TEKO Schweizerische Fachschule

Semesterarbeit
Objektorientiertes Programmieren 2

DatationBase

Severin Gafner — David Hänni

begleitet durch
Herr Christian Herren

23 August, 2021

1 Einleitung

Severin Gafner und David Hänni führen im Rahmen des Studiums zum Techniker HF Applikationsentwicklung für das Modul Objektorientiertes Programmieren 2eine Semesterarbeit durch.

2 Aufgabenstellung

Nachfolgend die Umschreibung der Aufgabenstellen 1:1 aus dem Auftrag kopiert.

2.1 Umschreibung der Aufgabenstellung

Wir sind die IT-Firma NewIdeas und wir haben die nachfolgende Geschäftsidee: Im Zusammenhang mit Automationsvorhaben werden von unseren Kunden immer mehr kleine aber individuelle Lösungen gesucht. Wir haben nun festgestellt, dass sich die bekannten (umfangreichen) Datenbanksysteme für derartige Vorhaben nicht oder nur bedingt eignen, vor allem die teilweise abstrus teuren Lizenzmodelle lassen diese als nur bedingt tauglich erscheinen. Die verfügbaren OpenSource/Freewarelösungen können wir nicht ernst nehmen resp. wollen wir aus Gründen der Nachhaltigkeit nicht verwenden.

3 Projektumfeldanalyse

Mit der Projekumfeldanalyse werden möglichst viele Information über vorhandene Interessen, Bedürfnisse, Einflussmöglichkeiten und Beziehungen im Projektumfeld ermittelt.

3.1 Stakeholder

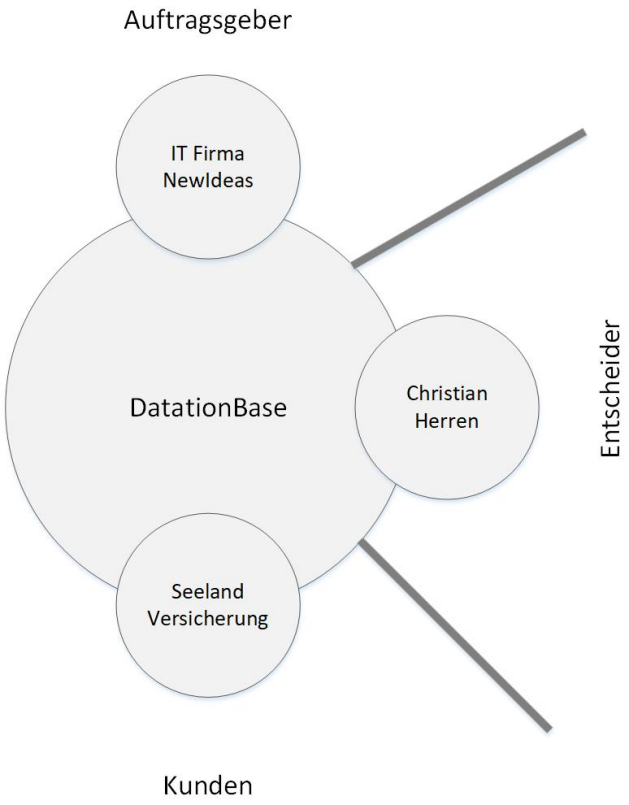

Abbildung 1: Identifizierte Stakeholder

Die Stakeholder wurden in drei Kategorien eingeteilt:

Auftraggeber Der Auftraggeber ist die IT Firma NewIdeas. Ihr Ziel ist es, ihre Kunden mit einer eigenen Datenbanklösung an sich zu binden und so zukünftige Wartungen und Modifikationen an der Lösung durchführen zu können.

Kunden Als erstes Kunde wird die Seeland-Versicherung die Datenbank einsetzen. Später werden weitere Kunden dazustossen. Der Kunde möchte eine performante Datenbank ohne Datenverlust.

Entscheider Christoph Herren ist ebenfalls ein Stakeholder dieses Projektes. Er überwacht die Durchführung der Arbeit und bewertet das Endresultat.

3.2 Situationsanalyse Seeland-Versicherung

Die Seeland-Versicherung spielt als erster Kunde eine sehr wichtige Rolle. Mit den 1'500 Kunden und 450 Schadensfälle pro Jahr beinhaltet die Datenbank nur wenig Datensätze. Angenommen es werden die Schadensfälle der letzten 20 Jahre migriert, so umfasst die Datenbank immer noch nur knapp 10'000 Datensätze.

4 Ziele

Die Projektziele zeigen auf, was erreicht werden muss.

4.1 MUSS-Ziele

4.1.1 Anforderungen geklärt

Die funktionalen und nicht-funktionalen Anforderungen an das Produkt sind klar definiert.

4.1.2 Use-Cases definiert

Use Cases wurden nach UML Richtlinien definiert und beschrieben.

4.1.3 Testkonzept erstellt

Ein Testkonzept wurde erstellt, welches die Testmethoden, die hilfmittel und die Testanlagen beschreiben. Zu jedem Use Case wurde mindestens ein positiv und ein negativ Test erfasst.

4.1.4 Systemarchitektur beschrieben

Die Systemarchitektur und die einzelnen Komponenten sowie deren Schnittstellen wurden beschrieben.

4.1.5 Fertigstellung erreicht

Das Produkt ist einsatzfähig und hat alle Testszenarien vom Testkonzept bestanden. Alle Use Cases wurden implementiert.

4.2 KANN-Ziele

4.2.1 Datenimport

Über eine Schnittstelle können aus einer Datei direkt in die Datenbank importiert werden.

5 Anforderungen

Die Anforderungen an das Produkt werden von den Zielen Abgeleitet. Sie werden in funktionale und nicht-funktionale Anforderungen aufgeteilt.

5.1 Funktionale Anforderungen

5.1.1 Tabellen definierten

Es können Tabellen mit unterschiedlichen Namen definiert werden.

5.1.2 Felder definieren

Auf einer Tabelle können Felder erfasst werden. Diese Felder besitzen einen Namen und einen der folgenden Datentypen: string, int, boolean

5.1.3 Datensätze hinzufügen

Auf einer bestehenden Tabelle können über eine C# Methode mehrere neue Datensätze hinzugefügt werden.

5.1.4 Datensätze löschen

Auf einer bestehenden Tabelle kann über eine C# Methode mehrere bestehender Datensätze gelöscht werden.

5.1.5 Datensätze verändert

Auf einer bestehenden Tabelle kann über eine C# Methode mehrere bestehender Datensätze verändert werden.

5.1.6 Datensätze anzeigen

Auf einer bestehenden Tabelle kann über eine C# Methode mehrere bestehender Datensätze ausgelesen werden.

5.1.7 Schnittstelle XML

Über eine Schnittstelle können Daten im XML Format in die Datenbank importiert werden.

5.1.8 Schnittstelle JSON

Über eine Schnittstelle können Daten im JSON Format in die Datenbank importiert werden.

5.1.9 Schnittstelle CSV

Über eine Schnittstelle können Daten im CSV Format in die Datenbank importiert werden.

5.1.10

5.2 Nicht funktionale Anforderungen

5.2.1 Speichervolumen

Das Produkt muss mindestens eine 2GB grosse Datenbank mit 1'000'000 Datensätzen bewältigen können.

5.2.2 Zugriffszeiten

Bei 100'000 Datensätzen darf die Zugriffszeit auf einen Datensatz nicht länger als 100 ms dauern.

5.2.3 Programmiersprache

Die Datenbank wird mittels C# implementiert.

5.2.4 Projektabgabe

Das Produkt sowie die Projektdokumentation müssen bis am 28.09.2021 um 18:30 Uhr abgegeben sein.

6 Konzept

6.1 Zugriffssprache IAccess

Wir haben uns dazu entschieden, dass der Zugriff auf die Datenbank direkt über eine DLL geschieht. Zum Nutzen der Datenbank muss der Entwickler unsere DLL einbinden und kann anschliessend direkt in C# sein Datenmodell erstellen und seine Abfragen tätigen. Zum definieren einer neuen Tabelle wird also Befehl übermittelt, sondern es wird eine neue Klasse erstellt, welche die Struktur der Tabelle angibt.

6.2 Feld

Ein Feld beinhaltet einen Namen und einen Wert in einem bestimmten Typ.

Beispiel Vorname = Hans Die Datenbank kann nur mit folgenden C# Datentypen umgehen:

- string
- int

- boolean

6.3 Record

Mehrere Felder werden zu einem Datensatz (Record) gegliedert. Ein Record definiert seine Felder, deren Namen und welchen Typ ein Feld hat.

6.4 Tabelle

Ein Tabelle besteht aus mehreren *gleichen* Records. Auf der Tabelle wird festgelegt welchen Type von Datensatz sie einhaltet.

6.5 Persistenz der Daten

6.5.1 Tabellen speichern

Für jede Tabelle wird eine Datei erstellt. Der Tabellenname wird im Filenamen hinterlegt. Die Datensätze dieser Tabelle werden im CSV-Format in der jeweiligen Tabellendatei abgelegt.

6.6 Synchronisieren der Daten mit dem persistenten Speicher

Die Daten müssen aus dem temporären Speicher in den persistenten Speicher geschrieben und auch wieder ausgelesen werden. Dieser Prozess ist sehr aufwändig und kann die Datenbank schnell verlangsamen. Wird jedoch nicht Regelmässig mit dem persistenten Speicher synchronisiert, können die Daten veraltet sein. Um festzulegen wann und wie die Synchronisation stattfindet wurde ein Variantenentscheid durchgeführt.

6.6.1 V1: Laden und Speichern bei jeder Aktion

Bei jedem hinzufügen eines Records wird dieser im Filesystem abgelegt.

6.6.2 V2: Laden und speicher über eigene Funktion