser Arbeit hatte ich wenig Zeit für die Planung und dies spürte ich nun in der Realisierung.

4.4.1 Code Analyse

Wie im Kapitel «Realisieren» beschrieben, habe ich SonarQube aufgesetzt und eine erste Analyse gestartet. Ich kann die Analyse jedoch kein zweites Mal starten, weshalb ich keine Code Analyse-Resultate darstellen kann. Das nervt mich ziemlich, weil ich extra dafür Testfälle geschrieben habe. Das Problem ist man muss gewisse Command auf der Windows-Shell ausführen, jedoch konnte ich diese nirgends mehr finden, was ziemlich blöd gemacht ist. Ich hätte mir die Befehle irgendwo speichern müssen, oder halt mit einem CI-Server, wie Jenkins, arbeiten, der das automatisch nach den Builds anstösst.

4.4.1 Fazit

Ich muss mich nächstes Mal besser auf die Aufgabenstellung vorbereiten, um alle Controls, Templates und so weiter schon auswendig zu kennen. Ich habe immer wieder Zeit verloren, weil ich noch nachschauen musste und es nicht beim ersten Mal alles geklappt hat.

4.5 Freitag, 09. November 2018

Tag 5 (09.11.2018)			
Phase: (Realisieren) Kontrollieren & Auswerten			
Tätigkeit	**Soll-Zeit (Stunden)**	**Ist-Zeit (Stunden)**	**Bemerkung**
Letzte Anpassungen an Programm	1	1	
Tests durchführen	0.5	0.5	
Testprotokoll & Fazit erstellen	1	1	
Expertengespräch	0.5	0.5	
Reflexion & Fazit schreiben	1.5	1.5	
Kurzfassung schreiben	1	1	
Arbeitsjournal schreiben	1	1	
Projekt abschliessen	2	2	
Total	8.5 Stunden	8.5 Stunden	

Tabelle 9: Arbeitsprotokoll 09.11.2018

Heute ging es an den Endspurt. Ich musst noch ein paar Änderungen an meiner Datenbank vornehmen, damit diese für die Demo bereitsteht. Ich habe mir am Vorabend nochmals die Implementation des Datenbank-Service angeschaut und konnte schonmal gewisse Fehler ausschliessen. Heute Morgen fand ich dann eine Lösung und konnte sie Umsetzen. Des Weiteren habe ich heute vor allem kontrolliert und ausgewertet.

4.5.1 Erfolge

Obwohl ich heute noch recht viel machen musste, konnte ich das Projekt mehr oder weniger erfolgreich abschliessen. Ich konnte relativ schnell die Kapitel Kontrollieren & Auswerten dokumentieren, da ich bereits einige Gedanken dazu gemacht und festgehalten habe. Ich habe mir gewisse Stichwörter während der Arbeit in dem Kapitel notiert, damit ich weiss, auf was ich noch achten muss. Dies hat mir gut geholfen und die Wörter sind nur so aus mir herausgeflossen.

4.5.1 Fazit

Der letzte Tag ist mir eigentlich am besten gelungen. Ich konnte noch sehr vieles abschliessen und bin sehr zufrieden mit meiner Arbeitsleistung. Das Projekt ist nun endlich fertig und ich kann mich wieder ein wenig entspannen.

5 Kurzfassung

In der Kurzfassung wird kurz und bündig das Kapitel «Teil 2: Projekt» zusammengefasst. Es zeigt die wichtigsten Stichpunkte aus diesem Teil und gibt dem Leser einen raschen Überblick der Arbeit.

5.1 Ausgangslage

Torben Dziuk, die Verantwortliche Fachkraft, hat mir, als Aufgabenstellung für die Probe-IPA, eine Aufgabe aus den Schweizermeisterschaften übertragen. Diese Aufgabe wurde dieses Jahr von den Kandidaten gelöst und ist öffentlich zugänglich.
Die Kandidaten an den Schweizermeisterschaften hatten nur einen ganzen Tag für diese Aufgabe Zeit, weshalb wir als Extra folgende Kriterien hinzugenommen haben. Die Kandidaten hatten keine bestimmten Anforderungen bezüglich Code-Prinzipien, noch Unittests für die Einheiten. Wir haben als Kriterien definiert, dass die Applikation im Firmenstandard implementiert werden muss und für die Business-Logik eine anständige Testabdeckung vorhanden sein muss. Der Firmenstandard beinhaltet vor allem die Coding-Regeln von Clean Code. (Developer, 2015)

5.2 Umsetzung

Für die Umsetzung wurde die Projektmanagementmethode «IPERKA» verwendet. Diese ist im Kapitel «Projektorganisation» beschrieben. Als wurde die Aufgabenstellung analysiert. Dazu wurde eine intensive Ist- / Sollanalyse durchgeführt, in der alle Soll-Kriterien dokumentiert wurden. Diese Analyse diente vor allem als Grundlage für die Planung. In der Planungsphase wurden verschiedene Use Cases erstellt, welche die Anwendungsfälle der Applikation festhalten. Aus diesen Use Cases wurden die Abnahmetests für die Applikation erstellt. Es gibt jeweils einen Testfall pro Use Case. Während der Planung wurden ebenfalls einzelne Konzepte erstellt, welche als Leitfaden dienten, und es wurden Architektur-Planungen vorgenommen. Während der Planung wurden vor allem abstrakte Aspekte behandelt. In der Entscheidungsphase wurden danach konkrete Elemente behandelt und dokumentiert.

Die bisherigen Umsetzungsmerkmale dienten der Vorarbeit für die Phase «Realisieren». In dieser Phase wurde hauptsächlich implementiert und Ergebnisse festgehalten. Diese Ergebnisse wurden später in der Kontrollphase hinterfragt und reflektiert. Dazu wurden die Abnahmetests durchgeführt und es wurde ein Testprotokoll erstellt.
In der letzten Phase «Auswerten» wurde die ganze Arbeitswoche reflektiert und Erfolge, wie Misserfolge festgehalten. Ausserdem wurden gelernte Erkenntnisse und Erfahrungen dokumentiert und es wurde ein Ausblick auf die möglichen Erweiterungen gegeben.

5.3 Ergebnis

Das Projekt wurde mit wenigen Ausnahmen vollumfänglich ungesetzt. Es besteht eine funktionierende WPF-Applikation mit Backend in C#. Als zentrale Datenverwaltung wurde eine Datenbank in MSSQL aufgebaut und mit Entity Framework verbunden.

6 Informieren

Das Informieren ist die erste Stufe in der gewählten Projektmethode «IPERKA». In diesem Schritt geht es vor allem um die Auftragsanalyse. Hier werden benötigte Information herausgesucht und zusammengestellt. Dabei werden Kriterien wie Umgebung, Bedingungen, Vorkenntnisse, Ressourcen und Ziel ganz genau hinterfragt. Man versucht in diesem Schritt den Auftrag ganz genau zu untersuchen, damit man bereits offene Fragen abdecken und mögliches Fehlerpotenzial aufweisen kann, um einen groben Überblick zu erhalten. Dieser Schritt dient als Grundlage für das nächste Kapitel «Planung».

6.1 Ist-Analyse

Im Modul 223, welches von Remo Steinmann geleitet wird, muss eine IPA als Vorbereitung erstellt werden. Dazu wurden die Teilnehmer aufgeboten eine Aufgabenstellung zu definieren. Ich habe bei der Atos, explizit bei Torben Dziuk, nachgefragt, ob er eine Aufgabe für mich hätte. Nach gemeinsamen Überlegungen haben wir uns für die Aufgabe aus den ICT-Skills-Schweizermeisterschaften aus diesem Jahr entschieden.
Diese Aufgabe wurde während den Schweizermeisterschaften in einem Tag gelöst, jedoch wurde dabei nur auf die allgemeine Funktionalität geachtet und nicht wirklich auf die Qualitätsmerkmale einer Applikation. Folgende Aspekte wurden bei den Schweizermeisterschaften nicht ausführlich bis gar nicht behandelt:

- Testing
  - Unit- / Modul Tests
  - RegressionTests
- Transaktionssicherheit
- Code Quality (z.B. mit SonarQube)
- Clean Code Prinzipien

6.2 Soll-Analyse

Die Soll-Analyse ist eine genauere Analyse der Anforderungen und zeigt den Zustand, welcher als Ziel definiert wurde. In diesem Kapitel werde ich zur Veranschaulichung die Screenshots, aus der Aufgabenstellung von ICT-Skills, als Wire-Frames für meine Benutzeroberfläche nutzen.

6.2.1 Frontend

Soll-Analyse der Frontend-Implementation.

Menu
Das Menu ist die Startseite der Applikation und dient sozusagen als Navigation für den Benutzer. Über das Menu kann man die unten beschriebenen Funktionalitäten auswählen und deren Benutzeroberfläche öffnen.

- Edit Customer Details
- Muesli Mixer
- My Muesli Mixes
- Order (Deaktiviert, wenn keine Muesli Mixes erfasst wurden)
- Exit (Applikationsaustritt)

Abbildung 2: Soll-Analyse Menu

Abbildung 3: Soll-Analyse Edit Customer Details

Edit Customer Details
Diese Benutzeransicht wird für das editieren eines Kunden benutzt und kann vom Menu aus geöffnet werden. Folgende Daten müssen für den Benutzer editierbar sein: Name, Adresse, Postleitzahl, Stadt, Land, Telefonnummer und email. Dafür müssen angemessene UI-Controls implementiert werden.
Beim Klicken auf den Save-Knopf müssen die vorhandenen Einträge in der Datenbank abgespeichert werden => Bei Missachtung der Validation Regel muss eine Fehlermeldung erstellt werden, in welcher der Benutzer über die fehlerhaften Felder informiert wird.
Beim Klicken auf den «Back to menu»-Knopf wird der Benutzer zum Menu zurückgeleitet.
Damit fehlerhafte Daten in der Datenbank verhindert werden, müssen folgende Validierungen implementiert werden.

Feld	Validationsregel
Name	Länge => mindestens 5 Zeichen
Adresse	Länge => mindestens 5 Zeichen
Postleitzahl	Länge => mindestens 4 Zeichen, nur numerische Werte
Stadt	Länge => mindestens 2 Zeichen
Land	Es muss ein Land von einer vordefinierten Liste selektiert werden.
Telefonnummer	Länge => mindestens 10 Zeichen, nur numerische Werte, Leerzeichen oder «+»
Email	Muster: x@y.z X => mindestens 1 Zeichen Y => mindestens 3 Zeichen Z => 2-3 Zeichen (nur [a-zA-z])

Tabelle 10: Validationsregeln Customer Details

Muesli Mixer
Diese Ansicht ist vom Menu aufrufbar und ist sozusagen der Kern der Applikation. Man kann auf dieser Seite sein individualisiertes Muesli erstellen. Ein Muesli-Mix muss immer aus einer Basis-Zutat (Kategorie Basics) bestehen und kann bis zu 12 zusätzliche Zutaten beinhalten. Als erstes muss immer die Basis-Zutat ausgewählt werden und erst dann kann man aus einer der folgenden Kategorien auswählen: Cerealien, Früchte, Nüsse & Co, Schokolade und Spezialitäten.

Das Gesamtgewicht von einem Muesli ist immer 600g. Die Basis-Zutat wird als «Füller» genutzt und wird automatisch berechnet (600g – (Totalgewicht der zusätzlichen Zutaten)).

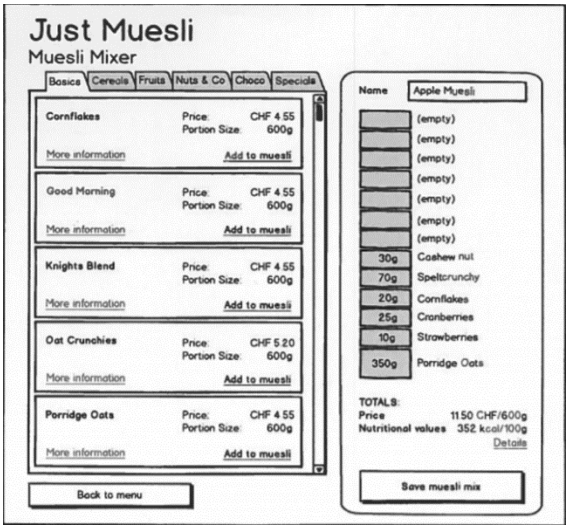


Abbildung 4: Soll-Analyse Muesli Mixer

Components list (Linke Seite von Muesli Mixer)
Auf der linken Seite vom Muesli Mixer kann man die verschiedenen Zutaten auswählen. Es muss für jede Kategorie ein Tab erstellt werden, welcher eine Liste seiner Zutaten enthält und anzeigt. Diese Liste sollte nach dem Namen sortiert sein. In der Liste muss für jede Zutat folgendes ersichtlich sein: Name, Preis, Gewicht einer Portion.
Der Benutzer muss diese Zutaten per Klick auf «Add to Muesli» in sein Muesli hinzufügen können. Wenn bereits 12 Zutaten hinzugefügt wurden, muss eine Fehlermeldung angezeigt werden. Als Erstes muss immer eine Basis-Zutat ausgewählt werden und es kann nur eine einzige ausgewählt sein. Wenn der User eine zusätzliche Basis-Zutat hinzufügen möchte, wird die Alte ersetzt.
Beim Klick auf den «Back to Menu»-Knopf wird der Benutzer zum Menu zurückgeleitet.
Beim Klick auf den «More Information»-Knopf muss ein Popup erscheinen, welches die Zutaten, Nährstoffe und den Energiegehalt einer Komponente anzeigt. Für die Berechnung des Energiegehaltes wird folgende Kalkulation benutzt:

Abbildung 5: Soll-Analyse Components List

(1 Kcal = 4.184 KJ)

Nährstoff	Kilocalorien pro gram (Kcal/g)	Kilojoules per gram (KJ/g)
Kohlenhydrate	4.1	17.2
Proteine	4.1	17.2
Fette	9.3	38.9

Tabelle 11: Nährstoff-Kalkulation

Current muesli mix list (Rechte Seite von Muesli Mixer)
Es muss auf der rechten Seite von Muesli Mixer ein Stapel mit den gewählten Zutaten existieren. Die Basis-Zutat muss immer zuunterst sein. Es soll eine Fehlermeldung erscheinen, wenn das gesamt Muesli schwerer als 600g ist. Der Gesamtpreis des Muesli (In CHF pro 600g) und dessen Nährstoffwerte (In kcal pro 100g) müssen kalkuliert und angezeigt werden. Für diese Berechnung und Anzeige muss folgendes beachtet werden:

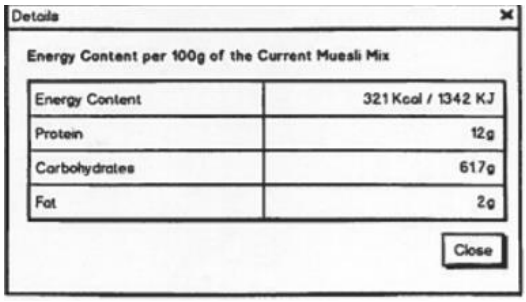


Abbildung 6: Soll-Analyse Current Muesli Mix list

- Der Preis muss auf die zweite Dezimalstelle gerundet sein (0.01).
- Der Benutzer kann eine Zutat entfernen, wenn er auf sie klickt. Da sollte eine Bestätigung für den Benutzer erforderlich sein (Popup - yes/no).
- Es soll für den Benutzer möglich sein, einen Namen für sein eigenes Muesli zu definieren.
- Über den «Save Muesli»-Button kann das Muesli in der Datenbank gespeichert werden.
    - Eine Fehlermeldung muss angezeigt werden, wenn folgendes nicht erfüllt wurde
        - Muesli hat einen Namen.
        - Eine Basis-Zutat ist selektiert

- Für die gesundheitsbewussten Benutzer soll es für jede Zutat eine eigene Detail-Seite geben auf der die Nährstoffgehalte angezeigt werden.

My Muesli Mixes
Auf dieser Ansicht kann der Benutzer seine eigenen Muesli-Kreationen ansehen. Diese Ansicht kann man vom Menu aus aufrufen. Alle Muesli sollen mit Name, Preis und Erstelldatum angezeigt werden. Die Tabelle soll nach dem Namen sortiert sein, doch es soll für den Benutzer möglich sein auf anderen Spalten zu sortieren. Der Edit-Knopf öffnet den Muesli Mixer mit dem ausgewählten Muesli als Inhalt. Der Benutzer kann das Muesli löschen, wenn er auf den Delete-Knopf drückt. Dabei wird eine Bestätigungsmeldung angezeigt, wo der Benutzer noch bestätigen muss, dass er das Muesli wirklich löschen möchte. Wenn Edit oder Delete geklickt wird ohne, dass ein Muesli selektiert wird, soll eine Fehlermeldung angezeigt werden. Der «Back to menu»-Knopf soll den Benutzer zurück aufs Menu leiten.

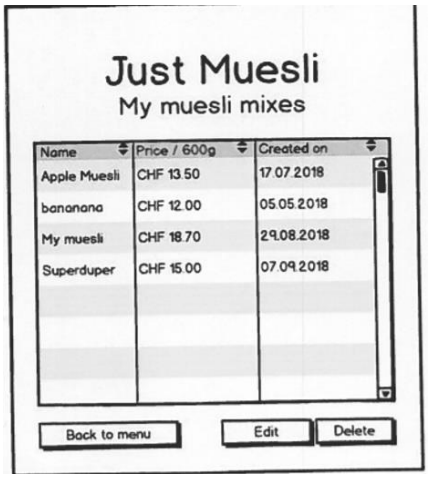


Abbildung 7: Soll-Analyse My Muesli Mixes

Order
Diese Ansicht wird genutzt, um bestehende Muesli zu bestellen. Sie ist aufrufbar über das Menu.

M*uesli Mix list (linke Seite von My Muesli Mixes)*
Alle gespeicherten Muesli sollen mit dem Namen und Preis angezeigt (Nach Preis abwärts sortiert). Sobald ein Müesli eine Anzahl von Bestellungen (Quantity) zugewiesen bekommt, soll der Order-Knopf verfügbar sein. Der Benutzer soll die Möglichkeit haben eine XXL Version seines Muesli zu bestellen (2'400g / Preis 4x). Jeder Eintrag soll am Schluss den Totalpreis für ein Muesli anzeigen (Angezeigter Preis * Anzahl Bestellungen). Der «Back to Menu»-Knopf leitet den Benutzer auf das Menu zurück.

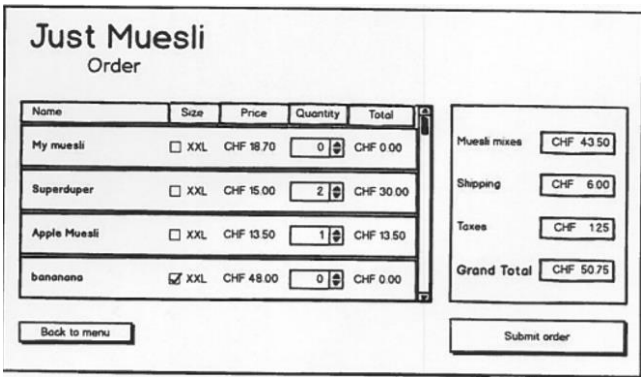


Abbildung 8: Soll-Analyse Order

Costs (rechte Seite von My Muesli Mixes)
Der angezeigt Bestellpreis soll aus den verschiedenen Zwischenpreisen entstehen. Die Lieferkosten innerhalb der Schweiz sind immer 6.- CHF und ausserhalb immer 8.- CHF. Wenn der Bestellpreis höher als 50.- CHF ist, sollen keine Lieferkosten angezeigt werden.
Wenn die Bestellung ausserhalb der Schweiz liegt, sollen keine Steuern berechnet werden. Für inländische soll eine Steuer von 2.5 % (gerundet auf nächste 0.05 Stelle). Die Anzeige der Kosten muss Read-Only sein. Wenn der Benutzer auf «Submit order» klickt, soll eine Bestellungsbestätigung angezeigt werden (siehe unten).

Order Confirmation
Die Bestellungsbestätigung soll als eine Textdatei generiert werden und soll folgendermassen formatiert sein

```
Dear [Name]

Thank you very much for using JustMuesli to order your personal Muesli

Order overview

Item                    Size          Price    Quantity          Total
-------------------------------------------------------------------------
Superduper              Normal        15.00           2          30.00
Apple Muesli            Normal        13.50           1          13.50
bananana                XXL           48.00           1          48.00
-------------------------------------------------------------------------
Total Muesli mixes                                                91.50
Shipping                                                           0.00
Taxes                                                              2.30
=========================================================================
Grand Total                                                       93.80
=========================================================================


The ordered items will be sent to the following address:
[Name]
[Address]
[Zip] [City]
[Country]
```

Abbildung 9: Soll-Analyse Order Confirmation

Diese Datei soll im Arbeitsverzeichnis der Applikation abgelegt werden und der Filename soll «orderconfirmation.txt» sein (Existierende Datei soll überschrieben werden). Die Platzhalter in den Klammern (z.B. [Name]) muss mit den Kundendaten ersetzt werden. Die Tabelle muss die bestellten Müslis enthalten (Name, Preis, Grösse, Anzahl und eigenes Total), Kosten der Müslis, Lieferkosten, Steuern und den Totalpreis der Bestellung. Die Spalten der Tabelle müssen folgendermassen formatier werden.

Spalte	Grösse (Zeichenanzahl)	Format	Ausrichtung
Item	25	Text	Links
Size	7	Text	Links
Price	10	Nummer mit 2 Kommastellen	Rechts
Quantity	10	Nummer ohne Kommastellen	Rechts
Total	14	Nummer mit 2 Kommastellen	Rechts

Tabelle 12: Bestellungsbestätigung Formatierung

Wir können dabei annehmen, dass kein Wert die definierte Grösse überschreitet.

6.2.1 Backend

Alle Frontend-Schnittstellen interagieren mit der Backend-Implementation. Das Backend ist dafür zuständig die erforderlichen Daten in der Datenbank zu persistieren.

Datenbank
Mit dem Entity Framework wird die Datenbank erstellt, erweitert, und manipuliert. Im Hintergrund steckt eine MSSQL-Datenbank. Über die Benutzerschnittstelle wird das backend

angesteuert und Entity Framework kommuniziert die gewünschten Änderungen mit der Datenbank.

6.3 Systemübersicht

In diesem Projekt gibt es eigentlich nur zwei Systeme welche miteinander kommunizieren. Es gibt die Client-Applikation welche jeweils auf die MSSQL-Datenbank zugreift. Dazwischen sitzt das Entity Framework, welches als ORM (Object-relational Mapper) fungiert und die Client-Applikation bei Datenbankabfragen unterstützt.

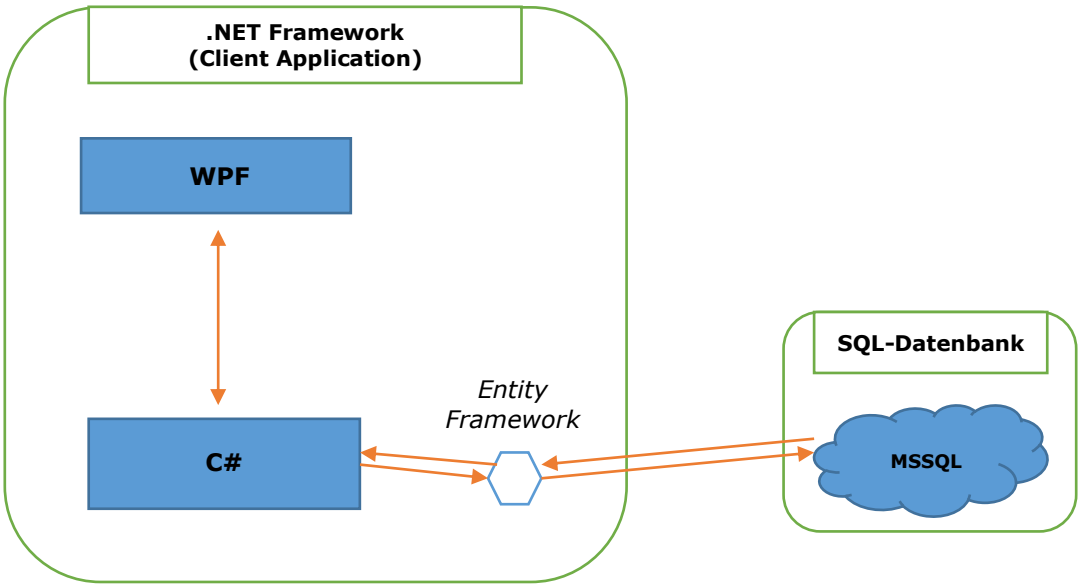


Abbildung 10: Systemübersicht

Wie man in der obigen Abbildung sehen kann, wird innerhalb des .NET-Framework mit C# und WPF gearbeitet. WPF wird für das Frontend, respektive für die Benutzeroberfläche, genutzt und C# für die Business-Logik und Daten-Aufbereitung der Benutzeroberfläche.

7 Planen

Im zweiten Schritt der IPERKA-Methode werden nun die möglichen Lösungsvarianten und das Vorgehen ausgearbeitet. Es werden einzelne Arbeitsschritte geplant, um einen gesamten Arbeitsablauf zu erstellen. Im Arbeitsablauf werden Hilfsmittel und Werkzeuge ermittelt, sowie der Zeitbedarf für die Arbeitsschritte geschätzt. Zudem werden Qualitäts-Kriterien für den Schritt «Kontrollieren» gesammelt. (VSSM, 2014)

7.1 Use Cases

In diesem Kapitel beschreibe ich die verschiedenen Anwendungsfälle anhand von Use Cases. Bei Use Cases ist es nicht nur wichtig, dass man darstellt, welche Funktionalität von wem aufgerufen werden kann, sondern auch was nach dieser Aktion passiert und ob es Vorbedingungen gibt.

In meiner Applikation gibt es eigentlich nur eine Rolle – Den Kunden. Dieser interagiert mit der ganzen Applikation. Er hat keine Einschränkungen und ist vollkommen dazu berechtigt jede Funktionalität aufzurufen.

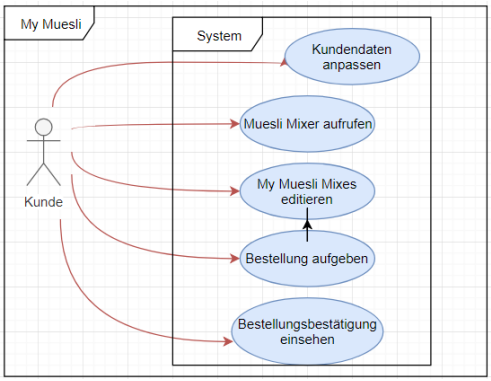


Abbildung 11: Use Case Diagram Kunde

Use Case 1 – Kundendaten anpassen	
Beschreibung	Als Kunde möchte ich meine Daten anpassen können
Konsequenz	Man wird auf die Ansicht «Edit Customer Details» weitergeleitet und kann seine Benutzer Daten anpassen und speichern.
Ablauf	1. Auf dem Menu sein. 2. Knopf «Edit Customer Details» klicken. 3. Seite «Edit Customer Details» wird angezeigt. 4. Daten anpassen. 5. Auf den Save-Knopf drücken.

Tabelle 13: Use Case Kundendaten anpassen

Use Case 2 – Muesli Mixer aufrufen	
Beschreibung	Als Kunde möchte ich ein individualisiertes Muesli erstellen können, wobei ich aus einer Vielzahl von Zutaten auswählen kann.
Konsequenz	Der Muesli Mixer wird geöffnet und man hat die Möglichkeit ein neues Muesli zu konfigurieren und abzuspeichern. Nebenbei sieht man immer die verschiedenen Nährstoff-/ und Energiegehalte der einzelnen oder des gesamten Mueslis.
Ablauf	1. Auf dem Menu sein. 2. Knopf «Muesli Mixer» klicken. 3. Seite «Muesli Mixer» wird angezeigt. 4. Zutaten auswählen und hinzufügen. 5. Energie-/ und Nährstoffgehalte anzeigen lassen. 6. Muesli speichern.

Tabelle 14: Use Case Muesli Mixer aufrufen

Use Case 3 – My Muesli Mixes editieren	
Beschreibung	Als Kunde möchte ich die Möglichkeit haben meine bereits erstellten Müeslis zu bearbeiten.
Vorbedingung	Es müssen bereits Muesli erfasst worden sein.
Konsequenz	Das entsprechend ausgewählte Muesli wird als Inhalt im Muesli Mixer geöffnet.
Ablauf	1. Auf dem Menu sein. 2. Knopf «My Muesli Mixes» klicken. 3. Seite «My Muesli Mixes» wird angezeigt. 4. Eigenes Muesli auswählen. 5. Auf den Edit-Knopf klicken. 6. Seite «Muesli Mixer» wird mit gewähltem Inhalt geöffnet. 7. Muesli bearbeiten. 8. Mit dem Save-Knopf das Muesli speichern.

Tabelle 15: Use Case My Muesli Mixes editieren

Use Case 4 – Bestellung aufgeben	
Beschreibung	Als Kunde möchte ich meine bereits erfassten Mueslis bestellen können. Ich möchte auswählen zwischen XXL und normal Portion und ich möchte eine Anzahl Bestellungen eingeben können.
Vorbedingung	Es müssen bereits Muesli erfasst worden sein.
Konsequenz	Die definierte Bestellung wird erfasst und eine Bestellungsbestätigung wird erstellt.
Ablauf	1. Auf dem Menu sein. 2. Knopf «Order» klicken. 3. Seite «Order» wird angezeigt. 4. Anzahl Bestellungen & Grösse der Muesli definieren. 5. Auf «Submit order» klicken.

Tabelle 16: Use Case: Bestellung aufgeben

Use Case 5 – Bestellungsbestätigung einsehen	
Beschreibung	Als Kunde möchte ich, nachdem ich die Bestellung aufgegeben habe, meine Bestellung nochmals einsehen.
Vorbedingung	Es muss eine Bestellung aufgegeben worden sein.
Konsequenz	Bestellbestätigung wird geöffnet und angezeigt.
Ablauf	1. Im Arbeitsverzeichnis der Applikation nach dem File «orderconfirmation.txt» und öffnen. 2. Bestellbestätigung einsehen.

Tabelle 17: Use Case Bestellungsbestätigung einsehen

7.2 Testkonzept

Das Testkonzept wird in der Planung erstellt und wird danach in der Phase «Kontrollieren» wiedergenutzt. Im Testkonzept werden alle gewünschten Funktionalitäten festgehalten, damit später sichergestellt werden, dass die Vorgaben eingehalten wurden. Um dieses Prozedere zu ermöglichen werden Testfälle erstellt.

7.3 Unit- / Modultests

Als erste Qualitätssicherung werden verschiedene Unit- / oder auch Modultests erfasst. Diese sind automatisiert und decken entweder einzelne Funktionen oder ganze Funktionsketten (Module) ab. Da die Zeit zu knapp ist, um für Alles Test zu schreiben, wird vor allem auf Kernfunktionalität geachtet. Mit dem Tool SonarQube kann auf die Code Coverage geachtet werden. Ich versuche mit den Unit-/ Modultest eine Code Coverage von 60 % zu erreichen. Sobald die Phase «Realisieren» beginnt, wird in jedem Arbeitsjournal ein kurzer Beitrag zu den Ergebnissen von SonarQube erstellt.
Die automatisierten Tests werden vor jedem Push auf das Git-Repository ausgeführt, um sicherzustellen, dass keine Funktionalität, welche kaputtgegangen ist, auf das «Backup» kommt.

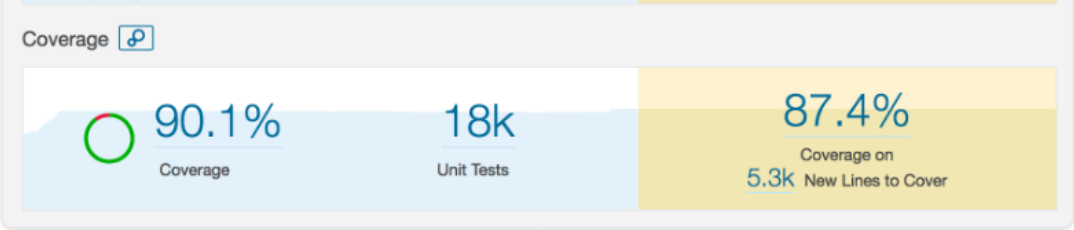


Abbildung 12: Code Coverage Beispiel SonarQube

7.4 Abnahmetests

Im zweiten Schritt für die Qualitätssicherung werden manuelle Abnahmetests erstellt. Unter einem Abnahmetest versteht man in der Softwaretechnik die Testfälle, welche gebraucht werden um sicherzustellen, dass die Muss-Kriterien einer Applikation erfüllt sind.

7.5 Testfälle

Anbei werden verschiedene manuelle Testfälle erstellt, welche mit den Anforderungen in den Use Cases korrelieren. Die Testfälle werden nach den Use Cases benannt (Nummer + Bezeichnung).

(Use Case 1) Nr.	Vorbedingung	Testschritte	Erwartetes Ergebnis
1.1	Seite «Edit Customer Details» ist geöffnet.	1. Vollständiger Kunde erfassen mit validen Datensätzen 2. Save-Knopf drücken	Neuer Kunde wird in der Datenbank gespeichert.
1.2	Siehe oben	1. Vollständiger Kunde mit Name «hans» erfassen. 2. Save-Knopf drücken	Fehlermeldung erscheint. Feld wird als falsch markiert.
1.3	Siehe oben	1. Vollständiger Kunde mit Adresse «hier» erfassen. 2. Save-Knopf drücken	Fehlermeldung erscheint. Feld wird als falsch markiert.

1.4	Siehe oben	1. Vollständiger Kunde mit Postleitzahl «3e» erfassen. 2. Save-Knopf drücken	Fehlermeldung erscheint. Feld wird als falsch markiert.
1.5	Siehe oben	1. Vollständiger Kunde mit Stadt «b» erfassen. 2. Save-Knopf drücken	Fehlermeldung erscheint. Feld wird als falsch markiert.
1.6	Siehe oben	1. Vollständiger Kunde mit Handnummer «sdfsa34234» erfassen. 2. Save-Knopf drücken	Fehlermeldung erscheint. Feld wird als falsch markiert.
1.7	Siehe oben	1. Vollständiger Kunde mit Email «a.a.a@d» erfassen. 2. Save-Knopf drücken	Fehlermeldung erscheint. Feld wird als falsch markiert.

Tabelle 18: Testfälle 1

(Use Case 2) Nr.	Vorbedingung	Testschritte	Erwartetes Ergebnis
2.1	Seite «Muesli Mixer» ist geöffnet.	1. Muesli erfassen a. Basiszutat b. Zusätzliche Zutaten c. Name 2. Auf «Save Muesli mix» klicken	Muesli wird in Datenbank gespeichert.
2.2	Siehe oben	1. Muesli ohne Basiszutat erfassen 2. Auf «Save Muesli mix» klicken	Fehlermeldung wird angezeigt
2.3	Siehe oben	1. Muesli mit mehr als 12 Zutaten erfassen.	Fehlermeldung wird angezeigt
2.4	Siehe oben	1. Muesli mit mehr als 600g Gesamtgewicht erfassen.	Fehlermeldung wird angezeigt
2.5	Siehe oben	1. Mit Muesli Mixer interagieren.	Preis & Nährstoff-/Energiegehalt wird neu berechnet und angezeigt.

2.6	Siehe oben	1. Auf «Details» klicken.	Detail Seite wird angezeigt und zeigt Nährstoff-/Energiegehalt für das Ganze Muesli
2.7	Siehe oben	1. Auf «More Information» klicken.	«More Information»-Seite wird angezeigt und zeigt Nährstoff-/Energiegehalt & Unterzutaten für eine zutat
2.8	Zutat muss in Muesli Mixer hinzugefügt sein	1. Klick auf Zutat in Zutatenliste	Zutat wird aus Liste entfernt

Tabelle 19: Testfälle 2

(Use Case 3) Nr.	Vorbedingung	Testschritte	Erwartetes Ergebnis
3.1	Seite «My muesli mixes» muss geöffnet sein. Ein Muesli muss bereits erstellt worden sein.	1. Muesli auswählen 2. Auf Edit klicken	Muesli Mixer mit gewähltem Muesli als Inhalt wird geöffnet.
3.2	Siehe oben	1. Muesli Auswählen 2. Auf Delete klicken	Muesli wird entfernt

Tabelle 20: Testfälle 3

(Use Case 4) Nr.	Vorbedingung	Testschritte	Erwartetes Ergebnis
4.1	Seite «Order» muss geöffnet sein. Ein Muesli muss bereits erstellt worden sein.	1. Muesli auswählen 2. Anzahl Bestellung auf 2 Setzen	Subtotal und Total soll angezeigt werden. Lieferkosten und Steuern sollen angezeigt werden.
4.2	Siehe oben	1. Muesli als XXL auswählen	Preis muss um 4-faches steigen
4.3	Siehe oben	1. Keine Anzahl für Bestellung eingeben	«Submit order»-Knopf soll deaktiviert sein
4.4	Siehe oben	1. Muesli Anzahl auf 2 setzen 2. «Submit order» klicken	Bestellungsbestätigung wird erstellt und im Applikationsverzeichnis hinterlegt.

Tabelle 21:Testfälle 4

(Use Case 5) Nr.	Vorbedingung	Testschritte	Erwartetes Ergebnis
5.1	Muesli muss, wie unter 4.4 beschrieben,	1. Ins Applikationsverzeichnis gehen.	Bestellbestätigung wird angezeigt in richtiger Formatierung.

	ausgewählt und bestellt werden	1. Datei «orderconfirmation.txt» öffnen	

Tabelle 22: Testfälle 5

7.6 Software-Architektur

In diesem Kapitel werden alle Themen bezüglich der allgemeinen Software-Architektur behandelt. Es werden die geplanten Komponenten definiert, ohne sie zu konkretisieren. Ich werde die funktionalen Verantwortungen der einzelnen Komponenten genauer beschreiben, jedoch ihre konkrete Implementation offenlassen, damit ich in der Phase «Realisieren» noch Anpassungsmöglichkeiten habe.

Komponente	Beschreibung	Abhängigkeiten
DatabaseService	Verknüpft die Models mit der Datenbank. Auf diesem Service werden alle Datenbankabfragen aufgerufen.	Entity Framework
LoggingService	Dieser Service wird von praktisch überall in der Applikation genutzt. Er empfängt die Logging-Einträge und leitet sie ans Log4Net-Framework weiter.	Log4Net
Models	Korrespondieren mit Datenbanktabellen. Behälter für die Daten	Keine Spezialitäten
ViewModels	Verbindung zwischen Models und View. Datenaufbereitung für Benutzerschnittstelle	Keine Spezialitäten
Views	Konkrete XAML Implementation mit Anknüpfung an ViewModels	Keine Spezialitäten
DI-Service	Wrapper für IOC-Container. Zuständig für Dependency Injection.	Keine Spezialitäten

Tabelle 23:Software-Komponenten

7.7 Klassendiagramm

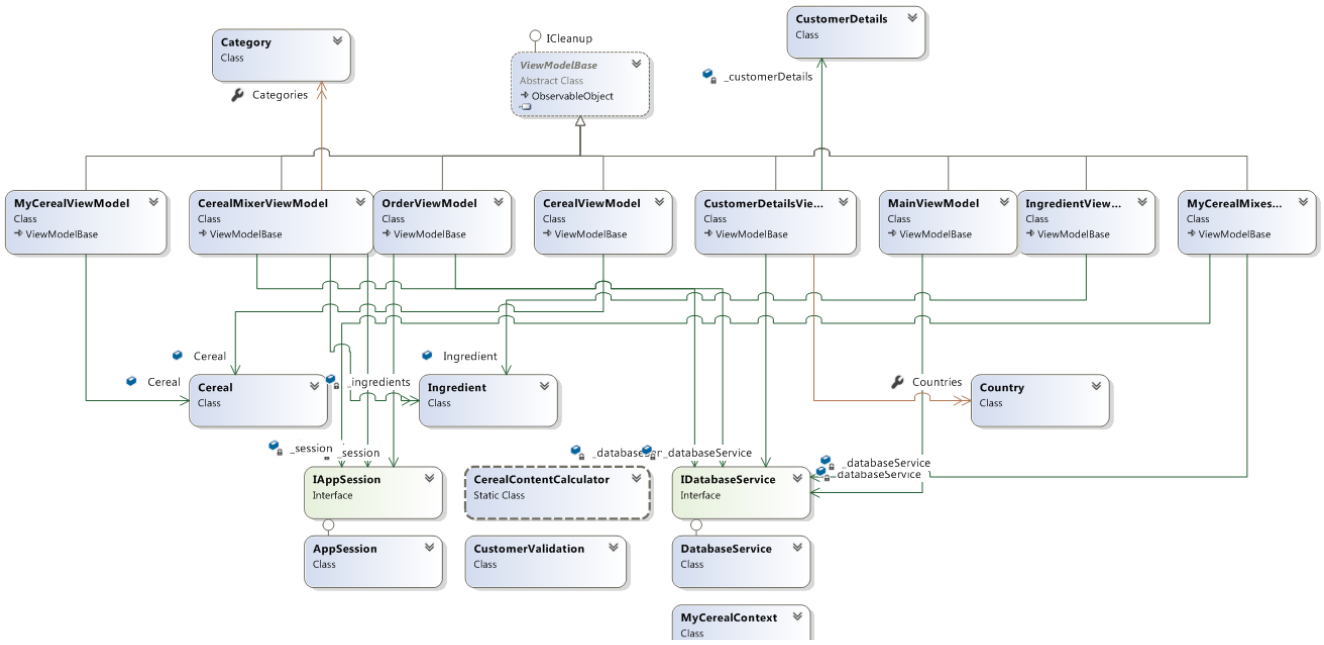


Abbildung 13: Klassendiagramm von Visual Studio

7.8 Datenbank-Architektur

Unter diesem Kapitel wird die Struktur der Datenbank festgelegt. Die Struktur wurde im ersten Schritt auf einem Notizblatt skizziert und wurde nach mehreren kleinen Verbesserungen in Draw.io als erstes ERD (Ohne Datentypen) abgebildet. Die Version von Draw.io wird für die Planung genutzt. Es wird jedoch nach der Datenbank-Implementation ein ERD generiert und ebenfalls unter diesem Kapitel beigefügt.

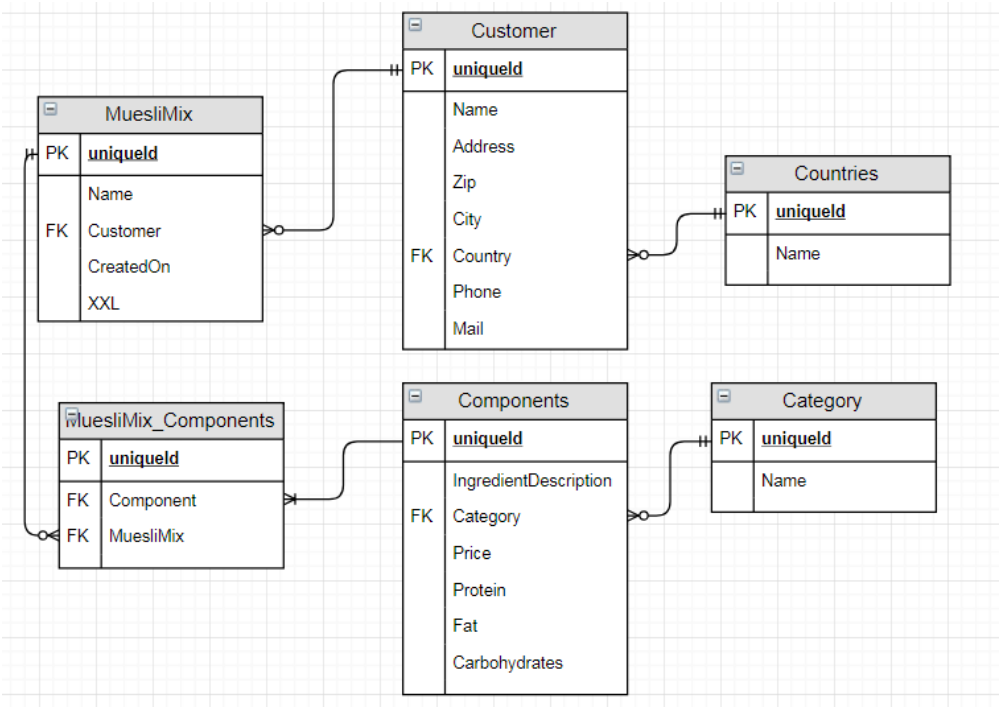


Abbildung 14: Erstes ERD mit Draw.io

7.9 ERD

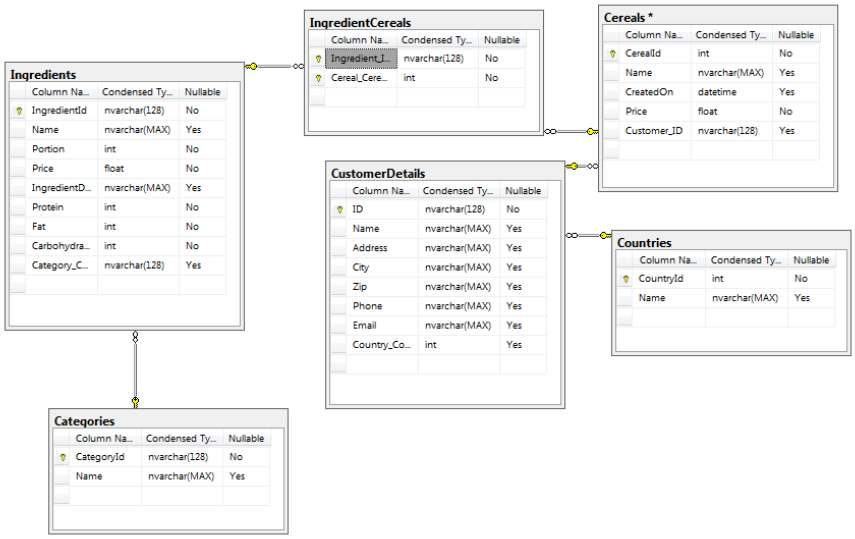


Abbildung 15: ERD von MSSQL Management Studio

7.10 Logging-Konzept

Für das Verständnis des Entwicklers und des Benutzers werden Fehlermeldungen oder sonstige Ereignisse in einer Log-Datei festgehalten.
Damit der Anwender dieser Log-Datei nicht ewig in unnötigen Informationen suchen muss, gibt es zwei verschiedene Log-Dateien.

Datei	Konfiguration	Inhalt
Error_Log.txt	Dieses File wird im Applikationsverzeichnis abgelegt und wird für jeden Tag neu erstellt. Es kann maximal bis zu 100 MB gross werden, bevor eine weitere Datei erstellt wird.	Alle Ereignisse mit Log-Level höher als «Warn» (Error & Fatal).
Log.txt	Dieses File wird im Applikationsverzeichnis abgelegt und wird nur neu erzeugt, wenn die maximal Grösse von 100 MB erreicht wurde.	Beinhaltet alle Ereignisse.

Tabelle 24: Log-Dateien

Log-Eintrag
Ein Log-Eintrag soll folgendermassen aufgebaut werden:
[Datum_Zeit] [Log-Level] [Methode] [Nachricht]

8 Entscheiden

Damit man mit dem dritten Schritt von IPERKA beginnen kann, müssen zuerst die Anforderungen aus dem zweiten Schritt «Planen» erfüllt sein. Nach der Planung müssen die verschiedenen Lösungsvarianten gegenübergestellt werden und eine davon ausgewählt werden. Hierbei werden verschiedene Faktoren miteinander abgeglichen und ausgewertet. (VSSM, 2014)

8.1 Architekturkonzept

Ich arbeite seit 2 Jahren mit C# und entwickle seit je her mit dem Entwurfsmuster «MVVM» (Model View ViewModel). Anders als bei MVC (Model View Controller) trennt man die Darstellung von der Logik der Benutzerschnittstelle. Dieses Konzept zielt, unter anderem, auf WPF (Windows Presentation Foundation), da ein Datenbindungsmechanismus erforderlich ist. Wegen meiner Erfahrung in diesem Bereich und der Kompatibilität zu WPF, arbeite ich nach MVVM in meiner Applikation. (Wikipedia, 2018)

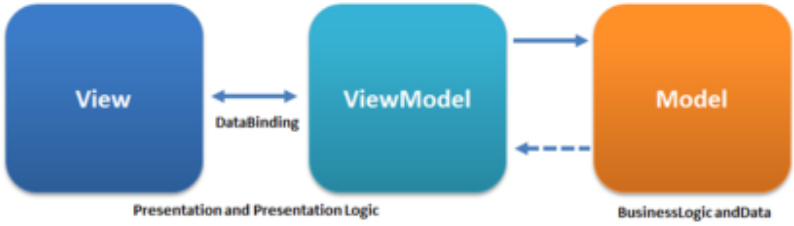


Abbildung 16: MVVM

8.2 Tools und Frameworks

Da ich bereits in der Aufgabenstellung meinen Technologie-Stack vorgestellt habe, gibt es nicht mehr viel was ich wirklich entscheiden kann. Anbei werde ich trotzdem noch kurz die Entscheidungen zu den verwendeten Tools und Frameworks aufzeigen.

Tool / Framework	Alternative	Begründung
Entity Framework	Datenbank mit vordefinierten SqlCommands erstellen und warten.	Das Entity Framework ist sozusagen ein Wrapper für alle SqlCommand, SqlReader, SqlWriter Klassen etc. EF vereinfacht die Arbeit und spart extrem viel Zeit.
Log4Net	NLog	Ich arbeite seit 2 Jahren mit Log4Net. Ich kenne mich viel besser aus damit und habe Praxiserfahrung, was ich bei NLog nicht behaupten kann.
MSSQL	SQLite	Ich wollte eigentlich zuerst mit SQLite arbeiten, da aber die SQLite-Anbindung an das EF nicht wirklich optimal ist, habe ich mich für die altbewährte MSSQL-Datenbank entschieden

Tabelle 25: Tool & Framework Entscheide

8.3 Datenbankentscheidungen

Die Datenbank ist für jedes Projekt und für jede Applikation unterschiedlich. Es gibt gewisse Richtlinien welche man befolgen muss, doch es gibt in der Konkretisierung kein richtig oder falsch. Beifügend habe ich meine Entscheidungen bezüglich Datenbank aufgelistet.

Datenbank-Implementation

Bei Entity Framework gibt es verschiedenen Möglichkeiten, wie man die Datenbank, respektive die Datenbankverbindung, realisiert. Die zwei meist gebrauchten Methoden sind Code-First und Database-First. Wie eigentlich schon in den beiden Namen ersichtlich, wird bei der ersten Methode aus dem Code eine Datenbank generiert und bei der zweiten wird zuerst die Datenbank erstellt und danach der Code.
Bei Code-First ist der Vorteil, dass man mehr Kontrolle über die Struktur der Datenbank hat und bei Database-First ist das Problem, dass man zum Teil Schwierigkeiten hat, die Datenbank-Struktur anzupassen und den Code neu daraus zu generieren. Database-First wird oft verwendet, wenn man mit einem Legacy-System arbeitet, da man dort meistens gewisse Sicherheitsschritte durchführen muss, was bei Code-First die Migrationen unterbinden würde. Da ich nicht mit einem Legacy-System arbeite und mir die Code-First Variante besser liegt, arbeite ich mit der Code-First-Methode.

Tabellenentscheidungen

Unter dem Kapitel «Datenbank-Architektur» habe ich eine erste Skizze für die Tabellenanordnung erstellt. Ich versuch während der Arbeit mich an diese Struktur zu halten, doch bin offen für Erweiterungen oder Anpassungen. Eine wichtige Entscheidung welche ich evaluieren musste ist die Auslagerung der «Countries»-Tabelle.
Wenn ich die Tabelle auslagere, habe ich sozusagen eine vordefinierte Liste, welche ich unter Kontrolle habe. Jedoch muss ich diese Liste immer auf dem neusten Stand halten, da sich Ländernamen ändern können.
Hätte ich die Tabelle nicht ausgelagert und ein Feld in der Customer-Tabelle hinzugefügt, hätte ich den Vorteil, dass jeder Benutzer selber sein Land eingeben kann. Somit muss ich meine Länderdaten nicht jährlich anpassen. Jedoch habe ich dann die Gefahr, dass in meiner Datenbank nachher korrupte Datensätze vorhanden sind, weil ein Benutzer dies falsch erfassen könnte.
Da ich jedoch, aus der Aufgabenstellung vorgegeben, vorgegeben habe, dass die Länder aus einer vorgefertigte Listen stammen, muss ich diese Tabelle auslagern.

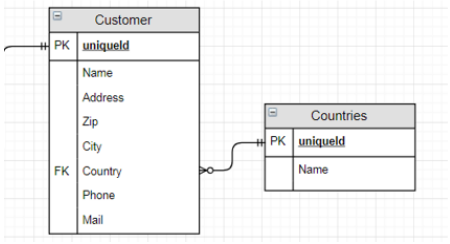


Abbildung 17: CountryTabelle ausgelagert

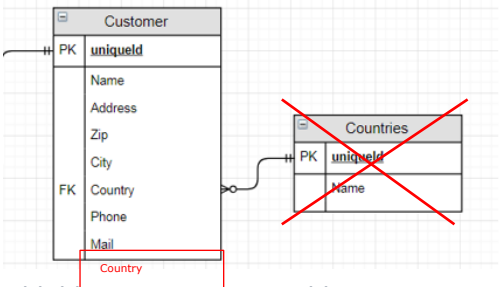


Abbildung 18:Country Feld in Customer

8.4 Benötigte Abhängigkeiten

Wenn man sich in der Objekt-Orientierten Programmierwelt befindet, stellt sich immer wieder die Frage, wie man die Abhängigkeiten eines Moduls, einer Klasse oder eines Service bereitstellt. Es gibt zwei Möglichkeiten die für mich in Frage kommen:

- Ich könnte normal wie üblich die Abhängigkeiten einer Klasse einfach mitübergeben. Somit müsste ich mich nicht wirklich um ein bestimmtes Konzept kümmern und kann mich auf die Logik konzentrieren. Hier laufe ich aber auf folgende Probleme: Tiefe Kohäsion, schlechte Testbarkeit, aufwändige Anpassungen.

- Als zweite Möglichkeit, könnte ich mit einem sogenannten IoC-Container arbeiten, welcher für mich die Abhängigkeiten auflöst und bereitstellt (Dependency Injection). Hierbei muss ich jedoch ein bestimmtes Konzept befolgen und bin eventuell auf weitere externe Abhängigkeiten angewiesen.

Ohne IoC-Container:

```
Public class MyClass
{
   IMyInterface _myInterface;

   Public MyClass ()
   {
      _myInterface = new InterfaceImplementation ();
   }
}
```

Abbildung 20: Ohne Ioc Beispiel

Mit IoC-Container:

```
Public class MyClass
{
   IMyInterface _myInterface;

   Public MyClass (IMyInterface interface)
   {
      _myInterface = myService;
   }
}
```

Abbildung 19: Mit IoC Beispiel

Eine konkrete Implementation des Interfaces wird vom IoC aufgelöst und übergeben.

Ich habe mich schlussendlich für die Version mit einem IoC-Container entschieden, da ich schon einige Zeit mit der IoC-Variante von Unity arbeite und mich relativ gut mit dem Konzept auskenne. Ausserdem lege ich bei dieser Arbeit hohen Wert auf die Qualität des Codes und ohne Dependency Injection ist dies sicherlich schwierig.

8.5 Unity

Ich habe mich bereits für die Verwendung eines IoC-Container entschieden. Es gibt diverse Container, welche das .NET-Framework unterstützen und für die Dependency Injection optimal sind. Da ich jedoch nur mit dem Framework Unity Erfahrung habe und ich keine Zeit für das Evaluieren der anderen Produkte habe, entscheide ich mich für Unity.

9 Realisierung

Die Realisierung und somit der vierte Schritt der IPERKA-Methode, ist die zeitintensivste Phase in den meisten Projekten. Hier werden die zuvor erfüllten Schritte von IPERKA (Planen & Entscheiden) aus der Theorie in die Praxis umgewandelt. Dabei ist es wichtig den Arbeitsablauf einzuhalten und nicht ohne zwingende Gründe zu ändern. In diesem Schritt ist es äusserst wichtig genügend Zeit für eine gute Reflexion einzuplanen, sodass man schnell auf Misserfolge reagieren kann und die Massnahmen daraus planen und umsetzen kann. (VSSM, 2014)

Während der Realisierung habe ich nach dem vorgegebenen Zeitplan unter Kapitel «Zeitplanung» gearbeitet. Es gab Planabweichung, weshalb mein Massnahmen-Konzept zum Einsatz kam und ich musste die letzte Ansicht bzw. Datei der Applikation streichen (OrderConfirmation.txt). Die Abweichungen für jeden Tag habe ich im Arbeitsjournal dokumentiert. Alle anderen Benutzeroberflächen, wie auch alle Funktionalitäten, wurden implementiert.

9.1 SonarQube

Nachdem ich mein Visual Studio mit den benötigten Referenzen aufgesetzt habe, fing ich an mein GitHub-Projekt mit SonarQube zu verlinken. Dazu benutze ich die Onlineversion von SonarQube. Man kann dort ganz einfach sein Repository über GitHub angeben und dann ist man schon fast bereit für die Code-Analyse. Man muss beim ersten Aufsetzen noch eine Path-Variabel, zu der Anwendungsdatei von SonarQube, hinzufügen. Nachdem dies erledigt war, musste ich nur noch präparierte Commands, auf der Visual Studio Command Prompt, ausführen (Commands sind in der SonarQube Anleitung). Momentan ist das Ergebnis von SonarQube nicht sehr aussagekräftig, da noch keine Unittests oder Logik vorhanden sind.

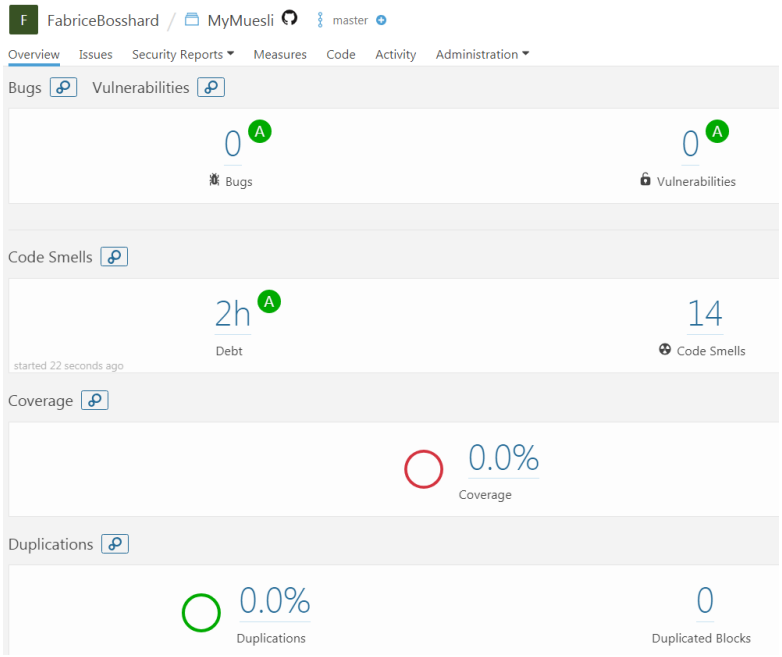


Abbildung 21: Code Analyse SonarQube

9.2 Logger

Als Logging-Framework wurde Log4net verwendet. Die Logging-Konfiguration wurden mit dem Logging-Konzept abgeglichen und implementiert (log4net.config). Man findet die Logging-Dateien im Applikationsverzeichnis (bin) unter «/Logger».

9.3 Styling

Für das Styling wurden keine speziellen Styling-Guides aufgestellt. Trotzdem habe ich mich ein wenig, soviel die Zeit bereit war, mit dem Stylen der Applikation beschäftigt. Folgendes wurde verwendet:

- Farbe: IndianRed (RGB: 205, 92, 92)
- Header-Schrift: Comic Sans MS

9.4 Datenbank

Für die Datenbank-Umsetzung wurde mit Code-First gearbeitet. Man kann dabei einfach Models erstellen, welche die Relationen zwischen den Tabellen enthalten und Entity Framework generiert die entsprechenden Tabellen mit den Datentypen, Primary Keys, und Foreign Keys. Die Models für die Datenbank habe ich unter dem Verzeichnis «/Model» gespeichert.

```
public class CustomerDetails
{
    0 references | FabriceBosshard, 6 days ago | 1 author, 1 change
    public string ID { get; set; }
    3 references | FabriceBosshard, 6 days ago | 1 author, 1 change
    public string Name { get; set; }
    3 references | FabriceBosshard, 6 days ago | 1 author, 1 change
    public string Address { get; set; }
    3 references | FabriceBosshard, 6 days ago | 1 author, 1 change
    public string City { get; set; }
    3 references | FabriceBosshard, 6 days ago | 1 author, 1 change
    public string Zip { get; set; }
    3 references | FabriceBosshard, 1 day ago | 1 author, 2 changes
    public Country Country { get; set; }
    3 references | FabriceBosshard, 6 days ago | 1 author, 1 change
    public string Phone { get; set; }
    3 references | FabriceBosshard, 6 days ago | 1 author, 1 change
    public string Email { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public virtual ICollection<Cereal> Cereals { get; set; }
}
```

Abbildung 22: Model Beispiel EF

Wenn man die Models erstellt hat, braucht es noch einen sogenannten Kontext, welcher für die Kommunikation, mit der Datenbank, zuständig ist. Man registriert auf dieser Klasse die verschiedenen DbSets. Ein DbSet ist die Repräsentation einer Tabelle in Entity Framework.

```
9 references | 0 changes | 0 authors, 0 changes
public class MyCerealContext : DbContext
{
    1 reference | 0 changes | 0 authors, 0 changes
    public DbSet<Country> Countries { get; set; }
    1 reference | 0 changes | 0 authors, 0 changes
    public DbSet<CustomerDetails> CustomerDetails { get; set; }
    2 references | 0 changes | 0 authors, 0 changes
    public DbSet<Ingredient> Ingredients { get; set; }
    4 references | 0 changes | 0 authors, 0 changes
    public DbSet<Cereal> Cereals { get; set; }
    1 reference | 0 changes | 0 authors, 0 changes
    public DbSet<Category> Categories { get; set; }
```

Abbildung 23: MyCerealContext

Dieser Kontext kann nun gebraucht werden um alle möglichen Abfragen zu machen. Das tolle an Entity Framework ist, ich muss keine SQL-Queries schreiben, sondern kann min LINQ arbeiten. LINQ (Language-Integrated Queries) sind Sprach-Integrierte-Abfragen, welche im Kontext von Entity Framework in Queries umgewandelt werden.

```
public void AddUser(CustomerDetails customer)
{
    using (var ctx = new MyCerealContext())
    {

        ctx.CustomerDetails.Add(customer);
        ctx.SaveChanges();
    }
}
```

Abbildung 24: Gebrauch von MyCerealContext

Somit konnte ich ziemlich rasch alle Datenbank-Abfragen vorbereiten und musste nicht noch Zeit aufwenden für die SQL-Command. Noch eine positive Eigenschaft an Entity Framework ist, dass die Datenbank zu Beginn erstellt wird und wenn man Anpassungen an einer Tabelle machen möchte, kann man dazu ganz einfach ein Migrations-Befehl ausführen und die Datenbank ist auf dem neusten Stand.
Für die Testdaten in der Datenbank wurden die Testdaten aus dem Schweizermeisterschaften von 2016 verwendet. Folgende Tabellen wurden bereitgestellt:

- Countries
- Ingredients

10 Kontrollieren

Der wichtigste Schritt in der IPERKA-Methode ist das Kontrollieren. Man kann seine Arbeit nicht abgeben, ohne dass man alles nochmals kontrolliert hat. Hierbei werden vor allem die Vorgaben, welche sich aus dem Schritt «Planen» ergeben haben, mit dem tatsächlichen Ergebnis verglichen und die Resultate festgehalten. Neben der Kontrolle des eigentlichen Produktes, ist es auch wichtig das Arbeitsverhalten zu überprüfen und allenfalls Abweichungen aus den Plänen aufzuzeigen. (VSSM, 2014)

10.1 Meilensteine

Ich habe mir in der Zeitplanung gewisse Meilensteine definiert, welche ich als Leitfaden für die Realisierung genutzt habe. In diesem Schritt kontrolliere ich nun meine geplanten Meilensteine mit den wirklichen Projektdaten. Im Kapitel «Auswerten» werden die Soll- / Ist-Zeiten reflektiert.

Meilenstein	Soll-Datum	Ist-Datum
Projektbeginn	31.10.2018 08:30 Uhr	31.10.2018 08:30 Uhr
Informieren	31.11.2018 17:00 Uhr	31.11.2018 17:00 Uhr
Planen	01.11.2018 17:00 Uhr	01.11.2018 17:00 Uhr
Entscheidung	02.11.2018 11:00 Uhr	02.11.2018 13:00 Uhr
Realisieren	07.11.2018 17:00 Uhr	07.11.2018 17:00 Uhr
Kontrollieren	09.11.2018 11:00 Uhr	09.11.2018 12:00 Uhr
Auswerten	09.11.2018 16:00 Uhr	09.11.2018 15:00 Uhr
Projektabschluss	09.11.2018 17:00 Uhr	09.11.2018 17:00 Uhr

10.2 Testprotokoll

Im Kapitel «Planen» habe ich Testfälle zu den verschiedenen Use Cases erstellt. Beim Kontrollieren führe ich diese Testfälle aus und halte ihre Ergebnisse fest.
Ich verweise jeweils auf die Nummer der Testfälle im Kapitel «8.2 Testkonzept» (z. B. Testresultat 1.1 = Testfall 1.1) und stelle die Ergebnisse wie folgt dar.

Bewertung
Ich habe mir drei verschiedene Bewertung für einen Test überlegt:

- ✓ (Bestanden ohne Abweichung)
- \- (Bestanden mit Abweichung)
- X (Nicht bestanden)

Wenn eine Abweichung bekannt ist, oder der Test fehlschlägt, dann muss im Testprotokoll die Abweichung beschrieben werden.

(Use Case 1) Nr.	Kurze Beschreibung	Resultat	Abweichung
1.1	Kunde valid erfassen	✓	-
1.2	Falscher Name erfassen	✓	-
1.3	Falsche Adresse erfassen	✓	-
1.4	Falsche Postleitzahl erfassen	✓	-

1.5	Falsche Stadt erfassen	✓	-
1.6	Falsche Telefonnummer erfassen	✓	-
1.7	Falsche Mail erfassen	✓	-

Tabelle 26: Testprotokoll UseCase 1

(Use Case 2) Nr.	Kurze Beschreibung	Resultat	Abweichung
2.1	Valides Muesli erfassen.	✓	-
2.2	Muesli ohne Basiszutat erfassen.	✓	-
2.3	Muesli mit mehr als 12 zutaten.	✓	-
2.4	Muesli mit mehr als 600g.	✓	-
2.5	Muesli Mixer interagieren => Neue werte	✓	-
2.6	Details öffnen	✓	-
2.7	Information öffnen	✓	-
2.8	Zutat entfernen	X	ListView hat DataBinding Probleme kann auf keine Commands anspringen

Tabelle 27:Testprotokoll UseCase 2

(Use Case 3) Nr.	Kurze Beschreibung	Resultat	Abweichung
3.1	MyMuesliMixes öffnen, editieren	✓	-
3.2	Muesli löschen	✓	-

Tabelle 28: Testprotokoll UseCase 3

(Use Case 4) Nr.	Kurze Beschreibung	Resultat	Abweichung
4.1	Order öffnen => Anzahl gleich 2	✓	-
4.2	XXL ist ausgewählt	✓	-
4.3	Keine Anzahl gewählt	✓	-
4.4	Anzahl auf 2 setzen Muesli bestellen	-	Order Confirmation wurde nicht erstellt

Tabelle 29: Testprotokoll UseCase 4

(Use Case 5) Nr.	Kurze Beschreibung	Resultat	Abweichung
5.1	OrderConfirmation ansehen	X	Order Confirmation wurde nicht erstellt

Tabelle 30: Testprotokoll UseCase 5

10.3 Unit-/Modultest

Ursprünglich war geplant, dass ich in meiner Business-Logik eine Testabdeckung von 60% erreiche. Da die Applikation jedoch fast nur mit ViewModels arbeitet und diese immer gewisse Abhängigkeiten injiziert bekommen, war es sehr schwer diese zu testen. Ich hätte dafür Mocks gebraucht, für die ich aber keine Zeit mehr hatte. Ich habe versucht, soviel es ging, die wichtigsten Logik-Einheiten zu testen:

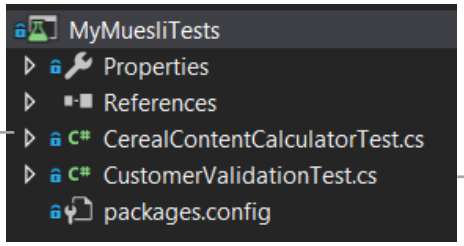


Abbildung 26: MyMuesliTests Struktur

Zuständig für die Logik des MuesliMixers: Berechnet Nährstoffwerte und Preise.

Zuständig für die Validierung eines Kunden.

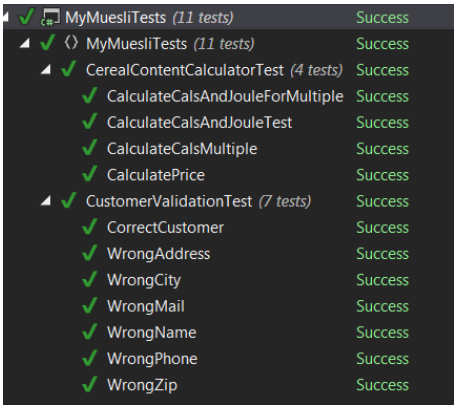


Abbildung 25: MyMuesliTests Resultat

11 Auswerten

Im letzten Schritt von IPERKA reflektiert man nochmals, vom Informieren bis zum Kontrollieren die gesamte Bearbeitung des Auftrags. Beim Auswerten geht es um eine Selbsteinschätzung der erledigten Tätigkeiten. Man überlegt sich darüber hinaus was man aus der Arbeit gelernt hat, was man verbessern kann und wo man das Gelernte in der Zukunft anwenden kann. (VSSM, 2014)

11.1 Reflexion

Ich hatte in diesem Projekt ständig gemischte Gefühle. Es war irgendwie sehr gut aber auch irgendwie nicht. Das liegt daran, dass mir viele Fehler passiert sind, die mir in den Zeitplan gepfuscht haben und mich in Stress verleitet haben. Diese Fehler waren mir nur unterlaufen, da ich mich Zuwenig auf die Aufgabe vorbereitet habe und deshalb die Planung etwas bedürftig war. Dieses Problem liess sich vor allem auf die ersten zwei Tage des Projekts zurückführen, denn dort war ich mit dem Informieren und Planen beschäftigt. Wie in meinen ersten Arbeitsjournalen ersichtlich, liefen diese Tage ziemlich chaotisch und ich hatte auch nie wirklich einen Tagesplan erstellt, sondern habe einfach darauf losgearbeitet. Dies hatte sich negativ auf meine ganze Arbeit ausgewirkt. Es ist nun mal so dass die Planung der Boden der Realisierung ist. Durch diese Komplexität hatte ich immer wieder Verzug und musst einfach schneller und effizienter arbeiten und wenn dann mal etwas nicht funktioniert hatte, geriet ich ziemlich in Stress und das war sehr unangenehm.
In der zweiten Woche, sprich am 07.11 und 09.11 lief es um weites besser. Ich wusste wie ich mir den Tag zurechtlegen muss, dass ich eine Art Leitfaden habe und auch immer wieder diese Unter-Meilensteine abschliessen und kontrollieren kann. Ich würde sagen ich habe fast mehr an den letzten zwei Tagen erledigt als zuvor. Das liegt ganz klar daran, dass ich organisiert in den Tag gestartet bin und mir bereits am Abend zuvor Gedanken zu den einzelnen Themen gemacht habe. Somit hatte ich viel weniger Stress und konnte mich voll und ganz auf die Arbeit konzentrieren. Ich glaube auch, dass das Arbeitsklima in den letzten Tagen etwas angenehmer war. Zu Beginn hatte ich damit ziemlich Mühe, da ständig geschwatzt wurde und ich so meine Gedankenflüsse unterbrach.

11.2 Erkenntnisse und Erfahrungen

Ich habe während dieser Arbeit sehr viele kleine Erkenntnisse gemacht. Ich habe gewisse neue Elemente in WPF kennengelernt, habe meine DataBinding-Kenntnisse auf Vordermann gebracht oder habe zum Beispiel viel Neues über das Entity Framework gelernt, doch was die grösste Erkenntnis für mich war, war die Ausprägung der Planung. Leider kam mir diese Erkenntnis erst zwei Tage nach der Planungsphase, aber besser dann als nie. Mir ist klar geworden, wie man sich gut auf so ein Projekt vorbereiten kann und wie die Planung richtig durchgeführt wird. Ich hatte einfach zu wenig die verschiedenen Faktoren, wie Expertengespräch, Leistungsabnahme bei Müdigkeit oder auch unerwartete Fehler, in der Planung eingebaut und musste deshalb eigentlich immer auf die Minute genau arbeiten. Ich muss mir für die richtige IPA immer einen Platzhalter in der Planung lassen, damit diese Zeit genutzt werden kann, damit ich allenfalls Verzug nachholen kann. Somit schaffe ich immer die Muss-Kriterien und kann allenfalls noch Optionale-Kriterien erledigen. Jetzt war es so, dass sobald ich Verzug hatte, musste ich etwas aus der Aufgabenstellung rauswerfen und das gab mich auch ein schlechtes Gefühl, da ich gerne meine Sachen sauber und korrekt abschliesse.

Eine weitere wichtige Erkenntnis kam mir in der Realisierungsphase. Ich hatte oftmals Mühe mit den komplexen Elementen auf der Benutzeroberfläche. Ich kannte alle Controls, doch wusste manchmal nicht die optimale Umsetzung und dies kostete mich benötigte Zeit. Ich sollte mir zuvor immer genau die Aufgabenstellung anschauen und mich über die komplexen Einheiten informieren, sodass ich allenfalls etwas bereits anschauen kann und nicht Zeit für das Lernen dieser Dinge verschwende.

11.3 Erweiterungen

Obwohl das Projekt eigentlich abgeschlossen ist, schweben mir noch so einige sinnvolle Erweiterungen im Kopf. Ich werde die Erweiterung aufzählen, welche ich theoretisch im Anschluss als Erstes machen würde.

OrderConfirmation
Wie schon an einigen Stellen erwähnt, konnte ich das letzte Kriterium der Aufgabenstellung aus Zeitgründen nicht mehr erledigen. Dies würde sicherlich als Erstes anstehen, da so zumindest die Muss-Kriterien der Aufgabenstellung erfüllt werden. Je nach Problemen, schätze ich für diese Aufgabe einen Aufwand von zwei Stunden.

Mocks
Ich hatte mir vorgenommen eine Testabdeckung von 60% für meinen Business-Code zu erreichen. Jedoch merkte ich schnell, dass meine ViewModels einen grossen Teil der Applikation ausmachen, und da diese mit Dependency Injection erstellt werden, müsste ich diese Abhängigkeiten mocken. Somit könnte ich die einzelnen ViewModels testen, ohne deren Abhängigkeiten zu kennen. Damit ich durch die Mocks eine Testabdeckung von 60% erreichen kann, bräuchte ich noch 2-3 Stunden.

DatabaseConnectionPool
Momentan wird für jede Datenbank-Abfrage eine neue Verbindung zur Datenbank aufgebaut. Man könnte diese Verbindungen in einen Pool legen, wo man sie immer wiederverwenden kann. Das würde die Ladezeit der einzelnen Abfragen verkürzen. Für diese Aufgabe schätze ich einen Mehraufwand von einer Stunde.

11.4 Fazit

Obwohl ich nicht ganz alles fertiggeschafft habe bin ich ziemlich Stolz auf meine Arbeit. Ich habe noch nie eine so grosse Arbeit geschrieben und auch die Stressbewältigung in dieser Zeit war eine sehr interessante Herausforderung für mich. Ich bin sehr froh, dass wir diese Probe-IPA gemacht haben, weil ich einen sehr grossen Lerneffekt während und nach der Arbeit hatte. Ich werde die Fehler, die mir passiert sind, angehen und versuchen mich entsprechend auf die richtige IPA vorzubereiten. Es ist fast schon besser, dass mir so viele Fehler jetzt passiert sind, denn nur so kann ich daraus lernen und es ist mir lieber, dass sie jetzt passieren als in der richtigen IPA.

12 Glossar

Im Glossar werden die wichtigsten Fremdwörter und Technische Ausdrücke erklärt.

Begriff	Erklärung
Vulnerability	Vulnerability, also eine Vulnerabilität, ist eine Schwachstelle in der Applikation.
Continious Integration (CI)	Kontinuierliche Integration beschreibt den Prozess, bei dem man fortlaufend Komponenten aus der Software zusammenfügt. CI wird vor allem für die stetige Steigerung der Softwarequalität genutzt.
Model View ViewModel (MVVM)	MVVM ist eine Beschreibung einer Architektur in der Software-Programmierung
Daily	Ein Daily ist eines der Scrum Artefakte. Ein Daily ist ein tägliches Meeting innerhalb des Entwicklerteams, wo über abgeschlossene und bevorstehende Arbeiten geredet wird.
ORM	Object-Relational Mapping, ist ein Konzept zum Abbildung von Objekten in einer relationalen Datenbank.
Wire-Frames	Wire-Frames werden dazu benutzt um eine konzeptionelle Abbildung einer Benutzeroberfläche darzustellen. Dabei wird wenig auf Design und Funktionalität geachtet, sondern man konzentriert sich auf die Anordnung der Elemente
UI-Controls	UI (User Interface) ist die Bezeichnung für die Benutzeroberfläche. UI-Controls sind die angewendeten Bausteine, für die Benutzeroberflächen-Programmierung.
Inversion of Control (IOC)	Dieses Programmierparadigma beschreibt die Arbeitsweise einer Einheit, eines Programms, oder eines Frameworks. In meinem Beispiel: Statt, dass die einzelnen Klassen sich um ihre Abhängigkeiten kümmern, wird die Kontrolle einer anderen Einheit übergeben. Sie regelt den Kontrollfluss und löst die Abhängigkeiten auf.
Command	Ein Command ist ein Befehl. Wenn ein Benutzer etwas klickt, führt er einen Befehl aus, welcher in der Applikation etwas auslöst.
Kohäsion	Kohäsion beschreibt, wie gut eine Einheit seine Logik abbildet. In der Programmierung versucht man eine möglichst hohe Kohäsion zu erreichen. Das heisst, dass jede Einheit(Klasse) nur für eine Aufgabe zuständig ist.
Mocks	Mocks, meistens Mockups, sind Vorführmodelle, welche in der Programmierung zur Modellierung von gewissen Einheiten genutzt werden.

Tabelle 31: Glossar

13 Verzeichnisse

Anbei werden alle verwendeten Ressourcen aufgelistet.

13.1 Quellenverzeichnis

Developer, C. C. (2015). Clean Code Prinzipien. Abgerufen am 09. November 2018 von https://clean-code-developer.de/

mackenir. (2009). Databinding TabControl. Abgerufen am 07. November 2018 von https://stackoverflow.com/questions/686074/wpf-tabcontrol-databinding

S.A, S. (2008). Leading Product for CI. Abgerufen am 30. Oktober 2018 von https://www.sonarqube.org/

VSSM, V. s. (2014). Die 6-Schritte-Methode. Abgerufen am 24. Oktober 2018 von https://www.vssm.ch/sites/default/files/vssm/BB/docs/AusbildungEFZ/IPERKA_1_4_D2017.pdf

Wikipedia. (16. September 2018). MVVM. Abgerufen am 01. November 2018 von https://de.wikipedia.org/wiki/Model_View_ViewModel

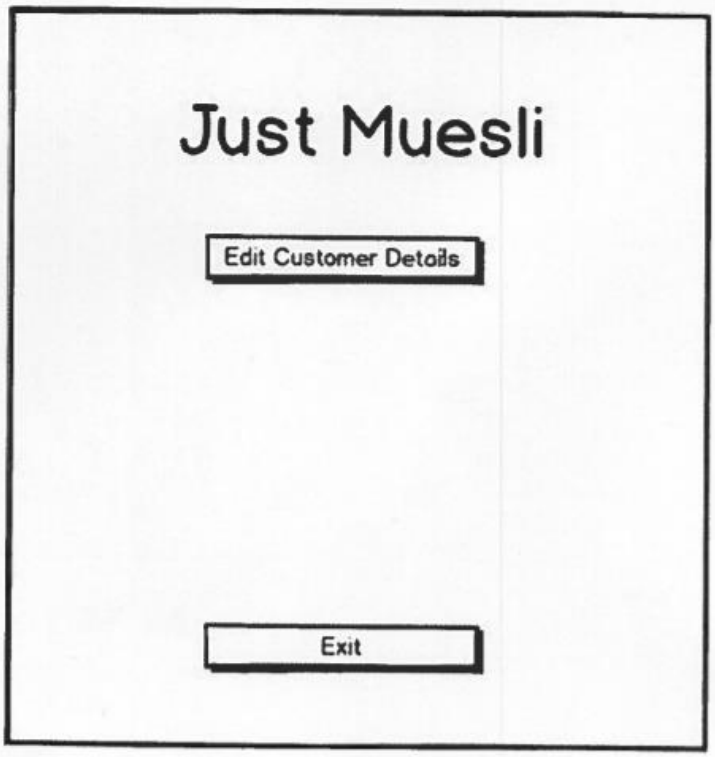
13.2 Abbildungsverzeichnis

13.3 Tabellenverzeichnis

14 Aufgabenstellung

7 Menu (Part 1)

After launching the application, a menu should be displayed. This menu allows the user to access the other parts of the application. Please create the window according to the following mockup and description. This menu will be extended with more options in Part 2.



- This menu (window) is displayed after launching the application.
- The Button **"Edit Customer Details"** should open the window described in the next chapter.
- The button **"Exit"** terminates the application.

Abbildung 27: Just Muesli "Menu"

8 Edit Customer Details

This window is used to manage the details of the customer. Please create the window according to the following mockup and description.

- This window is accessible by the button **"Edit Customer Details"** on the main menu.
- The user should be able to manage the following data: name, address, zip, city, country, phone and email. Please use appropriate edit controls.
- Use the imported list of countries from the file **"Countries.txt"**.
- To avoid erroneous entries, the data should be validated according to the validation rules below.
- The button **"Save"** stores the data in the database. Saving should only be possible, if there are no violations of validation rules. An error message, that mentions all invalid fields, should be shown if invalid data has been entered.
- The button **"Back to menu"** leads back to the main menu.

Field	Validation rule
Name	Length: min 5 characters
Address	Length: min 5 characters
Zip	Length: min 4 characters, only numeric values (0-9)
City	Length: min 2 characters
Country	A country from the provided list must be selected
Phone	Length: min 10 characters, only numeric values, blanks or "+"
Email	Pattern: x@y.z x: min 1 character y: min 3 characters z: 2-3 characters (only [a-zA-Z])

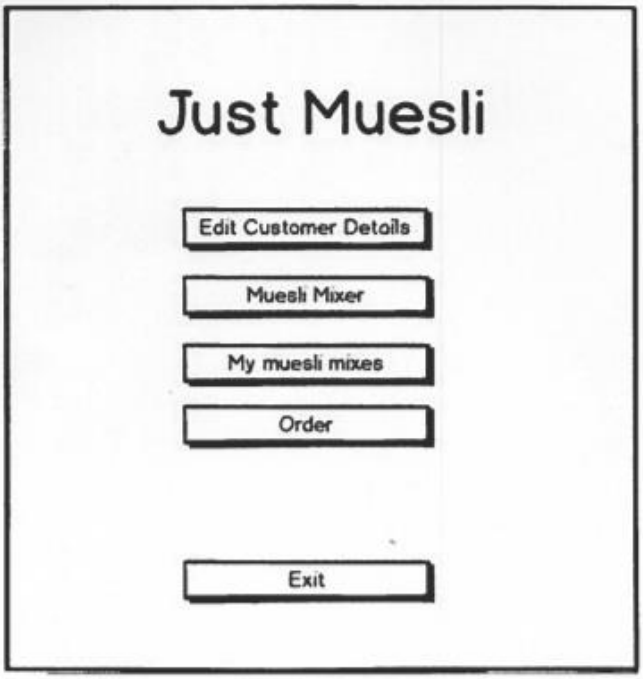
Abbildung 28: Just Muesli "Customer Details"

PART 2

- You can start working on part 2 anytime during the competition.
- Assessment of part 2 starts as soon as the competition has finished.
- Please adhere to the style guide when designing user interfaces.

9 Menu (Part 2)

Please extend the Menu by adding the additional buttons shown in the mockup.



- In the end, all buttons should lead to the correct part of the program according to the following chapters.
- The button **"Order"** should be disabled, if no saved muesli mix exists.

Abbildung 29: Just Muesli "Menu Part 2"

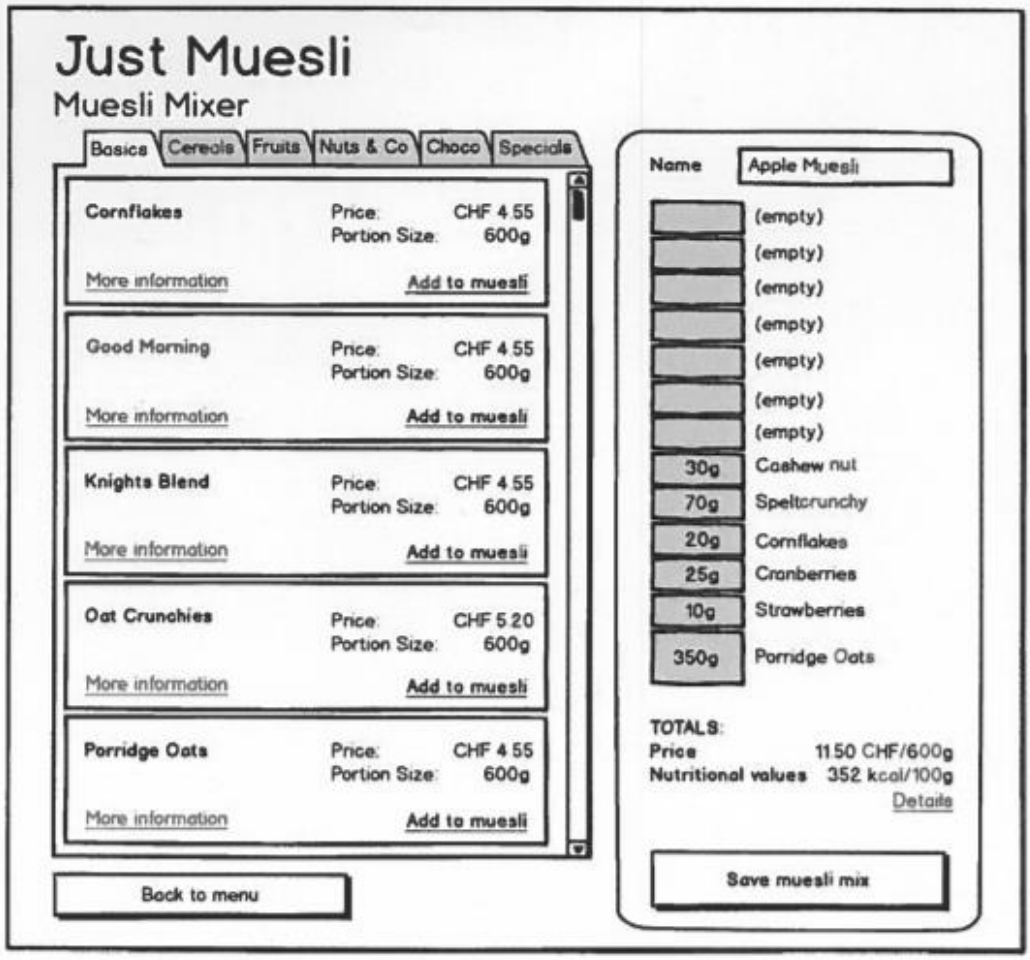
10 Muesli Mixer

This window is used for the core of the application: an innovative muesli mixer. It serves to create individualized muesli mixes out of many components. Please create the window according to the following mockup and description.

What is a "muesli mix"?

A muesli mix always consists of a base component (anything from category "Basics") and up to twelve additional components. The base component must always be added first. The components belong to the following categories: Basics, Cereal, Fruit, Nuts & Co, Choco, Specials.

The total weight of a muesli is always 600g. The base component acts as a «filler», whose amount is calculated automatically (600g – [Total weight of additional components]).

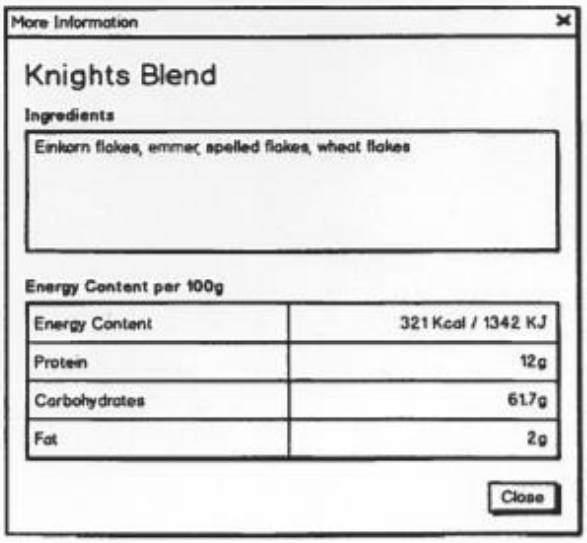


- This window is accessible by the button **"Muesli Mixer"** on the main menu.

Abbildung 30: Just Muesli "Muesli Mixer"

10.1 Component list (left side)

- There should be a tab for each category containing a list of all corresponding components from the database. The list should be sorted ascending by name.
- The following information should be visible for each component in the list: name, price, weight of the portion.
- The user should be able to add a component to the muesli mix by clicking a link or button **"Add to muesli"**. If the muesli mix already contains 12 components in addition to the base component, an error message should be shown instead.
- Only one base component can be added to the mix. When the user tries to add another base component, it should replace the existing one. The base component must always be added first.
- The button **"Back to menu"** leads back to the main menu.
- The button **"More information"** should open a popup window that displays the ingredients, nutrients and energy content of the component. Use the table from chapter 10.3 to calculate the energy content.



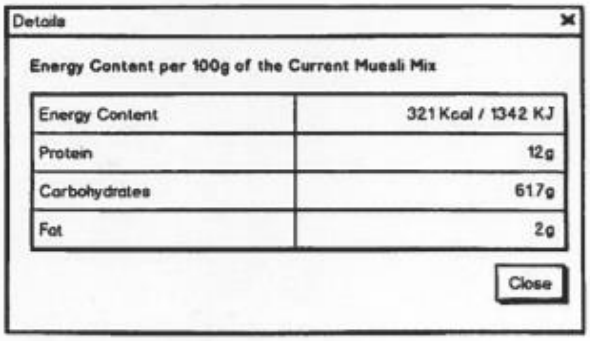
10.2 Current muesli mix (right side)

- The chosen components should be displayed "stacked" according to the adding order, but the base component should always be placed right at the bottom.
- An error message should be shown if the total weight (600g) is exceeded by the current muesli mix.
- The total price (in CHF per 600g) and the nutritional values (in kcal per 100g) of the current muesli mix should be automatically calculated based on the selected components and displayed. Please take care of the following points for the calculation:
    - The price for the basic component is based on a portion size of 600g. Since the portion size is determined automatically, the price must also be adjusted accordingly. The price must be rounded to the second decimal place (0.01).

Abbildung 31: Just Muesli "Component list"

- Nutritional information (e.g. carbohydrate content) in the import file are given per 100g of the component. The values must be converted to the portion size if necessary.

- The user can remove a component from the mix by clicking on this component in the muesli stack. A confirmation prompt should be shown to ask if the user really wants to remove the component.
- To help the user distinguish his muesli mixes later, the user should be able to assign a name and save his current mix. The button **"Save Muesli"** stores the muesli mix in the database.
  A meaningful error message should be shown in the following situations:
  - No muesli name has been entered.
  - No basic component has been chosen.
- Additional nutritional details should be provided for the health-conscious users. After clicking the button **"details"**, a detailed breakdown of the nutrients should be shown in a separate window (proteins, carbohydrates, fats in kcal and KJ per 100g muesli). Please use the table below to calculate these values.



10.3 Calculation of the energy content

Use the fat, carbohydrate and protein content of the components to calculate the energy content using this conversion table.

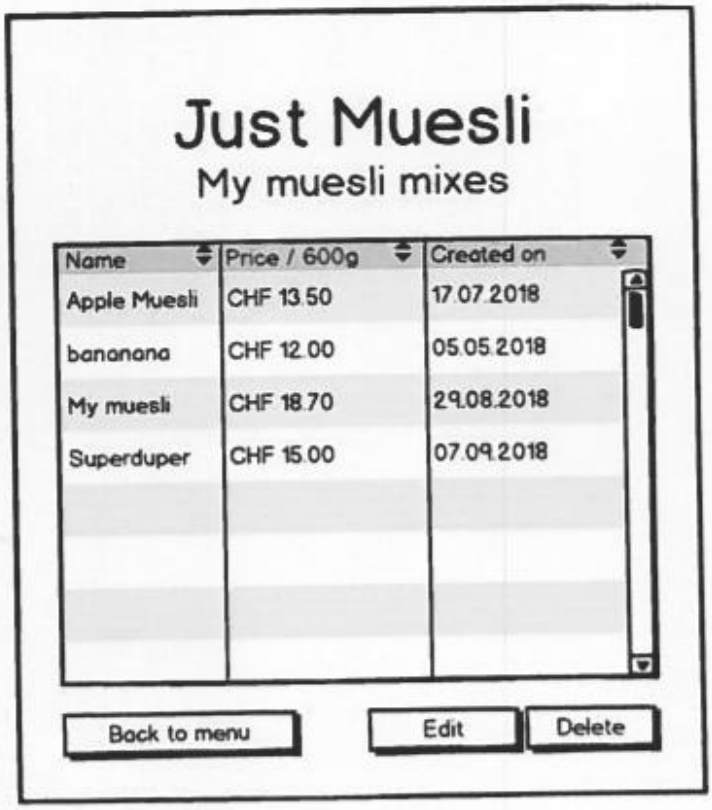
1 Kcal = 4.184 KJ

Nutrient	Kilocalories per gram [Kcal/g]	Kilojoules per gram [KJ/g]
Carbohydrates	4.1	17.2
Proteins	4.1	17.2
Fats	9.3	38.9

Abbildung 32: Just Muesli "Current Muesli"

11 My muesli mixes (Edit Muesli)

This window is used to manage the saved muesli mixes of the customer. Please create the window according to the following mockup and description.

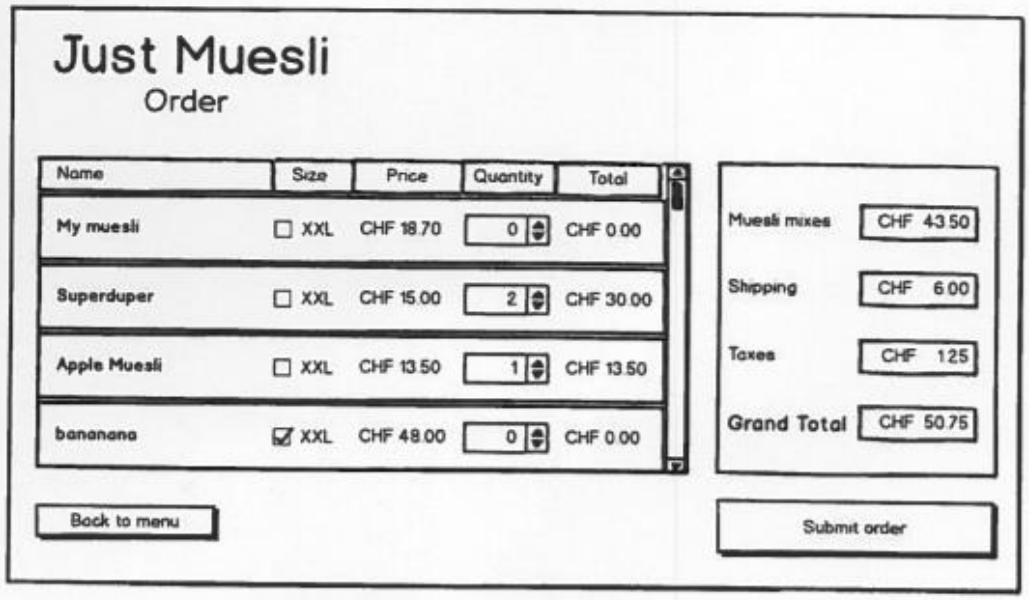


- This window is accessible by the button **"My muesli mixes"** on the main menu.
- All saved muesli mixes from the database should be listed in a table containing the name, price and date of creation in separate columns.
- The table should be sorted ascending by name, but the user should be able to sort the table by other columns too.
- The button **"Edit"** opens the window **"Muesli Mixer"** to edit the selected muesli mix.
- The user can remove the selected muesli mix by clicking on the button **"Delete"**. A confirmation prompt should be shown to ask if the user really wants to delete the mix.
- An error message should be shown if the user clicks **"Edit"** or **"Delete"** and no muesli mix is selected.
- The button **"Back to menu"** leads back to the main menu.

Abbildung 33: Just Muesli "My Muesli Mixes"

12 Order

This window is used to order saved muesli mixes. Please create the window according to the following mockup and description.



- This window is accessible by the button **"Order"** on the main menu.

12.1 Muesli mix list (left side)

- All saved muesli mixes from the database should be listed containing the name and price (sorted descending by price per 600g).
- The button **"Submit order"** is disabled, when no muesli mix is ordered (Quantity = 0 for all muesli mixes)
- The user should be able to order a large version by selecting the XXL option of a muesli mix. The XXL size has a weight of 2'400g and thus the displayed price will be increased to the fourfold.
- The user should be able to enter the desired quantity for each muesli mix.
- Each row should end up with a subtotal (displayed price * quantity).
- The button **"Back to menu"** leads back to the main menu.

12.2 Costs (right side)

- The order total ("Muesli mixes") corresponds to the sum of all subtotals.
- The shipping costs within Switzerland are usually CHF 6.00 and for shipping abroad CHF 8.00. If the order total (costs of ordered Muesli mixes) exceeds CHF 50.00, no shipping fee is charged.
- If the order will be shipped abroad, no tax will be added to the order. For Swiss customers, an additional 2.5% tax will be charged and added to the grand total (rounded to the nearest CHF 0.05).
- The text boxes showing the costs must be read-only.

Abbildung 34: Just Muesli "Order"

- When the user clicks the button **"Submit order"**, an order confirmation file should be created. The detailed requirements are defined in the next chapter. Please note that it is not required to save the order in the database.

13 Order Confirmation

The order confirmation should be created as a text file, which is formatted like the following example:

```
Dear [Name]

Thank you very much for using JustMuesli to order your personal Muesli

Order overview

Item                      Size         Price   Quantity         Total
-------------------------------------------------------------------------
Superduper                Normal       15.00          2         30.00
Apple Muesli              Normal       13.50          1         13.50
bananana                  XXL          48.00          1         48.00
-------------------------------------------------------------------------
Total Muesli mixes                                                91.50
Shipping                                                           0.00
Taxes                                                              2.30
=========================================================================
Grand Total                                                       93.80
=========================================================================


The ordered items will be sent to the following address:
[Name]
[Address]
[Zip] [City]
[Country]
```

The order confirmation should be saved in the working directory of the executable. Use the filename "orderconfirmation.txt" (existing order confirmation file should be overwritten).

Placeholders enclosed in brackets (e.g. [Name]) must be replaced by the actual customer data. The order overview table must contain the ordered items (name, size, price, quantity and calculated line total), total cost of muesli mixes, shipping fees, taxes and the grand total. The columns of the table must be formatted as follows:

Column	Size (number of characters)	Format	Alignment
Item	25	Text	Left
Size	7	Text	Left
Price	10	Number with 2 decimal places	Right
Quantity	10	Number with no decimal places	Right
Total	14	Number with 2 decimal places	Right

Use spaces to pad the data and achieve the desired alignment. You can assume that no data will be used that exceeds the defined size.

Abbildung 35: Just Muesli "Order Confirmation"

15 Programmcode

Views

OrderView.xaml

```
<Window x:Class="MyMuesli.Views.OrderView"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:MyMuesli.Views"
        mc:Ignorable="d"
        Title="OrderView" Height="700" Width="1200" ResizeMode="NoResize"
WindowStartupLocation="CenterScreen"
        DataContext="{Binding Order, Source={StaticResource ViewModelLocator}}">
  <Window.Resources>
    <DataTemplate x:Key="HeaderNumberBox">
      <Grid Margin="1">
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="*" />
          <ColumnDefinition Width="20" />
        </Grid.ColumnDefinitions>
        <TextBlock TextAlignment="Right"
                   Grid.Column="0" Text="{Binding}" />
        <StackPanel Orientation="Vertical"
                    Grid.Column="1"
                    VerticalAlignment="Center">
          <Button Height="5" Width="10" Command="{Binding IncreaseCommand}" />
          <Button Height="5" Width="10" Command="{Binding DecreaseCommand}" />
        </StackPanel>
      </Grid>
    </DataTemplate>
  </Window.Resources>
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="150" />
      <RowDefinition />
      <RowDefinition Height="100" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition />
      <ColumnDefinition Width="600" />
      <ColumnDefinition Width="400" />
    </Grid.ColumnDefinitions>
    <Grid Grid.ColumnSpan="2" Width="400" HorizontalAlignment="Left">
      <Grid.RowDefinitions>
        <RowDefinition Height="75" />
        <RowDefinition />
      </Grid.RowDefinitions>
      <Label Grid.Row="0" FontSize="50" VerticalAlignment="Center" FontFamily="Comic
Sans MS"
             HorizontalAlignment="Center" Content="Just Muesli" />
      <Label Grid.Row="1" FontSize="30" VerticalAlignment="Center" FontFamily="Comic
Sans MS"
             HorizontalAlignment="Center" Content="Order" />
    </Grid>
```

```
    <Border Grid.Row="1" Margin="20,20" Grid.Column="2" BorderBrush="Black"
BorderThickness="1">
      <Grid>
        <Grid.RowDefinitions>
          <RowDefinition />
          <RowDefinition />
          <RowDefinition />
          <RowDefinition />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
          <ColumnDefinition />
          <ColumnDefinition />
        </Grid.ColumnDefinitions>

        <Label Grid.Row="0" Grid.Column="0" Content="Muesli mixes" Height="50"
Width="150"
               VerticalAlignment="Center"
               VerticalContentAlignment="Center" HorizontalContentAlignment="Center"
               HorizontalAlignment="Center"
               FontSize="20" />
        <Label Grid.Row="1" Grid.Column="0" Content="Shipping" Height="50"
Width="150"
               VerticalAlignment="Center"
               VerticalContentAlignment="Center" HorizontalContentAlignment="Center"
               HorizontalAlignment="Center"
               FontSize="20" />
        <Label Grid.Row="2" Grid.Column="0" Content="Taxes" Height="50" Width="150"
               VerticalAlignment="Center"
               VerticalContentAlignment="Center" HorizontalContentAlignment="Center"
               HorizontalAlignment="Center"
               FontSize="20" />
        <Label Grid.Row="3" Grid.Column="0" Content="Grand Total" Height="50"
Width="150"
               VerticalAlignment="Center"
               VerticalContentAlignment="Center" HorizontalContentAlignment="Center"
               HorizontalAlignment="Center"
               FontSize="20" />
        <Border Grid.Row="0" Grid.Column="1" Height="50" Margin="5,5"
BorderBrush="Black"
                BorderThickness="1">
          <TextBlock Text="{Binding MuesliMixesValues,StringFormat='CHF \{0\}'}"
                     FontSize="16" />
        </Border>
        <Border Grid.Row="1" Margin="5,5" Grid.Column="1" Height="50"
BorderBrush="Black"
                BorderThickness="1">
          <TextBlock Text="{Binding ShippingPrice,StringFormat='CHF \{0\}'}"
                     FontSize="16" />
        </Border>
        <Border Grid.Row="2" Grid.Column="1" Margin="5,5" Height="50"
BorderBrush="Black"
                BorderThickness="1">
          <TextBlock Text="{Binding TaxesValue,StringFormat='CHF \{0\}'}"
FontSize="16" />
        </Border>
```

```
        <Border Grid.Row="3" Grid.Column="1" Margin="5,5" Height="50"
BorderBrush="Black"
                BorderThickness="1">
          <TextBlock Text="{Binding GrandTotal,StringFormat='CHF \{0\}'}"
FontSize="16" />
        </Border>
      </Grid>

    </Border>
    <DataGrid Grid.Row="1" ItemsSource="{Binding MyCereals}"
AutoGenerateColumns="False"
              HorizontalScrollBarVisibility="Visible"
              Grid.Column="0" Grid.ColumnSpan="2" Margin="20">
      <DataGrid.Columns>
        <DataGridTextColumn Header="Name" Binding="{Binding Name}" FontSize="16"
Width="250" />
        <DataGridTextColumn Header="Size" Binding="{Binding IsXXl}" FontSize="16"
Width="120" />
        <DataGridTextColumn Header="Price" Binding="{Binding Price, StringFormat='CHF
\{0\}'}"
                            FontSize="16"
                            Width="120" />
        <DataGridTextColumn Header="Quantity" HeaderTemplate="{StaticResource
HeaderNumberBox}"
                            Binding="{Binding Quantity}" FontSize="16" Width="120" />
        <DataGridTextColumn Header="Total" Binding="{Binding Total, StringFormat='CHF
\{0\}'}"
                            FontSize="16"
                            Width="175" />
      </DataGrid.Columns>
    </DataGrid>
    <Button Grid.Row="2" Margin="20,0" Grid.ColumnSpan="2" Background="IndianRed"
            HorizontalAlignment="Left"
            Grid.Column="0" Height="40"
            Width="200" Content="Back to menu" Command="{Binding MenuCommand}"
FontSize="20" />
    <Button Grid.Column="2" Margin="20" IsEnabled="{Binding IsOrderEnabled}"
Grid.Row="2"
            Background="IndianRed"
            Command="{Binding SaveCommand}"
            Content="Submit Order"
            FontSize="20" />
  </Grid>
</Window>
```

CerealMixerView.xaml

```
<Window x:Class="MyMuesli.Views.CerealMixerView"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      mc:Ignorable="d"
      Title="CerealMixerView" ResizeMode="NoResize" Height="800" Width="800"
WindowStartupLocation="CenterScreen"
      DataContext="{Binding Cereal, Source={StaticResource ViewModelLocator}}">
  <Window.Resources>
```

```
    <ControlTemplate x:Key="AddedListTemplate">
      <Grid Margin="2,2">
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="100" />
          <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Border BorderThickness="1" BorderBrush="Black">
          <TextBlock HorizontalAlignment="Center" FontSize="12" Background="DimGray"
                     Text="{Binding Portion, StringFormat='\{0\} g'}"
                     TextAlignment="Center" Width="100" />
        </Border>
        <TextBlock HorizontalAlignment="Left" Grid.Column="1" FontSize="16" Margin="5,0"
                   VerticalAlignment="Center" Text="{Binding Name, FallbackValue='(Empty)'}" />
      </Grid>
    </ControlTemplate>
    <DataTemplate x:Key="ListTemplate">
      <Grid Height="100" Width="450" Margin="5,5">
        <Grid.RowDefinitions>
          <RowDefinition />
          <RowDefinition />
          <RowDefinition Height="30" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="150" />
          <ColumnDefinition />
          <ColumnDefinition Width="120" />
        </Grid.ColumnDefinitions>
        <Label Grid.Row="0" Grid.Column="1" HorizontalAlignment="Right" FontSize="16"
               Content="Price:" />
        <Label Grid.Row="1" Grid.Column="1" HorizontalAlignment="Right" FontSize="16"
               Content="Portion Size" />
        <TextBlock Grid.Row="0" Grid.Column="0" FontSize="18" Text="{Binding Name}" />
        <TextBlock Grid.Row="0" Grid.Column="2" HorizontalAlignment="Right" FontSize="16"
                   Text="{Binding Price, StringFormat='CHF \{0\}'}" />
        <TextBlock Grid.Row="1" Grid.Column="2" HorizontalAlignment="Right" FontSize="16"
                   Text="{Binding Portion, StringFormat='\{0\} g'}" />
        <Button Grid.Row="2" HorizontalAlignment="Left" Margin="5" Height="20" FontSize="12"
                VerticalAlignment="Center"
                BorderThickness="0" FontStyle="Oblique" Background="Transparent"
                BorderBrush="Transparent"
                Command="{Binding RelativeSource={RelativeSource FindAncestor,
                          AncestorType={x:Type Window}}, Path=DataContext.InformationCommand}"
                Content="More Information"
                Grid.Column="0" Width="100" />
        <Button Grid.Row="2" Grid.Column="1" BorderThickness="0"
                Grid.ColumnSpan="2" HorizontalAlignment="Right" Margin="5" Height="20"
                BorderBrush="Transparent" FontSize="12" Background="Transparent"
                VerticalAlignment="Center"
                Content="Add to muesli"
                Command="{Binding RelativeSource={RelativeSource FindAncestor,
                          AncestorType={x:Type Window}}, Path=DataContext.AddCommand}" />
      </Grid>
    </DataTemplate>
  </Window.Resources>
  <Grid Margin="10,10">
    <Grid.RowDefinitions>
      <RowDefinition Height="125" />
```

```
    <RowDefinition />
    <RowDefinition Height="75" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="500" />
    <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="75" />
      <RowDefinition />
    </Grid.RowDefinitions>
    <Label Grid.Row="0" FontSize="50" VerticalAlignment="Center"
          FontFamily="Comic Sans MS"
          HorizontalAlignment="Left" Content="Just Muesli" />
    <Label Grid.Row="1" FontSize="30" VerticalAlignment="Center"
          FontFamily="Comic Sans MS"
          HorizontalAlignment="Left" Content="Muesli Mixer" />
  </Grid>

  <TabControl Margin="5,5" x:Name="ContS" Grid.Row="1" Grid.Column="0"
            ItemsSource="{Binding Categories}"
            SelectedItem="{Binding SelectedTab}">
    <TabControl.ItemTemplate>
      <DataTemplate>
        <Label Content="{Binding Name}" HorizontalAlignment="Center"
             VerticalAlignment="Center" />
      </DataTemplate>
    </TabControl.ItemTemplate>
    <TabControl.ContentTemplate>
      <DataTemplate>
        <ListView
         ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor,
                    AncestorType={x:Type Window}}, Path=DataContext.IngredientList}"
         ItemTemplate="{StaticResource ListTemplate}"
         SelectedItem="{Binding RelativeSource={RelativeSource FindAncestor,
                          AncestorType={x:Type Window}},
Path=DataContext.SelectedIngredient}" />
      </DataTemplate>
    </TabControl.ContentTemplate>
  </TabControl>

  <Border Grid.Row="1" BorderThickness="1" BorderBrush="Black" Grid.Column="1"
        Margin="5,5" Grid.RowSpan="2">
    <Grid Margin="5,5">
      <Grid.RowDefinitions>
        <RowDefinition Height="50" />
        <RowDefinition />
        <RowDefinition Height="100" />
        <RowDefinition Height="25" />
        <RowDefinition Height="75" />
      </Grid.RowDefinitions>
      <Grid>
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="75" />
          <ColumnDefinition />
        </Grid.ColumnDefinitions>
```

```
  <Label Content="Name" FontSize="16" Grid.Row="0" Grid.Column="0"
         VerticalAlignment="Center"
         HorizontalAlignment="Center" />
  <TextBox Grid.Column="1" FontSize="16" HorizontalAlignment="Center"
          Width="160"
          Text="{Binding CerealName}" VerticalAlignment="Center" />
</Grid>
<ListView Margin="10,10" Grid.Row="1"
         SelectedItem="{Binding SelectedAddedIngredient}">
  <ListViewItem Template="{StaticResource AddedListTemplate}"
              DataContext="{Binding SelectedIngredientList[11]}"
              Height="25" />
  <ListViewItem Template="{StaticResource AddedListTemplate}"
              Height="25"
              DataContext="{Binding SelectedIngredientList[10]}" />
  <ListViewItem Template="{StaticResource AddedListTemplate}"
              Height="25"
              DataContext="{Binding SelectedIngredientList[9]}" />
  <ListViewItem Template="{StaticResource AddedListTemplate}"
              Height="25"
              DataContext="{Binding SelectedIngredientList[8]}" />
  <ListViewItem Template="{StaticResource AddedListTemplate}"
              Height="25"
              DataContext="{Binding SelectedIngredientList[7]}" />
  <ListViewItem Template="{StaticResource AddedListTemplate}"
              Height="25"
              DataContext="{Binding SelectedIngredientList[6]}" />
  <ListViewItem Template="{StaticResource AddedListTemplate}"
              Height="25"
              DataContext="{Binding SelectedIngredientList[5]}" />
  <ListViewItem Template="{StaticResource AddedListTemplate}"
              Height="25"
              DataContext="{Binding SelectedIngredientList[4]}" />
  <ListViewItem Template="{StaticResource AddedListTemplate}"
              Height="25"
              DataContext="{Binding SelectedIngredientList[3]}" />
  <ListViewItem Template="{StaticResource AddedListTemplate}"
              Height="25"
              DataContext="{Binding SelectedIngredientList[2]}" />
  <ListViewItem Template="{StaticResource AddedListTemplate}"
              Height="25"
              DataContext="{Binding SelectedIngredientList[1]}" />
  <ListViewItem Template="{StaticResource AddedListTemplate}"
              Height="40"
              DataContext="{Binding SelectedIngredientList[0]}" />
</ListView>
<Button Grid.Row="4" Content="Save muesli mix" VerticalAlignment="Bottom"
        Command="{Binding SaveCommand}"
        FontSize="20" Height="40" Background="IndianRed" Margin="10,10" />
<Grid Grid.Row="2" Margin="10,10">
  <Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition />
```

```
            <RowDefinition />
        </Grid.RowDefinitions>
        <Label HorizontalAlignment="Left" VerticalAlignment="Center"
               Content="TOTALS:" FontSize="12" />
        <Label HorizontalAlignment="Left" Grid.Row="1" Grid.Column="0"
               VerticalAlignment="Center"
               FontSize="13" Content="Price:" />
        <Label HorizontalAlignment="Left" Grid.Row="2" Grid.Column="0"
               VerticalAlignment="Center"
               FontSize="12" Content="Nutritional values:" />
        <TextBlock HorizontalAlignment="Right" Grid.Row="1" Grid.Column="1"
                   VerticalAlignment="Center"
                   FontSize="13" Text="{Binding TotalPrice,StringFormat='#.##'}" />
        <TextBlock HorizontalAlignment="Right" Grid.Row="2" Grid.Column="1"
                   VerticalAlignment="Center"
                   FontSize="12" Text="{Binding TotalNValues}" />
      </Grid>
      <Button Grid.Row="3" VerticalAlignment="Top" Background="IndianRed"
              Content="Details"
              Command="{Binding DetailsCommand}"
              HorizontalAlignment="Right"
              Width="100" Height="20" Margin="10,0" />
    </Grid>
  </Border>
  <Button Background="IndianRed" Height="40" Grid.Row="2" Grid.Column="0" Width="200"
          HorizontalAlignment="Left"
          VerticalAlignment="Center" Margin="20,0"
          Content="Back to menu" Command="{Binding BackCommand}" FontSize="20" />
 </Grid>
</Window>
```

CustomerDetailsView.xaml

```
<Window x:Class="MyMuesli.Views.CustomerDetailsView"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d"
        Title="CustomerDetailsView" ResizeMode="NoResize" Height="600" Width="600"
WindowStartupLocation="CenterScreen"
        DataContext="{Binding Customer, Source={StaticResource ViewModelLocator}}">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="125" />
            <RowDefinition />
            <RowDefinition Height="100" />
        </Grid.RowDefinitions>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="75" />
                <RowDefinition />
            </Grid.RowDefinitions>
            <Label Grid.Row="0" FontSize="50" VerticalAlignment="Center"
FontFamily="Comic Sans MS"
                   HorizontalAlignment="Center" Content="Just Muesli" />
            <Label Grid.Row="1" FontSize="30" VerticalAlignment="Center"
FontFamily="Comic Sans MS"
```

```
                HorizontalAlignment="Center" Content="Customer Details" />
        </Grid>
        <Grid Grid.Row="1" Margin="5,5">
            <Grid.RowDefinitions>
                <RowDefinition />
                <RowDefinition />
                <RowDefinition />
                <RowDefinition />
                <RowDefinition />
                <RowDefinition />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="150" />
                <ColumnDefinition Width="100" />
                <ColumnDefinition />
            </Grid.ColumnDefinitions>
            <Label Content="Name" Height="50" Width="150" VerticalAlignment="Center"
VerticalContentAlignment="Center"
                HorizontalContentAlignment="Center" HorizontalAlignment="Center"
FontSize="20" />
            <Label Grid.Row="1" Grid.Column="0" Content="Address" Height="50"
Width="150" VerticalAlignment="Center"
                VerticalContentAlignment="Center"
HorizontalContentAlignment="Center" HorizontalAlignment="Center"
                FontSize="20" />
            <Label Grid.Row="2" Grid.Column="0" Content="Zip / City" Height="50"
Width="150" VerticalAlignment="Center"
                VerticalContentAlignment="Center"
HorizontalContentAlignment="Center" HorizontalAlignment="Center"
                FontSize="20" />
            <Label Grid.Row="3" Grid.Column="0" Content="Country" Height="50"
Width="150" VerticalAlignment="Center"
                VerticalContentAlignment="Center"
HorizontalContentAlignment="Center" HorizontalAlignment="Center"
                FontSize="20" />
            <Label Grid.Row="4" Grid.Column="0" Content="Phone" Height="50"
Width="150" VerticalAlignment="Center"
                VerticalContentAlignment="Center"
HorizontalContentAlignment="Center" HorizontalAlignment="Center"
                FontSize="20" />
            <Label Grid.Row="5" Grid.Column="0" Content="Email" Height="50"
Width="150" VerticalAlignment="Center"
                VerticalContentAlignment="Center"
HorizontalContentAlignment="Center" HorizontalAlignment="Center"
                FontSize="20" />

            <TextBox Grid.Row="0" Text="{Binding Name}" Grid.Column="1" Margin="5,5"
Grid.ColumnSpan="2" />
            <TextBox Grid.Row="1" Text="{Binding Address}" Grid.Column="1"
Margin="5,5" Grid.ColumnSpan="2" />

            <TextBox Grid.Row="2" Text="{Binding Zip}" Grid.Column="1" Margin="5,5"
/>
            <TextBox Grid.Row="2" Text="{Binding City}" Grid.Column="2" Margin="5,5"
/>
```

```
            <ComboBox Grid.Row="3" Background="IndianRed" ItemsSource="{Binding
Countries}"
                      SelectedItem="{Binding SelectedCountry}"
                      Grid.Column="1" Margin="5,5" Grid.ColumnSpan="2">
                <ComboBox.ItemTemplate>
                    <DataTemplate>
                        <Label Content="{Binding Name}" />
                    </DataTemplate>
                </ComboBox.ItemTemplate>
            </ComboBox>
            <TextBox Grid.Row="4" Text="{Binding Phone}" Grid.Column="1" Margin="5,5"
Grid.ColumnSpan="2" />
            <TextBox Grid.Row="5" Text="{Binding Email}" Grid.Column="1" Margin="5,5"
Grid.ColumnSpan="2" />
        </Grid>
        <Grid Grid.Row="2">
            <Grid.ColumnDefinitions>
                <ColumnDefinition />
                <ColumnDefinition />
            </Grid.ColumnDefinitions>
            <Button Height="40" Width="200" Background="IndianRed" Content="Save"
Command="{Binding SaveCommand}"
                    FontSize="20" />
            <Button Height="40" Grid.Column="1" Background="IndianRed"
Command="{Binding MenuCommand}" Width="200"
                    Content="Back to menu"
                    FontSize="20" />
        </Grid>
    </Grid>
</Window>
```

DetailsView.xaml

```
<Window x:Class="MyMuesli.Views.DetailsView"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:MyMuesli.Views"
        mc:Ignorable="d"
        Title="DetailsView" Height="300" Width="400">
  <Grid Margin="10,10">
    <Grid.RowDefinitions>
      <RowDefinition Height="40" />
      <RowDefinition />
      <RowDefinition Height="40" />
    </Grid.RowDefinitions>
    <Label Content="Energy Content per 100g of the Current Muesli"
VerticalAlignment="Center" />
    <Border Grid.Row="1" BorderThickness="1" BorderBrush="Black">
      <Grid ShowGridLines="True" Margin="5,5">
        <Grid.RowDefinitions>
          <RowDefinition />
          <RowDefinition />
          <RowDefinition />
          <RowDefinition />
        </Grid.RowDefinitions>
```

```
        <Grid.ColumnDefinitions>
          <ColumnDefinition />
          <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Label Content="Energy Content" HorizontalAlignment="Left"
VerticalAlignment="Center" FontFamily="16" />
        <Label Content="Protein" Grid.Row="1" Grid.Column="0"
HorizontalAlignment="Left"
               VerticalAlignment="Center" FontFamily="16" />
        <Label Content="Carbohydrates" Grid.Row="2" Grid.Column="0"
HorizontalAlignment="Left"
               VerticalAlignment="Center" FontFamily="16" />
        <Label Content="Fat" Grid.Row="3" Grid.Column="0" HorizontalAlignment="Left"
VerticalAlignment="Center"
               FontFamily="16" />

        <TextBlock Text="{Binding EnergyContent,StringFormat='\{0\} g'}" Grid.Row="0"
Grid.Column="1"
                   HorizontalAlignment="Right" VerticalAlignment="Center"
FontFamily="16" />
        <TextBlock Text="{Binding Protein,StringFormat='\{0\} g'}" Grid.Row="1"
Grid.Column="1"
                   HorizontalAlignment="Right" VerticalAlignment="Center"
FontFamily="16" />
        <TextBlock Text="{Binding Carbohydrates,StringFormat='\{0\} g'}" Grid.Row="2"
Grid.Column="1"
                   HorizontalAlignment="Right" VerticalAlignment="Center"
FontFamily="16" />
        <TextBlock Text="{Binding Fat,StringFormat='\{0\} g'}" Grid.Row="3"
Grid.Column="1"
                   HorizontalAlignment="Right" VerticalAlignment="Center"
FontFamily="16" />
      </Grid>
    </Border>
    <Button Grid.Row="2" Content="Back" Background="IndianRed" Click="Close"
Width="100" VerticalAlignment="Center"
            HorizontalAlignment="Right" />
  </Grid>
</Window>
```

InformationView.xaml

```
<Window x:Class="MyMuesli.Views.InformationView"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d"
        Title="InformationView" Height="500" Width="500">
  <Grid Margin="10,10">
    <Grid.RowDefinitions>
      <RowDefinition Height="50" />
      <RowDefinition Height="30" />
      <RowDefinition Height="120" />
      <RowDefinition Height="50" />
      <RowDefinition Height="150" />
      <RowDefinition Height="50" />
```

```
    </Grid.RowDefinitions>
    <Label Grid.Row="0" Height="50" VerticalAlignment="Center"
HorizontalAlignment="Left" Width="200"
           Content="{Binding Name}" FontSize="26" />
    <Label Grid.Row="1" Content="Ingredients" HorizontalAlignment="Left"
VerticalAlignment="Center" FontFamily="16" />
    <Border Grid.Row="2" BorderBrush="Black" BorderThickness="1">
      <TextBlock Text="{Binding IngredientDescription}" />
    </Border>
    <Label Grid.Row="3" Content="Energy Content per 100g" HorizontalAlignment="Left"
VerticalAlignment="Center"
           FontFamily="16" />
    <Border Grid.Row="4" BorderThickness="1" BorderBrush="Black">
      <Grid ShowGridLines="True" Margin="5,5">
        <Grid.RowDefinitions>
          <RowDefinition />
          <RowDefinition />
          <RowDefinition />
          <RowDefinition />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
          <ColumnDefinition />
          <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Label Content="Energy Content" HorizontalAlignment="Left"
VerticalAlignment="Center" FontFamily="16" />
        <Label Content="Protein" Grid.Row="1" Grid.Column="0"
HorizontalAlignment="Left"
               VerticalAlignment="Center" FontFamily="16" />
        <Label Content="Carbohydrates" Grid.Row="2" Grid.Column="0"
HorizontalAlignment="Left"
               VerticalAlignment="Center" FontFamily="16" />
        <Label Content="Fat" Grid.Row="3" Grid.Column="0" HorizontalAlignment="Left"
VerticalAlignment="Center"
               FontFamily="16" />

        <TextBlock Text="{Binding EnergyContent}" Grid.Row="0" Grid.Column="1"
HorizontalAlignment="Right"
                   VerticalAlignment="Center" FontFamily="16" />
        <TextBlock Text="{Binding Protein , StringFormat='\{0\} g'}" Grid.Row="1"
Grid.Column="1"
                   HorizontalAlignment="Right" VerticalAlignment="Center"
FontFamily="16" />
        <TextBlock Text="{Binding Carbohydrates, StringFormat='\{0\} g'}"
Grid.Row="2" Grid.Column="1"
                   HorizontalAlignment="Right" VerticalAlignment="Center"
FontFamily="16" />
        <TextBlock Text="{Binding Fat, StringFormat='\{0\} g'}" Grid.Row="3"
Grid.Column="1"
                   HorizontalAlignment="Right" VerticalAlignment="Center"
FontFamily="16" />
      </Grid>
    </Border>
    <Button Grid.Row="5" Content="Back" Background="IndianRed" Click="Close"
HorizontalAlignment="Right"
            Width="100" VerticalAlignment="Center" />
```

```
  </Grid>
</Window>
```

MyCerealsMixesView.xaml

```
<Window x:Class="MyMuesli.Views.MyCerealMixesView"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:MyMuesli.Views"
        mc:Ignorable="d"
        Title="MyCerealMixesView" Height="500" ResizeMode="NoResize" Width="500"
WindowStartupLocation="CenterScreen"
        DataContext="{Binding MyCereal, Source={StaticResource ViewModelLocator}}">
  <Grid Margin="10,10">
    <Grid.RowDefinitions>
      <RowDefinition Height="125" />
      <RowDefinition />
      <RowDefinition Height="60" />
    </Grid.RowDefinitions>
    <Grid>
      <Grid.RowDefinitions>
        <RowDefinition Height="75" />
        <RowDefinition />
      </Grid.RowDefinitions>
      <Label Grid.Row="0" FontSize="50" VerticalAlignment="Center" FontFamily="Comic
Sans MS"
             HorizontalAlignment="Center" Content="Just Muesli" />
      <Label Grid.Row="1" FontSize="30" VerticalAlignment="Center" FontFamily="Comic
Sans MS"
             HorizontalAlignment="Center" Content="My muesli mixes" />
    </Grid>
    <DataGrid Grid.Row="1" IsReadOnly="True" AutoGenerateColumns="False"
SelectedItem="{Binding SelectedCereal}"
              AlternationCount="2" ItemsSource="{Binding MyCereals}">
      <DataGrid.Columns>
        <DataGridTextColumn FontSize="16" Header="Name" Binding="{Binding Name}"
Width="175" />
        <DataGridTextColumn FontSize="16" Header="Price / 600g" Binding="{Binding
Price}" Width="175" />
        <DataGridTextColumn FontSize="16" Header="Created on" Binding="{Binding
CreatedOn}" />
      </DataGrid.Columns>
    </DataGrid>
    <Grid Grid.Row="2">
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="200" />
        <ColumnDefinition />
        <ColumnDefinition />
      </Grid.ColumnDefinitions>
      <Button Height="40" Width="150" HorizontalAlignment="Left" FontSize="16"
Background="IndianRed"
              Content="Back to menu" Command="{Binding MenuCommand}" />
      <Button Height="40" Grid.Column="1" HorizontalAlignment="Right" FontSize="16"
Background="IndianRed"
              Command="{Binding EditCommand}" Width="125" Content="Edit" />
```

```
      <Button Height="40" Grid.Column="2" HorizontalAlignment="Right" FontSize="16"
Background="IndianRed"
              Command="{Binding DeleteCommand}" Width="125" Content="Delete" />
    </Grid>
  </Grid>
</Window>
```

ViewModels

MyCerealViewModel.cs

```
public class MyCerealViewModel : ViewModelBase
{
    public readonly Cereal Cereal;

    public MyCerealViewModel(Cereal c)
    {
        Cereal = c;
    }
    public string Name => Cereal.Name;
    public string Price => "CHF " + Cereal.Price;
    public string CreatedOn => Cereal.CreatedOn.ToString("dd.MM.yyyy");
}
```

CerealViewModel.cs

```
public class CerealViewModel : ViewModelBase
{
    private const double Divider = 6.0;
    private const double PortionDivider = 100.0;
        private readonly ObservableCollection<IngredientViewModel>
        _selectedIngredientList;
    public readonly Cereal Cereal;
    private bool _isXXl;
    private int _quantity;
    private double _total;

        public CerealViewModel(ObservableCollection<IngredientViewModel>
        selectedIngredientList)
    {
        _selectedIngredientList = selectedIngredientList;
        Protein = CalculateProtein();
        Carbohydrates = CalculateCarbohydrates();
        Fat = CalculateFat();
              EnergyContent =
              CerealContentCalculator.CalculateCalsAndJouleForMultiple(selectedIngredi
              entList);
        IncreaseCommand = new RelayCommand(() => Quantity++);
        DecreaseCommand = new RelayCommand(() => Quantity--);
    }

    public CerealViewModel(Cereal cereal)
    {
        Cereal = cereal;
    }

    public ICommand IncreaseCommand { get; set; }
    public ICommand DecreaseCommand { get; set; }
```

```
public bool IsXXl
{
    get => _isXXl;
    set
    {
        _isXXl = value;
        CheckNewTotal();
        RaisePropertyChanged();
    }
}

public int Quantity
{
    get => _quantity;
    set
    {
        _quantity = value;
        CheckNewTotal();
        RaisePropertyChanged();
    }
}

public double Total
{
    get => _total;
    set
    {
        _total = value;
        RaisePropertyChanged();
    }
}

public string Name
{
    get => Cereal.Name;
    set
    {
        Cereal.Name = value;
        RaisePropertyChanged();
    }
}

public double Price
{
    get => Cereal.Price;
    set
    {
        Cereal.Price = value;
        RaisePropertyChanged();
    }
}

public string CreatedOn =>
Cereal.CreatedOn.ToString(CultureInfo.InvariantCulture);

public string EnergyContent { get; set; }
```

```
    public double Protein { get; set; }
    public double Carbohydrates { get; set; }
    public double Fat { get; set; }

    private void CheckNewTotal()
    {
        if (IsXXl) Price = Price * 4.0;
        Total = Price * Quantity;
    }

    private double CalculateFat()
    {
        var sum = _selectedIngredientList.Select(CalculateFat).Sum();
        return sum / Divider;
    }

    private static double CalculateFat(IngredientViewModel ingredientVm)
    {
        var var = ingredientVm.Ingredient.Fat / PortionDivider;
        return var * ingredientVm.Ingredient.Portion;
    }

    private double CalculateCarbohydrates()
    {
        var sum = _selectedIngredientList.Select(CalculateCarbohydrates).Sum();
        return sum / Divider;
    }

        private static double CalculateCarbohydrates(IngredientViewModel
        ingredientVm)
    {
        var var = ingredientVm.Ingredient.Carbohydrates / PortionDivider;
        return var * ingredientVm.Ingredient.Portion;
    }

    private static double CalculateProtein(IngredientViewModel ingredientVm)
    {
        var var = ingredientVm.Ingredient.Protein / PortionDivider;
        return var * ingredientVm.Ingredient.Portion;
    }

    private double CalculateProtein()
    {
        var sum = _selectedIngredientList.Select(CalculateProtein).Sum();
        return sum / Divider;
    }
}
```

CustomerDetailsViewModel.cs

```
    public class CustomerDetailsViewModel : ViewModelBase
        {
            private static readonly ILog Log =
    LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);
            private readonly CustomerDetails _customerDetails = new CustomerDetails();
            private readonly IDatabaseService _databaseService;
            public CustomerDetailsViewModel(IDatabaseService service)
```

```
{
    _databaseService = service;

    Countries = _databaseService.GetCountries();

    SaveCommand = new RelayCommand(SaveUser);
    MenuCommand = new RelayCommand(BackToMenu);
}

public ICommand SaveCommand { get; set; }
public ICommand MenuCommand { get; set; }

public string Name
{
    get => _customerDetails.Name;
    set
    {
        _customerDetails.Name = value;
        RaisePropertyChanged();
    }
}

public string Address
{
    get => _customerDetails.Address;
    set
    {
        _customerDetails.Address = value;
        RaisePropertyChanged();
    }
}

public string Zip
{
    get => _customerDetails.Zip;
    set
    {
        _customerDetails.Zip = value;
        RaisePropertyChanged();
    }
}

public string City
{
    get => _customerDetails.City;
    set
    {
        _customerDetails.City = value;
        RaisePropertyChanged();
    }
}

public ObservableCollection<Country> Countries { get; set; }

public Country SelectedCountry
{
```

```
            get => _customerDetails.Country;
            set
            {
                _customerDetails.Country = value;
                RaisePropertyChanged();
            }
        }

        public string Phone
        {
            get => _customerDetails.Phone;
            set
            {
                _customerDetails.Phone = value;
                RaisePropertyChanged();
            }
        }

        public string Email
        {
            get => _customerDetails.Email;
            set
            {
                _customerDetails.Email = value;
                RaisePropertyChanged();
            }
        }

        private bool ValidateCustomer()
        {
            if (CustomerValidation.ValidateCustomer(_customerDetails)) return true;

            MessageBox.Show(CustomerValidation.WrongField, "Wrong Input!");
            return false;
        }

        private void BackToMenu()
        {
            var currentView = Application.Current.Windows[0];
            new MainWindow().Show();
            currentView?.Close();
        }

        private void SaveUser()
        {
            if (ValidateCustomer())
            {
                _databaseService.AddUser(_customerDetails);
                ViewModelLocator.Instance.InitCustomer(_customerDetails);
                MessageBox.Show("Customer has been saved!", "info");
                BackToMenu();
            }
        }
    }
```

CerealMixerViewModel.cs

```
public class CerealMixerViewModel : ViewModelBase
  {
    private const int MaxMuesliPortion = 600;
    private const string VBasic = "Basics";

    private static readonly ILog Log = LogManager.GetLogger(MethodBase
      .GetCurrentMethod()
      .DeclaringType);

    private readonly IDatabaseService _databaseService;
    private readonly ObservableCollection<Ingredient> _ingredients;
    private readonly IAppSession _session;
    private string _cerealName;
    private ObservableCollection<IngredientViewModel> _ingredientList;

    private ObservableCollection<IngredientViewModel> _selectedIngredientList =
      new ObservableCollection<IngredientViewModel>();

    private string _totalNValues;
    private string _totalPrice;

    public CerealMixerViewModel(IDatabaseService databaseService,
      IAppSession session)
    {
      _databaseService = databaseService;
      _session = session;
      _ingredients = _databaseService.GetIngredients();
      Categories = _databaseService.GetCategories();

      InformationCommand = new RelayCommand(OpenInformation);
      AddCommand = new RelayCommand(AddIngredient);
      BackCommand = new RelayCommand(BackToMenu);
      SaveCommand = new RelayCommand(SaveCereal);
      DetailsCommand = new RelayCommand(OpenDetails);
      RemoveCommand = new RelayCommand(RemoveIngredient);
      ChangeIngredientCategoryList("Basics");

      CheckForEditableMuesli();
    }

    public ICommand SaveCommand { get; set; }
    public ICommand DetailsCommand { get; set; }
    public ICommand InformationCommand { get; set; }
    public ICommand BackCommand { get; set; }
    public ICommand AddCommand { get; set; }
    public ICommand RemoveCommand { get; set; }

    public string CerealName
    {
      get => _cerealName;
      set
      {
        _cerealName = value;
        RaisePropertyChanged();
      }
```

```
    }

    public IngredientViewModel SelectedIngredient { get; set; }
    public IngredientViewModel SelectedAddedIngredient { get; set; }
    public ObservableCollection<Category> Categories { get; set; }

    public ObservableCollection<IngredientViewModel> IngredientList
    {
      get => _ingredientList;
      set
      {
        _ingredientList = value;
        RaisePropertyChanged();
      }
    }

    public string TotalPrice
    {
      get => _totalPrice;
      set
      {
        _totalPrice = value;
        RaisePropertyChanged();
      }
    }

    public Category SelectedTab
    {
      set => ChangeIngredientCategoryList(value.Name);
    }

    public string TotalNValues
    {
      get => _totalNValues;
      set
      {
        _totalNValues = value;
        RaisePropertyChanged();
      }
    }

    public ObservableCollection<IngredientViewModel> SelectedIngredientList
    {
      get => _selectedIngredientList;
      set
      {
        _selectedIngredientList = value;
        RaisePropertyChanged();
      }
    }

    private void CheckForEditableMuesli()
    {
      if (_session.EditableCereal != null)
        SetSelectedCereal(_session
          .EditableCereal);
```

```
    }

    public void SetSelectedCereal(Cereal cereal)
    {
      CerealName = cereal.Name;
      SelectedIngredientList = new ObservableCollection<IngredientViewModel>(
        _databaseService
          .GetIngredientList(cereal).Select(t => new IngredientViewModel(t)));
      CalculateCerealValues();
    }

    private void OpenDetails()
    {
      if (SelectedIngredientList != null)
      {
        var vm = new CerealViewModel(
          new ObservableCollection<IngredientViewModel>(SelectedIngredientList
            .Select(t => t)));
        var view = new DetailsView(vm);
        view.Show();
      }
    }

    private void SaveCereal()
    {
      if (SelectedIngredientList.Count > 0 && !string.IsNullOrWhiteSpace
            (CerealName))
      {
        var cereal = new Cereal
        {
          CerealId = _session.EditableCereal.CerealId,
          CreatedOn = DateTime.Now,
          Customer = _session.Customer,
          Name = CerealName,
          Price = CerealContentCalculator.CalculatePrice
            (SelectedIngredientList),
          Ingredients = SelectedIngredientList.Select(i => i.Ingredient)
            .ToList()
        };
        if (_session.EditableCereal != null)
          _databaseService.UpdateCereal(cereal);
        else
          _databaseService.AddCereal(cereal);

        MessageBox.Show("Muesli Created!", "Info");
        BackToMenu();
      }
      else
      {
        MessageBox.Show("Muesli must contain at least 1 Ingredient and have a valid"
+
                        " Name", "Error");
      }
    }

    public void OpenInformation()
```

```
{
  if (SelectedIngredient != null)
  {
    var vm = new IngredientViewModel(SelectedIngredient.Ingredient);
    var iv = new InformationView(vm);
    iv.Show();
  }
}

public void RemoveIngredient()
{
  var dialogResult =
    MessageBox.Show("Are you sure you want to remove the selected ingredient?",
      "Warning",
      MessageBoxButton.OKCancel);

  if (dialogResult == MessageBoxResult.OK)
  {
    SelectedIngredientList.Remove(SelectedAddedIngredient);
    CalculateCerealValues();
    CalculateBasicComponent();
  }
}

public void AddIngredient()
{
  if (SelectedIngredient != null)
  {
    if (!BasicHasBeenChosen() && SelectedIngredient.Ingredient.Category.Name
          .Equals(VBasic))
    {
      AddIngredientToList();
    }
    else if (BasicHasBeenChosen() && MaxPortionNotReached()
                                  && SelectedIngredientList
                                    .Count < 12 &&
                                  !SelectedIngredient.Ingredient.Category
                                    .Name.Equals(VBasic))
    {
      AddIngredientToList();
      CalculateBasicComponent();
    }
    else if (BasicHasBeenChosen() && SelectedIngredient.Ingredient.Category
               .Name
               .Equals(VBasic))
    {
      SelectedIngredientList[0] = GetIngredientCopy();
      CalculateBasicComponent();
    }
    else
    {
      MessageBox.Show(
        "Rules: Basic Component has been chosen first /" +
        " Max. Ingredient = 12 /" +
        " Only 1 Basic / Max  600g",
        "Error");
```

```
        }

        CalculateCerealValues();
      }
    }

    private void AddIngredientToList()
    {
      SelectedIngredientList.Add(GetIngredientCopy());
    }

    private IngredientViewModel GetIngredientCopy()
    {
      var a = SelectedIngredient.Ingredient;
      return new IngredientViewModel(new Ingredient
      {
        Carbohydrates = a.Carbohydrates,
        Category = a.Category,
        Fat = a.Fat,
        IngredientId = a.IngredientId,
        IngredientDescription = a.IngredientDescription,
        Name = a.Name,
        Portion = a.Portion,
        Price = a.Price,
        Protein = a.Protein
      });
    }

    private bool MaxPortionNotReached()
    {
      var totalPortionAdded = SelectedIngredientList.Where(t => !t.Ingredient
          .Category
          .Name.Equals(VBasic))
        .Select(t => t.Ingredient.Portion).Sum();
      return totalPortionAdded + SelectedIngredient.Ingredient.Portion
             <= MaxMuesliPortion;
    }

    private void CalculateBasicComponent()
    {
      var totalPortionAdded = SelectedIngredientList.Where(i => !i.Ingredient
          .Category
          .Name.Equals(VBasic))
        .Select(t => t.Ingredient.Portion).Sum();
      var difference = MaxMuesliPortion - totalPortionAdded;

      SelectedIngredientList[0].Price = CalculatePrice(difference);
      SelectedIngredientList[0].Portion = difference;
    }

    private double CalculatePrice(int difference)
    {
      var before = SelectedIngredientList[0].Ingredient.Price;
      return before / SelectedIngredientList[0].Ingredient.Portion * difference;
    }
```

```
    private void CalculateCerealValues()
    {
      TotalPrice = CalculateTot();
      TotalNValues = CerealContentCalculator.CalculateCalsMultiple
        (SelectedIngredientList);
    }

    private string CalculateTot()
    {
      var total = SelectedIngredientList.Select(t => t.Ingredient.Price).Sum();
      return Math.Round(total, 2) + " CHF / 600g";
    }

    private bool BasicHasBeenChosen()
    {
      return SelectedIngredientList.Any(i => i.Ingredient.Category.Name
        .Equals("Basics"));
    }

    private void ChangeIngredientCategoryList(string value)
    {
      if (_ingredients != null)
        IngredientList =
          new ObservableCollection<IngredientViewModel>(_ingredients
            .Where(i => i.Category
              .Name.Equals(value))
            .Select(t => new IngredientViewModel(t)));
    }

    private static void BackToMenu()
    {
      var currentView = Application.Current.Windows[0];
      new MainWindow().Show();
      currentView?.Close();
    }
  }
```

IngredientViewModel.cs

```
public class IngredientViewModel : ViewModelBase
    {
        private string _energyContent;
        public Ingredient Ingredient;

        public IngredientViewModel(Ingredient selectedIngredient)
        {
            Ingredient = selectedIngredient;
            EnergyContent = CalculateEnergyContent();
        }

        public string Name
        {
            get => Ingredient.Name;
            set
            {
                Ingredient.Name = value;
                RaisePropertyChanged();
```

```
        }
    }

    public int Portion
    {
        get => Ingredient.Portion;
        set
        {
            Ingredient.Portion = value;
            RaisePropertyChanged();
        }
    }

    public double Price
    {
        get => Ingredient.Price;
        set
        {
            Ingredient.Price = value;
            RaisePropertyChanged();
        }
    }

    public string IngredientDescription
    {
        get => Ingredient.IngredientDescription;
        set
        {
            Ingredient.IngredientDescription = value;
            RaisePropertyChanged();
        }
    }

    public string EnergyContent
    {
        get => _energyContent;
        set
        {
            _energyContent = value;
            RaisePropertyChanged();
        }
    }

    public double Protein
    {
        get => Ingredient.Protein;
        set
        {
            Ingredient.Protein = value;
            RaisePropertyChanged();
        }
    }

    public double Fat
    {
        get => Ingredient.Fat;
```

```
            set
            {
                Ingredient.Fat = value;
                RaisePropertyChanged();
            }
        }

        public double Carbohydrates
        {
            get => Ingredient.Carbohydrates;
            set
            {
                Ingredient.Carbohydrates = value;
                RaisePropertyChanged();
            }
        }

        private string CalculateEnergyContent()
        {
            return CerealContentCalculator.CalculateCalsAndJoule(this);
        }
    }
```

MainViewModel.cs

```
    public class MainViewModel : ViewModelBase
    {
        private static readonly ILog Log =
        LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);
        private readonly IDatabaseService _databaseService;
        private bool _isCustomerCreated;
        private bool _isMuesliCreated;

        public MainViewModel(IDatabaseService databaseService)
        {
            XmlConfigurator.Configure(new FileInfo("Logger\\log4net.config"));

            _databaseService = databaseService;
            ExitCommand = new RelayCommand(() => Environment.Exit(0));
            CustomerDetailsCommand = new RelayCommand(OpenCustomerDetails);
            CerealMixerCommand = new RelayCommand(OpenCerealMixer);
            MyCerealCommand = new RelayCommand(OpenMyCereals);
            OrderCommand = new RelayCommand(OpenOrders);

            IsCustomerCreated = CheckForCustomer();
            IsMuesliCreated = CheckForMuesli();
        }

        public bool IsMuesliCreated
        {
            get => _isMuesliCreated;
            set
            {
                _isMuesliCreated = value;
                RaisePropertyChanged();
            }
        }
```

```
public ICommand ExitCommand { get; set; }
public ICommand CustomerDetailsCommand { get; set; }
public ICommand CerealMixerCommand { get; set; }
public ICommand MyCerealCommand { get; set; }
public ICommand OrderCommand { get; set; }

public bool IsCustomerCreated
{
    get => _isCustomerCreated;
    set
    {
        _isCustomerCreated = value;
        RaisePropertyChanged();
    }
}

private bool CheckForMuesli()
{
    try
    {
      var session =
      ViewModelLocator.Instance.Container.Resolve<IAppSession>();
         if (session.Customer != null) return
      _databaseService.GetMyCereals(session.Customer).Count > 0;
    }
    catch
    {
        Log.Warn("No Muesli created");
    }

    return false;
}

private bool CheckForCustomer()
{
    try
    {
        var session =
        ViewModelLocator.Instance.Container.Resolve<IAppSession>();
        if (session.Customer != null) return true;
    }
    catch
    {
        Log.Warn("No Customer created");
    }

    return false;
}

private void OpenOrders()
{
    var orderView = new OrderView();
    orderView.Show();
    CloseMain();
}
```

```
        private static void CloseMain()
        {
            var main = Application.Current.Windows[0];
            main?.Close();
        }

        private void OpenMyCereals()
        {
            var orderView = new MyCerealMixesView();
            orderView.Show();
            CloseMain();
        }

        private void OpenCerealMixer()
        {
            var orderView = new CerealMixerView();
            orderView.Show();
            CloseMain();
        }

        private void OpenCustomerDetails()
        {
            var orderView = new CustomerDetailsView();
            orderView.Show();
            CloseMain();
        }
    }
```

MyCerealMixesViewModel.cs

```
    public class MyCerealMixesViewModel : ViewModelBase
      {
        private static readonly ILog Log =
    LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);
        private readonly IDatabaseService _databaseService;
        private readonly IAppSession _session;
        private ObservableCollection<MyCerealViewModel> _myCereals;

        public MyCerealMixesViewModel(IDatabaseService databaseService, IAppSession
    session)
        {
          _databaseService = databaseService;
          _session = session;
          _myCereals =
    CreateViewModelFromCereal(_databaseService.GetMyCereals(_session.Customer));
          MenuCommand = new RelayCommand(BackToMenu);
          DeleteCommand = new RelayCommand(Delete);
          EditCommand = new RelayCommand(EditCereal);
        }

        public ICommand EditCommand { get; set; }
        public ICommand DeleteCommand { get; set; }
        public ICommand MenuCommand { get; set; }

        public MyCerealViewModel SelectedCereal { get; set; }
```

```
    public ObservableCollection<MyCerealViewModel> MyCereals
    {
      get => _myCereals;
      set
      {
        _myCereals = value;
        RaisePropertyChanged();
      }
    }

    private static ObservableCollection<MyCerealViewModel> CreateViewModelFromCereal(
      IEnumerable<Cereal> getMyCereals)
    {
      return new ObservableCollection<MyCerealViewModel>(getMyCereals.Select(c => new
MyCerealViewModel(c)));
    }

    private void EditCereal()
    {
      if (SelectedCereal != null)
      {
        var currentView = Application.Current.Windows[0];
        _session.EditableCereal = SelectedCereal.Cereal;
        new CerealMixerView().Show();
        currentView?.Close();
      }
      else
      {
        MessageBox.Show("Please select a Muesli", "Warn");
        Log.Warn("No Muesli Selected!");
      }
    }

    private void Delete()
    {
      if (SelectedCereal != null)
      {
        var result = MessageBox.Show("Are you sure you want to delete the selected
Muesli?", "Warning",
          MessageBoxButton.OKCancel);

        if (result == MessageBoxResult.OK)
        {
          _databaseService.DeleteMuesli(SelectedCereal.Cereal);
          MyCereals.Remove(SelectedCereal);
        }
      }
      else
      {
        MessageBox.Show("Please select a Muesli", "Warn");
        Log.Warn("No Muesli Selected!");
      }
    }

    private void BackToMenu()
    {
```

```
            var currentView = Application.Current.Windows[0];
            new MainWindow().Show();
            currentView?.Close();
        }
    }
```

OrderViewModel.cs

```
        public class OrderViewModel : ViewModelBase
        {
            private static readonly ILog Log =
    LogManager.GetLogger(MethodBase.GetCurrentMethod()
                .DeclaringType);
            private readonly IDatabaseService _databaseService;
            private readonly IAppSession _session;
            private double _grandTotal;
            private bool _isOrderEnabled;
            private double _muesliMixesValues;
            private ObservableCollection<CerealViewModel> _myCereals;
            private double _shippingPrice;
            private double _taxesValue;


            public OrderViewModel(IDatabaseService databaseService, IAppSession session)
            {
                _session = session;
                _databaseService = databaseService;

                MyCereals = GetCereals();

                SaveCommand = new RelayCommand(Save);
                MenuCommand = new RelayCommand(BackToMenu);

                CheckQuantity();
                CalculateValues();
            }

            public bool IsOrderEnabled
            {
                get => _isOrderEnabled;
                set
                {
                    _isOrderEnabled = value;
                    RaisePropertyChanged();
                }
            }

            public double MuesliMixesValues
            {
                get => _muesliMixesValues;
                set

                {
                    _muesliMixesValues = value;
                    RaisePropertyChanged();
                }
            }
```

```
public double ShippingPrice
{
    get => _shippingPrice;
    set
    {
        _shippingPrice = value;
        RaisePropertyChanged();
    }
}


public double TaxesValue
{
    get => _taxesValue;
    set
    {
        _taxesValue = value;
        RaisePropertyChanged();
    }
}

public double GrandTotal
{
    get => _grandTotal;
    set
    {
        _grandTotal = value;
        RaisePropertyChanged();
    }
}

public ObservableCollection<CerealViewModel> MyCereals
{
    get => _myCereals;
    set
    {
        _myCereals = value;
        RaisePropertyChanged();
    }
}

public ICommand SaveCommand { get; set; }
public ICommand MenuCommand { get; set; }

private void CheckQuantity()
{
    if (_myCereals.Any(c => c.Quantity > 0)) IsOrderEnabled = true;
    IsOrderEnabled = false;
}

private void CalculateValues()
{
    MuesliMixesValues = GetMuesliPrices();
    ShippingPrice = GetShippingCost();
    TaxesValue = GetTax();
```

```
            GrandTotal = MuesliMixesValues + TaxesValue + ShippingPrice;
        }

        private double GetTax()
        {
            if (IsSwiss()) return MuesliMixesValues * 0.025;

            return 0.00;
        }

        private bool IsSwiss()
        {
            return _session.Customer.Country.Name.Equals("Switzerland");
        }

        private double GetShippingCost()
        {
            if (MuesliMixesValues > 50.00) return 0.00;

            return IsSwiss() ? 6.00 : 8.00;
        }

        private double GetMuesliPrices()
        {
            return MyCereals.Select(t => t.Price).Sum();
        }

        private ObservableCollection<CerealViewModel> GetCereals()
        {
            return new ObservableCollection<CerealViewModel>(_databaseService
                .GetMyCereals(_session.Customer)
                .Select(t => new CerealViewModel(t)));
        }

        private static void BackToMenu()
        {
            var currentView = Application.Current.Windows[0];
            new MainWindow().Show();
            currentView?.Close();
        }

        private static void Save()
        {
            MessageBox.Show("Thank you for Ordering! Your Order will soon be" +
                            " delivered");
            Log.Info("Order was  saved!");
        }
    }
```

Models

Category.cs

```
public class Category
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
```

```
        public int CategoryId { get; set; }

        public string Name { get; set; }
    }
```

Cereal.cs

```
    public class Cereal
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int CerealId { get; set; }
        public string Name { get; set; }
        public CustomerDetails Customer { get; set; }
        public DateTime CreatedOn { get; set; }
        public double Price { get; set; }
        public virtual ICollection<Ingredient> Ingredients { get; set; }
    }
```

Country.cs

```
    public class Country
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int CountryId { get; set; }
        public string Name { get; set; }
    }
```

CustomerDetails.cs

```
    public class CustomerDetails
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int CustomerDetailsId { get; set; }
        public string Name { get; set; }
        public string Address { get; set; }
        public string City { get; set; }
        public string Zip { get; set; }
        public Country Country { get; set; }
        public string Phone { get; set; }
        public string Email { get; set; }
        public virtual ICollection<Cereal> Cereals { get; set; }
    }
```

Ingredient.cs

```
    public class Ingredient
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int IngredientId { get; set; }
        public string Name { get; set; }
        public int Portion { get; set; }
        public double Price { get; set; }
        public string IngredientDescription { get; set; }
        public double Protein { get; set; }
        public double Fat { get; set; }
        public double Carbohydrates { get; set; }
        public Category Category { get; set; }
        public virtual ICollection<Cereal> Cereals { get; set; }
```

```
}
```

Service & Helper

IAppSession.cs

```
public interface IAppSession
{
    CustomerDetails Customer { get; set; }
    Cereal EditableCereal { get; set; }
}
```

IDatabaseService.cs

```
public interface IDatabaseService
{
    void AddUser(CustomerDetails customer);
    ObservableCollection<Country> GetCountries();
    ObservableCollection<Cereal> GetMyCereals(CustomerDetails customerDetails);
    ObservableCollection<Ingredient> GetIngredients();
    ObservableCollection<Category> GetCategories();
    void AddCereal(Cereal cereal);
    void DeleteMuesli(Cereal selectedCereal);
    ObservableCollection<Ingredient> GetIngredientList(Cereal cereal);
    void UpdateCereal(Cereal cereal);
}
```

AppSession.cs

```
public class AppSession : IAppSession
{
   public AppSession(CustomerDetails selectedCustomer)
    {
        Customer = selectedCustomer;
    }
    public CustomerDetails Customer { get; set; }
    public Cereal EditableCereal { get; set; }
}
```

DatabaseService.cs

```
public class DatabaseService : IDatabaseService
    {
        private static readonly ILog Log =
LogManager.GetLogger(MethodBase.GetCurrentMethod()
            .DeclaringType);
        public void AddUser(CustomerDetails customer)
        {
            using (var ctx = new MyCerealContext())
            {
                ctx.CustomerDetails.Add(customer);
                ctx.SaveChanges();
            }
        }
        public ObservableCollection<Country> GetCountries()
        {
            using (var ctx = new MyCerealContext())
            {
                return new ObservableCollection<Country>(ctx.Countries);
            }
        }
        public ObservableCollection<Cereal> GetMyCereals(CustomerDetails customer)
```

```
{
    using (var ctx = new MyCerealContext())
    {
        IQueryable<Cereal> cereals;
        try
        {
            cereals = from cereal in ctx.Cereals
            join cust in ctx.CustomerDetails on
           cereal.Customer.CustomerDetailsId
                    equals cust
                    .CustomerDetailsId
                select cereal;
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
            throw;
        }
        return new ObservableCollection<Cereal>(cereals.Where(c => c.Customer
            .Equals(customer)));
    }
}
public ObservableCollection<Ingredient> GetIngredients()
{
    using (var ctx = new MyCerealContext())
    {
        IQueryable<Ingredient> ingredients = null;
        try
        {
            ingredients = from ing in ctx.Ingredients
                join category in ctx.Categories on ing.Category.CategoryId
                    equals category.CategoryId
                select ing;
        }
        catch (Exception e)
        {
            Log.Error("GetIngredients failed: " + e.InnerException);
        }
        return new ObservableCollection<Ingredient>(ingredients);
    }
}

public ObservableCollection<Category> GetCategories()
{
    using (var ctx = new MyCerealContext())
    {
        return new ObservableCollection<Category>(ctx.Categories);
    }
}

public void AddCereal(Cereal cereal)
{
    using (var ctx = new MyCerealContext())
    {
        try
        {
```

```
                ctx.Cereals.Add(cereal);
                ctx.SaveChanges();
            }
            catch (Exception e)
            {
                Log.Error("AddCereal failed: " + e.InnerException);
            }
        }
    }

    public void DeleteMuesli(Cereal selectedCereal)
    {
        using (var ctx = new MyCerealContext())
        {
            try
            {
                ctx.Cereals.Remove(selectedCereal);
                ctx.SaveChanges();
            }
            catch (Exception e)
            {
                Log.Error("DeleteMuesli failed: " + e.InnerException);
            }
        }
    }

    public ObservableCollection<Ingredient> GetIngredientList(Cereal cereal)
    {
        using (var ctx = new MyCerealContext())
        {
            IQueryable<Ingredient> ingredients = null;
            try
            {
                ingredients = from ing in ctx.Ingredients
                    join category in ctx.Categories on ing.Category.CategoryId
                        equals category.CategoryId
                    select ing;
            }
            catch (Exception e)
            {
                Log.Error("GetIngredientList failed: " + e.InnerException);
            }
            return new ObservableCollection<Ingredient>(ingredients.Where(i => i
                .Cereals.Contains(cereal)));
        }
    }

    public void UpdateCereal(Cereal cereal)
    {
        using (var ctx = new MyCerealContext())
        {
            try
            {
                var old = ctx.Cereals.First(c => c.CerealId == cereal.CerealId);
                old.Name = cereal.Name;
                old.CreatedOn = cereal.CreatedOn;
```

```
                    old.Ingredients = cereal.Ingredients;
                    old.Price = cereal.Price;
                    ctx.SaveChanges();
                }
                catch (Exception e)
                {
                    Log.Error("UpdateCereal failed: " + e.InnerException);
                }
            }
        }
    }
```

MyCerealContext.cs

```
public class MyCerealContext : DbContext
{
    public MyCerealContext() : base("CerealDatabase")
    {
    }

    public DbSet<Country> Countries { get; set; }
    public DbSet<CustomerDetails> CustomerDetails { get; set; }
    public DbSet<Ingredient> Ingredients { get; set; }
    public DbSet<Cereal> Cereals { get; set; }
    public DbSet<Category> Categories { get; set; }
}
```

CerealContentCalculator.cs

```
public static class CerealContentCalculator{
    private const double CalsToJouleExp = 4.184;

    public static string CalculateCalsMultiple(
        ObservableCollection<IngredientViewModel> selectedIngredientList)
    {
        var total = selectedIngredientList.Select(CalculateCalories).Sum();
        return total + " Kcal / 100g";
    }
    public static double CalculatePrice(
        ObservableCollection<IngredientViewModel> selectedIngredientList)
    {
        return selectedIngredientList.Select(t => t.Ingredient.Price).Sum();
    }
    public static string CalculateCalsAndJouleForMultiple(
        ObservableCollection<IngredientViewModel> selectedIngredientList)
    {
        var total = selectedIngredientList.Select(CalculateCalories).Sum();
        var joule = total * CalsToJouleExp;
        return total + " Kcal / " + joule + " KJ";
    }
    private static double CalculateCalories(IngredientViewModel ingredient)
    {
            return (ingredient.Ingredient.Carbohydrates * 4.1 +
      ingredient.Ingredient.Fat * 9.3 +
                ingredient.Ingredient.Protein * 4.1) / 100 *
ingredient.Ingredient.Portion;
        }
        public static string CalculateCalsAndJoule(IngredientViewModel ingredient)
        {
```

```
            var cals = CalculateCalories(ingredient);

            var joule = cals * CalsToJouleExp;

            return cals + " Kcal / " + joule + " KJ";
        }
    }
```

CustomerValidation.cs

```
public class CustomerValidation
{
    public static string WrongField;

    public static bool ValidateCustomer(CustomerDetails customer)
    {
        return ValidateName(customer.Name) &&
               ValidateAddress(customer.Address) &&
               ValidateZip(customer.Zip) &&
               ValidateCity(customer.City) &&
               ValidatePhone(customer.Phone) &&
               ValidateEmail(customer.Email);
    }
    private static bool ValidateEmail(string email)
    {
        if (ValidateRegex(
            @"^([a-zA-Z0-9_\-\.]+)@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.)
            |(([a-zA-Z0-9\-]+\.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(\]?)$",
            email))
            return true;

        WrongField = "The Email must fulfill a valid Pattern" +
                     " (x@y.z - x: min 1 Char, y: min 3 Char," +
                     " z: 2-3 Char only A-Z.)";
        return false;
    }

    private static bool ValidatePhone(string phone)
    {
        if (ValidateLength(10, phone) && ValidateRegex("", phone)) return true;
        WrongField = "The Phone must be 10 character long and only numeric," +
                     " blanks or '+'";
        return false;
    }

    private static bool ValidateCity(string city)
    {
        if (ValidateLength(2, city)) return true;
        WrongField = "The City must be 2 character long";
        return false;
    }

    private static bool ValidateZip(string zip)
    {
        if (ValidateLength(4, zip) && ValidateRegex("^(0|[1-9][0-9]*)$", zip))
            return true;
        WrongField = "The Zip must be 4 characters long and contain only" +
```

```
                    " numeric values";
        return false;
    }

    private static bool ValidateRegex(string regex, string word)
    {
        return word != null && Regex.IsMatch(word, regex);
    }

    private static bool ValidateAddress(string address)
    {
        if (ValidateLength(5, address)) return true;
        WrongField = "The Address must be 5 character long";
        return false;
    }

    private static bool ValidateName(string name)
    {
        if (ValidateLength(5, name)) return true;
        WrongField = "The Name must be 5 character long";
        return false;
    }

    private static bool ValidateLength(int length, string word)
    {
        return length <= word?.Length;
    }
}
```