Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

# Bachelorarbeit Informatik

## Machbarkeitsanalyse für eine ressourcenorientierte Schnittstelle zur Verarbeitung grundlegender Probleme der Informatik

| | |
|---|---|
| **Autor** | Raffael Santschi<br>santsraf@students.zhaw.ch<br>Student im 8. Semester |
| **Hauptbetreuung** | Alain M. Lafon<br>lafo@zhaw.ch |
| **Experte** | Silvan Spross<br>silvan.spross@gmail.com |

| | |
|---|---|
| **Abgabedatum** | 23.07.2015 |

# Abstract

Ziel dieser Arbeit war es zu analysieren, ob eine generische Schnittstelle für grundlegende Probleme der Informatik möglich ist. Die Schnittstelle sollte keinerlei Kenntnisse der theoretischen Informatik oder der jeweiligen Probleme voraussetzen. Dazu wurden fünf Probleme mit hoher Laufzeitkomplexität aus 'Computers and Intractability: A Guide to the Theory of NP-Completeness' von Micheal Garey und David S. Johnson ausgewählt. Die Problemfelder wurden auf ihre Ein- und Ausgabeparameter analysiert und die dazugehörigen Algorithmen wurden betrachtet. Im späteren Verlauf der Arbeit wurde noch ein sechstes Problem dazugenommen, welches sehr ähnlich zu einem bereits ausgewählten Problem war. Dies um zu schauen, wie sich die beiden Probleme bei der Implementierung im Gegensatz zu den anderen Problemen verhalten.

Beim Erstellen des Konzept wurden die Gemeinsamkeiten des Berechnungsablaufs betrachtet. Um mehr Freiheiten zu haben und Benutzerfreundlichkeit zu garantieren, wurde die Nutzer- und Algorithmus-Welt voneinander entkoppelt. Dies bot die Möglichkeit jeweils eine andere Domänensprache zu verwenden. Zwischen den beiden Welten kamen pre- und post-Aktionen zum Einsatz, welche die Datenaufbereitung für den Algorithmus bzw. den Nutzer durchführten.

Vor der Umsetzung wurde eine Nutzwertanalyse zur Auswahl eines geeigneten Datenbanksystems durchgeführt. Nach der Vorselektierung standen die relationalen und dokumentorientierten Datenbanken zur Auswahl. Beim Vergleich wies das dokumentorientierte Datenbanksystem einige Vorzüge auf, wie seine Flexibilität, welche eine schnelle und unkomplizierte Erweiterung bzw. Anpassung ermöglicht.

Als Prototyp wurde ein REST API implementiert, welches die Funktionalität bereitstellt, Berechnung von den sechs verschiedenen Problemen zu erstellen. Hinter einem dieser Probleme wurde ein Algorithmus angebunden, womit der ganze Prozess durchgespielt werden konnte. Bei den anderen Problemen wurde anhand der Recherche Ein- und Ausgabeschemata der Algorithmen definiert. Zusätzlich wurde zu jedem Problem ein Validator geschrieben, welcher überprüft, ob ein Resultat gültig ist oder nicht.

Das Konzept für die Schnittstelle konnte für alle sechs Probleme angewandt und der Ablauf generisch gehalten werden. Die Machbarkeitsstudie ist als erfolgreich zu betrachten. Das Auswahlverfahren der Probleme hat sich bewährt, die Probleme zeigten unterschiedliche Ausprägungen. Einige Probleme könnten mit dem gleichen generische Algorithmus gelöst werden. Zwei andere Probleme, welche beide aus dem gleichen Bereich stammen, benötigen sehr unterschiedliche Herangehensweisen. Die Schnittstelle bietet genug Flexibilität um ganz unterschiedliche Probleme zu behandeln und verschiedene Algorithmen anzusteuern.

# Erklärung betreffend das selbständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:

……

Unterschriften:

……...

……

……

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Bachelorarbeiten zu Beginn der Dokumentation nach dem Abstract bzw. dem Management Summary mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

# Inhaltsverzeichnis


**1 Problemabgrenzung** **6**
- 1.1 Ausgangslage . . . 6
- 1.2 Ziele der Arbeit . . . 6
- 1.3 Aufgabenstellung . . . 6
- 1.4 Erwartete Resultate . . . 7
- 1.5 Nicht-Ziele . . . 7
- 1.6 Dokumentstruktur . . . 7

**2 Projektplanung** **8**
- 2.1 Meilensteine . . . 8
- 2.2 Arbeitspakete . . . 8
  - 2.2.1 Planung . . . 8
  - 2.2.2 Analyse und Auswahl der Probleme . . . 8
  - 2.2.3 Requirement Engineering . . . 9
  - 2.2.4 Konzept . . . 9
  - 2.2.5 Umsetzung Prototyp . . . 9
  - 2.2.6 Testing . . . 9
  - 2.2.7 Dokumentation . . . 9
- 2.3 Zeitplan . . . 9
  - 2.3.1 Geplante Abwesenheiten . . . 9
  - 2.3.2 Projektplan . . . 10
  - 2.3.3 Zeitschätzung auf Arbeitspaketebene . . . 10

**3 Theoretische Grundlagen** **12**
- 3.1 Komplexitätsklassen der theoretischen Informatik . . . 12
  - 3.1.1 NP . . . 12
  - 3.1.2 P . . . 12
  - 3.1.3 NP-schwer . . . 12
  - 3.1.4 NP-vollständig . . . 12
  - 3.1.5 $P = NP$? . . . 12
- 3.2 Algorithmentypen . . . 13
  - 3.2.1 Backtracking Algorithmen . . . 13
  - 3.2.2 Greedy Algorithmen . . . 13
  - 3.2.3 Evolutionäre Algorithmen . . . 14

**4 Analyse und Auswahl der Probleme [R1]** **15**
- 4.1 Auswahlverfahren . . . 15
- 4.2 Problemauswahl . . . 16
  - 4.2.1 Hierarchie der Reduktion . . . 16
  - 4.2.2 Graphentheorie . . . 16
  - 4.2.3 Netzwerk Design . . . 18
  - 4.2.4 Sequenzierung und Planung . . . 20
  - 4.2.5 Mathematisches Programmieren . . . 22
  - 4.2.6 Logik . . . 24
- 4.3 Übersicht Eingabe- und Ausgabedaten [R2] . . . 26
- 4.4 Beispiel für den Einfluss der Parameter auf die Komplexität [R1a] . . . 27
  - 4.4.1 Ausgangslage . . . 27
  - 4.4.2 Resultat . . . 27

**5 Anforderungsdokument [R3] 29**
5.1 Übersicht . . . 29
5.1.1 System- und Kontextabgrenzung . . . 29
5.1.2 Systemumgebung . . . 30
5.1.3 Annahmen . . . 30
5.1.4 Stakeholder . . . 31
5.2 Anforderungen . . . 32
5.2.1 Use Cases . . . 32
5.2.2 Anforderungen . . . 36
5.2.3 Zusammenfassung der Anforderungen . . . 43

**6 Konzept der Schnittstelle [R4] 44**
6.1 Übersicht . . . 44
6.2 Konzept . . . 44
6.2.1 REST API . . . 45
6.2.2 Business Logik . . . 45
6.2.3 Persistance API . . . 45
6.2.4 Datenbank . . . 46
6.2.5 Ablauf . . . 47
6.3 Datenbank Varianten . . . 49
6.3.1 CAP-Theorem . . . 49
6.3.2 Relationales Datenbanksystem . . . 50
6.3.3 Objektrelationales Datenbanksystem . . . 51
6.3.4 Objekorientiertes Datenbanksystem . . . 51
6.3.5 NoSQL Datenbanksystem . . . 52
6.3.6 Vorselektierung . . . 54
6.4 Nutzwertanalyse . . . 55
6.4.1 Bewertungskriterien . . . 55
6.4.2 Bewertung . . . 56
6.4.3 Fazit . . . 56
6.5 Datendiagramm des Datenspeichers . . . 56

**7 Umsetzung des Prototyps [R5] 59**
7.1 Erster Durchstich . . . 59
7.1.1 Erkenntnisse nach dem ersten Durchstich . . . 59
7.1.2 Anpassung des Konzepts nach dem ersten Durchstich . . . 60
7.2 Implementierung der Schnittstelle . . . 60
7.2.1 Statusabfrage . . . 60
7.2.2 WebHook Möglichkeit . . . 60
7.2.3 Technische Umsetzung und Probleme . . . 61
7.3 Implementierung der Probleme . . . 61
7.3.1 Rucksack . . . 62
7.3.2 Knotenfärbung . . . 63
7.3.3 Problem des Handlungsreisenden . . . 63
7.3.4 Briefträgerproblem . . . 64
7.3.5 Stundenplan Erstellung . . . 64
7.3.6 Spielplan Erstellung . . . 66
7.3.7 Übersicht der Schnittstellen . . . 67
7.3.8 Erstellung eines neuen Problems . . . 68
7.4 Entwicklungsumgebung . . . 69
7.4.1 IDE - Integrated Development Environment . . . 69
7.4.2 Versionierung . . . 70
7.4.3 Testen - Analysieren . . . 71

**8 Tests [R6] 72**
8.1 Einführung . . . 72

8.2 Testing . . . 72
8.3 Systemtest . . . 72
8.3.1 Testprotokoll . . . 72

**9 Schlussfolgerungen** **75**
9.1 Verwendung . . . 75
9.2 Fazit . . . 75
9.3 Ausblick . . . 75

**A Anhang** **I**
A.1 Risikoanalyse des Projekts . . . I
A.1.1 Risikoermittlung . . . I
A.1.2 Risikobewertung . . . I
A.1.3 Risikomatrix . . . II
A.1.4 Massnahmen . . . II
A.2 Schnittstellen Dokumentation . . . IV
A.2.1 Rucksack . . . IV
A.2.2 Knotenfärbung . . . V
A.2.3 Problem des Handlungsreisenden . . . VII
A.2.4 Briefträgerproblem . . . IX
A.2.5 Stundenplan Erstellung . . . XI
A.2.6 Spielplan Erstellung . . . XVI

**Akronyme** **XXI**

**Glossar** **XXIII**

**Literaturverzeichnis** **XXVI**

**Abbildungsverzeichnis** **XXVII**

**Listingsverzeichnis** **XXVIII**

**Tabellenverzeichnis** **XXX**

# 1 Problemabgrenzung

Die Problemabgrenzung dient dem Zweck, einen generellen Überblick über das Dokument zu verschaffen. Sie beinhaltet die Ausgangslage, das Ziel der Arbeit, die Aufgabenstellung, die erwarteten Resultate und die Nicht-Ziele. Zusätzlich wird der Aufbau dieses Dokumentes erklärt.

## 1.1 Ausgangslage

Bei einigen Problemen der Informatik kann deterministisch die exakte Lösung nicht in *Polynomialzeit* berechnet werden. Um in sinnvoller Zeit eine brauchbare Lösung zu erhalten, müssen diese Probleme im Allgemeinen mit Hilfe von Approximierungsalgorithmen angegangen werden. Zu dieser Kategorie gehören zum Beispiel das Problem des Handlungsreisenden oder das Rucksack Problem. Praktische Anwendung finden solche Probleme beispielsweise in der Logistik, bei der Routenplanung und beim Verladen von Fracht.

Die bekannten Approximierungsalgorithmen haben verschiedene Ausprägungen und auch unterschiedliche Eingabeparameter. Es gibt keine Schnittstelle für die Benutzung von diesen Algorithmen, welche keine detaillierte Kenntnis der darunterliegenden Probleme und Algorithmen erfordert. Eine Schnittstelle mit dieser Eigenschaft kann die Handhabung solcher Probleme enorm erleichtern.

## 1.2 Ziele der Arbeit

In der Arbeit soll ein Konzept für eine Schnittstelle zur Lösung verschiedener grundlegender Probleme der Informatik erarbeitet werden. Diese Schnittstelle soll basierend auf den Erkenntnissen einer Analyse über die Gemeinsamkeiten dieser Approximierungsalgorithmen aufgebaut werden. Dadurch soll es einem Benutzer ermöglicht werden seine jeweiligen Probleme, beispielsweise die effizienten Verpackung von Gegenständen, ohne ein Verständnis der darunterliegenden Probleme der Informatik anzugehen.

Bei der Erarbeitung der Schnittstelle stehen eine geeignete Persistenz-Lösung und sowie Datenstrukturen für Ein- und Ausgabe im Vordergrund.

## 1.3 Aufgabenstellung

Folgende Punkte werden in dieser Bachelorarbeit behandelt:

1. Recherche von real auftretenden Problemen, welche ausschliesslich durch den Einsatz von Algorithmen mit hoher Laufzeitkomplexität gelöst werden können. Einarbeiten und Analyse in die ausgewählten Algorithmen.
2. Ist-Analyse der verwendeten Datenstrukturen für die Algorithmen.
3. Anforderungsanalyse einer Schnittstelle für die ausgewählten Algorithmen.
4. Erarbeiten eines Konzeptes für die Implementierung der Schnittstelle sowie einer zugehörigen Persistenz-Schicht.
5. Implementierung eines Prototypen für die Schnittstelle und der Persistenz-Schicht.
6. Automatisiertes Testen der Schnittstelle.

## 1.4 Erwartete Resultate

Folgende Punkte werden als Resultate dieser Bachelorarbeit erwartet:

1. Übersicht der Probleme mit den dazugehörigen Algorithmen und Beschreibung der Algorithmen mit ihren Kerneigenschaften.
   a) Ausführungen zum Einfluss der Parameter der jeweiligen Probleme auf die Komplexität.
2. Übersicht über die verwendeten Datenstrukturen als Input / Output der Probleme.
3. Anforderungskatalog an die Schnittstelle.
4. Konzept einer generellen Schnittstelle zur Lösung der komplexen Probleme und Datendiagramm des Datenspeichers.
5. Prototypische Implementation der Schnittstelle und des Datenspeichers.
6. Automatische Tests mit dem dazugehörigen Testprotokoll.

## 1.5 Nicht-Ziele

Folgende Punkte wurden mit dem Auftraggeber als Nicht-Ziele definiert und sind somit nicht Teil dieses Projekts:

- Der Sicherheitsaspekt der Schnittstelle wird in diesem Projekt nicht behandelt.
- Es werden keine Algorithmen implementiert.
- Die Hochrechnung von Ausführungszeiten einzelner Probleme ist nicht Teil dieser Arbeit.

## 1.6 Dokumentstruktur

Dieses Dokument spiegelt die geleistete Arbeit wieder und ist in einzelne Kapitel unterteilt.

- Projektplanung: Schritte für die Erstellung des Projektplanes
- Theoretische Grundlagen: Beschreibung der wichtigsten verwendeten Begriffe und Theorien, welche für das Verstehen der Arbeit notwendig sind
- Analyse und Auswahl der Probleme: Erläuterung der Problemauswahl und Beschreibung der einzelnen Probleme
- Anforderungsdokument: System- und Kontextabgrenzung, *Stakeholder*, getroffene Annahmen und der Anforderungskatalog mit Use Cases und Anforderungen
- Konzept: Übersicht über das ganze System, Nutzwertanalyse der verschiedenen Datenbanktypen und die Konzeptbeschreibung der Schnittstelle
- Umsetzung: Erkenntnisse aus dem ersten Durchstich, Implementierung des Prototyps und Beschreibung der Entwicklungsumgebung
- Tests: Erläuterung der Test-Methoden und das Testprotokoll

Im Anhang sind die Risiko-Analyse, die Schnittstellen Dokumentation, das Glossar und alle Verzeichnisse zu finden. Falls es zu einem Begriff eine gängige Abkürzung gibt, wird diese beim ersten Auftauchen des Wortes in Klammern geschrieben und danach verwendet. Glossar Begriffe oder Akronyme sind beim ersten Erscheinen *kursiv* geschrieben, im weiteren Verlauf werden sie nicht weiter gekennzeichnet.

# 2 Projektplanung

Dieses Kapitel handelt von der Projektplanung und den verschiedenen Arbeitspaketen für dieses Projekt.

## 2.1 Meilensteine

Folgende Meilensteine wurden für dieses Projekt festgelegt:

| **Projektstart** | **23.01.2015** |
|---|---|
| Anforderungsdokument fertig | 28.02.2015 |
| Wissen über Probleme aufgebaut | 19.04.2015 |
| Erster *vertikaler Durchstich* | 26.04.2015 |
| Architektur festgelegt | 03.05.2015 |
| Prototyp fertig | 29.05.2015 |
| Dokumentation fertig | 12.06.2015 |
| Dokumentation korrigiert | 26.06.2015 |
| Präsentation | 08.07.2015 |

Tabelle 2.1: Meilensteine

## 2.2 Arbeitspakete

Das Projekt beinhaltet sieben Arbeitspakete:

- Planung
- Analyse und Auswahl der Probleme
- Requirement Engineering
- Konzept
- Umsetzung Prototyp
- Testing
- Dokumentation

### 2.2.1 Planung

In der Planungsphase wird geschaut, was in dem Projekt erreicht werden muss und wie diese Tätigkeiten auf die vorhandene Zeit aufgeteilt werden. Es wird auch das erste Mal mit dem Stakeholder geredet und erste Abmachungen getroffen.

### 2.2.2 Analyse und Auswahl der Probleme

Ein sehr wichtiges Paket ist die Analyse und die Auswahl der Probleme. Es ist wichtig, dass die Probleme möglichst vielfältig gewählt werden und sie gut analysiert werden, damit das Konzept mit den erhobenen Daten sauber geplant werden kann.

### 2.2.3 Requirement Engineering

Bei der Erstellung eines neuen Systems ist es immer wichtig, dass die Grundanforderungen bekannt sind. Um die Anforderungen zu erfassen, wird der Stakeholder befragt, was seine Wünsche sind. Oft werden bei der Anforderungsanalyse einige Anforderungen nicht aufgelistet, sondern einfach vorausgesetzt, sogenannte *Basisfaktoren*. Diese Anforderungen müssen dann vom Entwickler erfasst werden. Der Anforderungskatalog wird nach der Vollendung nochmals mit dem Stakeholder in einem Review angeschaut (siehe dazu auch [PR11]).

### 2.2.4 Konzept

Das Hauptziel dieser Arbeit ist ein Konzept, welches generisch ist und viele verschiedene Probleme mit wenig Aufwand verarbeiten kann. Die Architektur muss gut durchdacht sein und die Möglichkeiten müssen gegeneinander abgewägt werden. Schliesslich muss eine Konzept entstehen, welches in einem Prototypen umsetzbar ist.

### 2.2.5 Umsetzung Prototyp

In diesem Arbeitspaket werden die Anforderungen mit dem festgelegten Konzept umgesetzt. Die Schnittstelle wird entworfen, die ersten Tests werden durchgeführt und Unstimmigkeiten in den Anforderungen werden mit dem Stakeholder geklärt.

### 2.2.6 Testing

Das Projekt benötigt automatische Tests, welche erstellt und überprüft werden müssen. Die Tests sollten einen Grossteil des Projekts abdecken und bei einer Anpassung oder Erweiterung des Codes Sicherheit bieten.

### 2.2.7 Dokumentation

Die Dokumentation wird während des ganzen Projekts hindurch aktuell gehalten. Für das Erfassen dieses Dokuments wird LaTeX und das LaTeX-Template der ZHAW [zha15] mit ein paar kleinen Anpassungen verwendet.

## 2.3 Zeitplan

Der Zeitplan gibt eine grobe Übersicht, wann an dem Projekt gearbeitet werden kann und wann die verschiedenen Tätigkeiten fertig sein sollten. Die Angaben sind nur Richtwerte, da neben dem Projekt noch berufliche Verpflichtungen und andere Tätigkeiten Zeit benötigen.

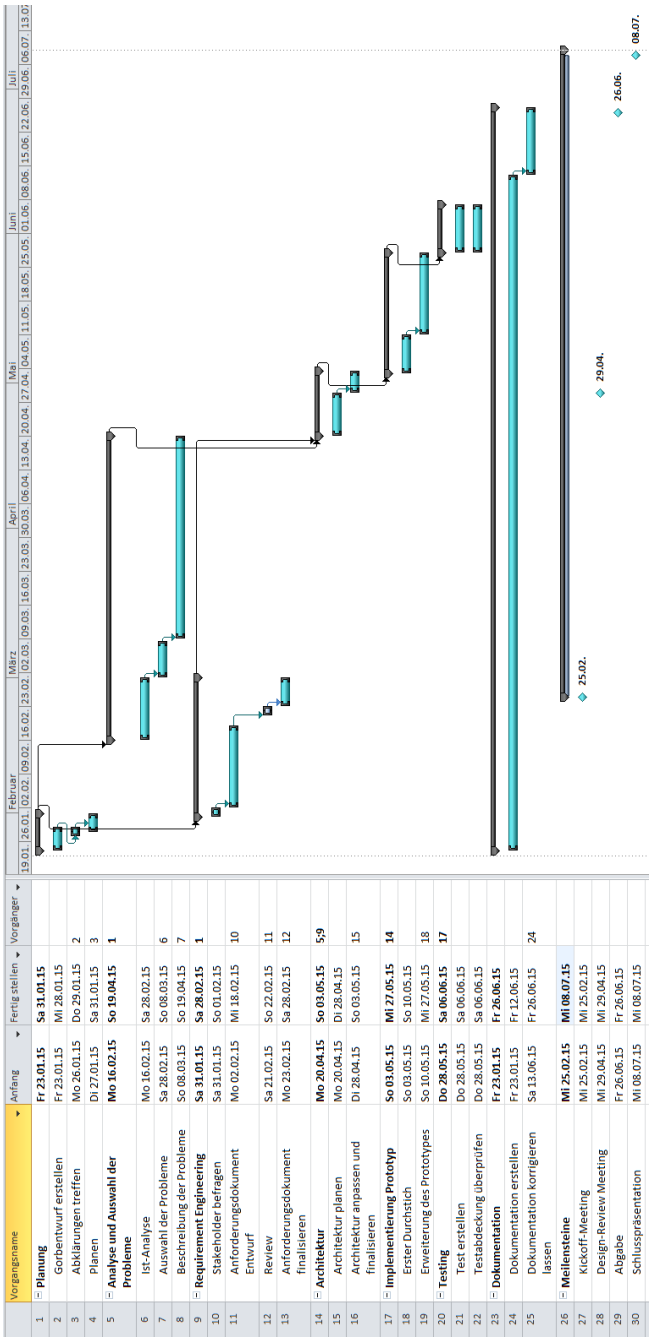
### 2.3.1 Geplante Abwesenheiten

| **Abwesenheit** | **Start - Ende** |
|---|---|
| Ferien | 07.02.2015 - 15.02.2015 |
| Seminararbeiten | 06.04.2015 - 19.04.2015 |
| Modulprüfungen und Vorträge Seminararbeit | 15.06.2015 - 28.06.2015 |

Tabelle 2.2: Geplante Abwesenheiten

### 2.3.2 Projektplan

Der Zeitplan basierend auf den Arbeitspaketen und den geplanten Abwesenheiten sieht wie folgt aus:



| | Vorgangsname | Anfang | Fertig stellen | Vorgänger |
|---|---|---|---|---|
| 1 | **Planung** | **Fr 23.01.15** | **Sa 31.01.15** | |
| 2 | Gorbentwurf erstellen | Fr 23.01.15 | Mi 28.01.15 | |
| 3 | Abklärungen treffen | Mo 26.01.15 | Do 29.01.15 | 2 |
| 4 | Planen | Di 27.01.15 | Sa 31.01.15 | 3 |
| 5 | **Analyse und Auswahl der Probleme** | **Mo 16.02.15** | **So 19.04.15** | **1** |
| 6 | Ist-Analyse | Mo 16.02.15 | Sa 28.02.15 | |
| 7 | Auswahl der Probleme | Sa 28.02.15 | So 08.03.15 | 6 |
| 8 | Beschreibung der Probleme | So 08.03.15 | So 19.04.15 | 7 |
| 9 | **Requirement Engineering** | **Sa 31.01.15** | **Sa 28.02.15** | **1** |
| 10 | Stakeholder befragen | Sa 31.01.15 | So 01.02.15 | |
| 11 | Anforderungsdokument Entwurf | Mo 02.02.15 | Mi 18.02.15 | 10 |
| 12 | Review | Sa 21.02.15 | So 22.02.15 | 11 |
| 13 | Anforderungsdokument finalisieren | Mo 23.02.15 | Sa 28.02.15 | 12 |
| 14 | **Architektur** | **Mo 20.04.15** | **So 03.05.15** | **5;9** |
| 15 | Architektur planen | Mo 20.04.15 | Di 28.04.15 | |
| 16 | Architektur anpassen und finalisieren | Di 28.04.15 | So 03.05.15 | 15 |
| 17 | **Implementierung Prototyp** | **So 03.05.15** | **Mi 27.05.15** | **14** |
| 18 | Erster Durchstich | So 03.05.15 | So 10.05.15 | |
| 19 | Erweiterung des Prototypes | So 10.05.15 | Mi 27.05.15 | 18 |
| 20 | **Testing** | **Do 28.05.15** | **Sa 06.06.15** | **17** |
| 21 | Test erstellen | Do 28.05.15 | Sa 06.06.15 | |
| 22 | Testabdeckung überprüfen | Do 28.05.15 | Sa 06.06.15 | |
| 23 | **Dokumentation** | **Fr 23.01.15** | **Fr 26.06.15** | |
| 24 | Dokumentation erstellen | Fr 23.01.15 | Fr 12.06.15 | |
| 25 | Dokumentation korrigieren lassen | Sa 13.06.15 | Fr 26.06.15 | 24 |
| 26 | **Mellensteine** | **Mi 25.02.15** | **Mi 08.07.15** | |
| 27 | Kickoff-Meeting | Mi 25.02.15 | Mi 25.02.15 | |
| 28 | Design-Review Meeting | Mi 29.04.15 | Mi 29.04.15 | |
| 29 | Abgabe | Fr 26.06.15 | Fr 26.06.15 | |
| 30 | Schlusspräsentation | Mi 08.07.15 | Mi 08.07.15 | |

Abbildung 2.1: Projektplan (eigene Darstellung)

### 2.3.3 Zeitschätzung auf Arbeitspaketebene

| Arbeitspaket | Schätzung (h) | Tatsächlich (h) |
|---|---|---|
| Requirement Engineering | 20 | 30 |
| Reale Optimierungsprobleme suchen | 50 | 70 |
| Wissensaufbau Algorithmen | 70 | 50 |
| Konzept Planung | 60 | 70 |
| Prototyp Entwicklung | 50 | 80 |
| Tests | 10 | 20 |
| Dokumentation | 120 | 100 |
| Total | 380 | 420 |

Tabelle 2.3: Zeitschätzung auf Arbeitspaketebene

### Erklärung der Abweichungen

Das Requirement Engineering und Testen wurde unterschätzt, die Zeit floss jedoch vor allem in Detailarbeiten. Bei der Suche von realen Optimierungsprobleme wurde mehr Zeit beansprucht, als ursprünglich geplant. Während der Suche konnte jedoch bereits Wissen über die Algorithmen aufgebaut werden, was dazu führte, dass in diesem Bereich weniger Zeit benötigt wurde. Als Evaluation des gewählten Konzept wurde ein vertikaler Durchstich durchgeführt, was sich in der benötigten Zeit widerspiegelt. Die Entwicklung des Prototyps war aufwändiger als gedacht, was daran lag, dass sechs statt fünf Probleme umgesetzt wurden. Die Dokumentation benötigte nicht so viel Zeit, jedoch ist dieser Punkt auch etwas ungenau zu messen, da in den anderen Paketen bereits Dokumentation entsteht.

# 3 Theoretische Grundlagen

Die theoretischen Grundlagen dient dazu, wichtige Informationen zum Verständnis der Arbeit zu erläutern. Es werden die Komplexitätsklassen der theoretischen Informatik und die verschiedenen Algorithmentypen erklärt.

## 3.1 Komplexitätsklassen der theoretischen Informatik

In der theoretischen Informatik wird zwischen verschiedenen Komplexitätsklassen unterschieden. In diesem Kapitel werden nur auf die Komplexitätsklassen NP, P, NP-schwer und NP-vollständig eingegangen, weitere Klassen wie zum Beispiel Random Polynomial (RP) oder Zero-Error, Probabilistic, Polynomial (ZPP) werden nicht erläutert, da sie nicht für das Verständnis der Arbeit notwendig sind. Die nachfolgenden Erklärungen sind aus [HMU11] und [Ric12] abgeleitet.

### 3.1.1 NP

Falls die Korrektheit einer Lösung zu einem Problem in Polynomialzeit überprüft werden kann, liegt das Problem in NP. Die Prozedur, welche für die Überprüfung benötigt wird, wird *Polynomialzeit-Verifizierer* genannt. In Polynomialzeit lösbar heisst, dass die Laufzeitkomplexität in einem Polynom mit der Form $n^k$ dargestellt werden kann, wobei n die Eingabelänge und k eine Konstante ist.

### 3.1.2 P

Probleme, welche in Polynomialzeit mit Hilfe einer deterministischen *Turingmaschine* lösbar sind, gehören zu der Klasse der P-vollständigen Probleme.

### 3.1.3 NP-schwer

Probleme, welche nicht in Polynomialzeit lösbar sind, gehören zu den NP-schweren Problemen. Sie besitzen eine Laufzeitkomplexität höher als polynomial, zum Beispiel $k^n$ (exponentiell) oder $n!$ (faktoriell). Um zu beweisen, dass das Problem NP-schwer ist, wird versucht, ein anderes bekanntes NP-schweres Problem auf dieses Problem zu reduzieren. Mit diesem Beweis wird gezeigt, dass das Problem mindestens so schwer wie das andere Problem ist.

### 3.1.4 NP-vollständig

Stephen Cook hat mit dem Beweis der NP-Vollständigkeit des SAT-Problems [Coo71] die Grundlage für den Beweis der NP-Vollständigkeit vieler weiterer Probleme gelegt. Damit ein Problem als NP-vollständig gilt, müssen folgende zwei Punkte erfüllt sein:

- Ein Polynomialzeit-Verifizierer für das Problem ist vorhanden.
- Ein anderes bekanntes NP-vollständiges Problem ist auf dieses Problem reduzierbar.

### 3.1.5 $P = NP$?

Die Abbildung 3.1 zeigt die Aufteilung der verschiedenen Komplexitätsklassen für die beiden Fälle $P \neq NP$ und $P = NP$. Momentan wird davon ausgegangen, dass $P \neq NP$ ist und somit die linke Aufteilung stimmt. Keine der beiden Behauptungen konnte bis jetzt bewiesen werden. Der letzte Versuch, $P \neq NP$ zu beweisen, wurde von Vinay Deolalikar im Jahr 2010 unternommen [Deo10]. In diesem Versuch wurden jedoch zwei Fehler entdeckt (siehe [Lip10]) und somit bleibt der Beweis weiter offen.

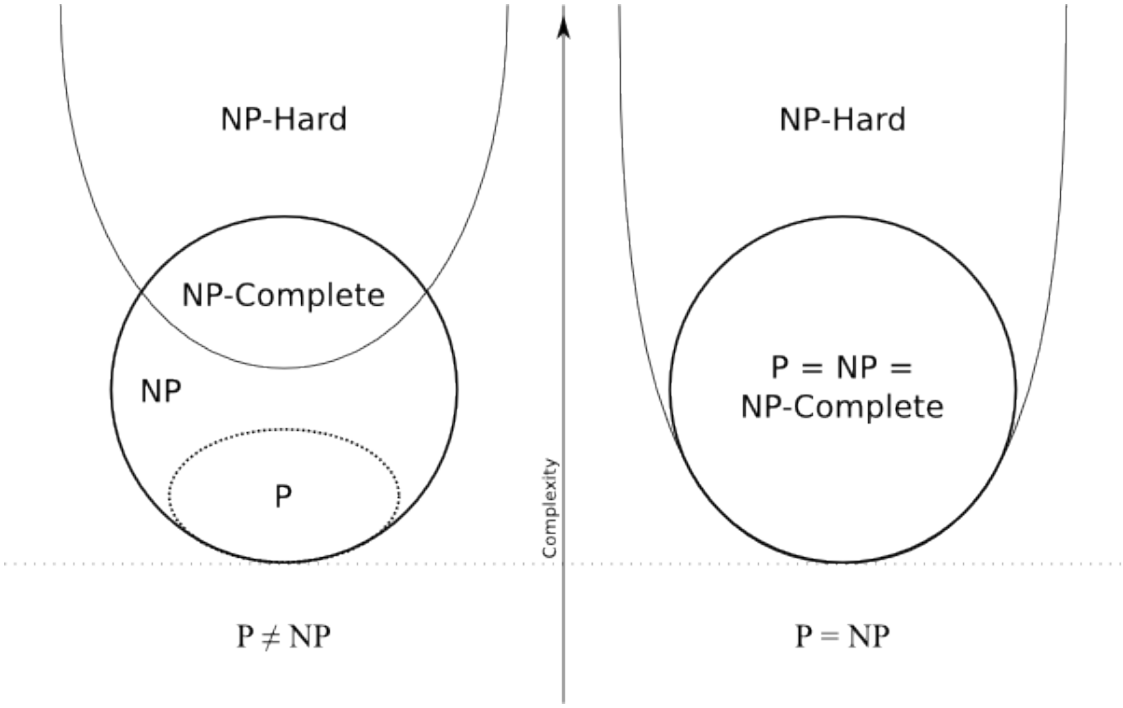



Abbildung 3.1: Übersicht der Komplexitätsklassen ($P \neq NP$ und $P = NP$) (Grafik entnommen aus [Esf15])

## 3.2 Algorithmentypen

### 3.2.1 Backtracking Algorithmen

Beim Backtracking geht es darum, sich einer Lösung eines Problems schrittweise zu nähern. Bei jedem neuen Schritt wird geprüft, ob das Resultat noch eine gültige Lösung darstellt. Falls dies nicht der Fall ist, wird der letzte Schritt rückgängig gemacht und es wird ein anderer Weg eingeschlagen. In [Pie11] wird dieses Verfahren anhand des Damenproblems aufgezeigt.

Die Laufzeit eines Backtracking Algorithmus ist Abhängig von den Anzahl an Möglichkeiten und der Verzweigungsrate. Im schlechtesten Fall beträg sie $O(z^n)$, wobei $n$ die Anzahl Möglichkeiten und $z$ die Verzweigungsrate ist. Die Suche dauert oft lange, deshalb empfiehlt sich ein Backtracking Algorithmus nur bei kleinen Lösungsbäumen [Ste13]. Jedoch wird mithilfe eines Backtracking Algorithmus immer eine Lösung gefunden, falls eine existiert.

Laufzeit, Exaktheit, Schwierigkeit

### 3.2.2 Greedy Algorithmen

Greedy Algorithmen liefern oft eine schnelle Lösung, welche aber meist nicht optimal ist. Die Algorithmen entscheiden bei jedem Schritt, was die aktuell beste Möglichkeit ist. Da sie nicht alle Möglichkeiten betrachten, finden sie oft nur ein lokales Minimum bzw. Maximum [McM14]. In Abbildung 3.2 ist in grün der Weg des Greedy Algorithmus zu sehen. Der Algorithmus entscheidet sich für die 12 auf der zweiten Ebene, da diese im Betrachtungsraum als beste Lösung gilt. Mit dem violetten Pfad könnte jedoch ein viel höherer Wert erzielt werden.

Laufzeit, Exaktheit, Schwierigkeit

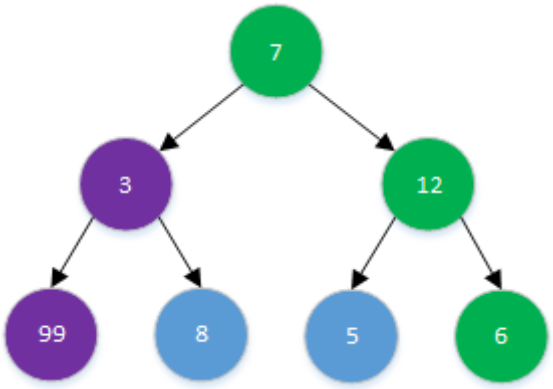



Abbildung 3.2: Suchablauf eines Greedy Algorithmus (eigene Darstellung, Daten entnommen aus [pic11])

### 3.2.3 Evolutionäre Algorithmen

Evolutionäre Algorithmen nähern sich einer optimalen Lösung an. Sie basieren auf Kombinationen von Objekten und einer Fitnessfunktion zur Bewertung der einzelnen Generationen. Ein Ablauf eines Evolutionären Algorithmus sieht meist wie folgt aus:

1. Initialisierung: Die erste Generation wird meist zufällig erzeugt
2. Iteration durch folgende Schritte, bis die Lösung den gewünschten Wert erreicht hat:
   a) Evaluation: Mit Hilfe der Fitnessfunktion wird die erstellte Generation bewertet
   b) Selektion: Auswahl von Objekten (Individuen) für die Rekombination
   c) Rekombination: Erstellen einer neuen Generation durch die Kombination der ausgewählten Individuen
   d) Mutation: Veränderung der Eigenschaften (Gene) der Nachfahren

Die Mutation und Rekombination können positive, negative oder neutrale Eigenschaften haben. Wie in der Natur überlebt der am besten Angepasste und somit wird die Lösung immer besser.

Laufzeit, Exaktheit, Schwierigkeit

# 4 Analyse und Auswahl der Probleme [R1]

In diesem Kapitel werden die verschiedene Probleme, welche für die Erstellung der Schnittstelle betrachtet wurden, und das dazugehörige Auswahlverfahren erläutert.

## 4.1 Auswahlverfahren

Da es in der Welt unzählige Probleme mit hoher Laufzeitkomplexität gibt, musste ein geeignetes Auswahlverfahren gefunden werden. Mit diesem Verfahren sollten möglichst unterschiedliche Typen dieser Probleme für die Schnittstelle evaluiert werden. Dafür wurde eine Kategorisierung der Probleme gesucht, auf welche sich gestützt werden konnte. Immer wieder wird sich in der Informatik auf das Buch 'Computers and Intractability: A Guide to the Theory of NP-Completeness' [GJ79] von Micheal Garey und David S. Johnson bezogen. Laut einer Studie von CiteSeer war es im Jahr 2006 das meist zitierte Buch der Informatik [cit15]. In diesem Buch werden verschiedene NP-vollständige und NP-schwere Probleme vorgestellt und in Kategorien unterteilt. Diese Kategorien (siehe Auflistung unten) werden auch benutzt, um die zu analysierenden Probleme auszuwählen.

- Graphentheorie
- Netzwerk Design
- Sets und Partitionen
- Speicherung und Wiederherstellung
- Sequenzierung und Planung
- Mathematisches Programmieren
- Algebra und Zahlentheorie
- Spiele und Puzzles
- Logik
- Automaten und Sprachtheorie
- Programm Optimierung
- Sonstiges
- Offene Probleme

Im Rahmen dieser Arbeit konnten nicht alle Probleme behandelt werden. Deshalb wurde eine Auswahl von fünf Problemen getroffen, welche in der Realität auftreten und nicht nur von rein wissenschaftlichem Interesse sind. Bei der Auswahl wurde darauf geachtet, dass die Probleme aus unterschiedlichen Kategorien kommen. Es wurden jedoch auch zwei aus der gleichen Kategorie ausgewählt, um zu analysieren wie sich diese im Gegensatz zu den anderen verhalten.

Die ausgewählten Probleme werden im folgenden Abschnitt genauer erläutert und für die weiteren Schritte der Erstellung der Schnittstelle berücksichtigt. Durch dieses Vorgehen konnte eine hohe Diversität von Problemen sichergestellt werden, was bei der Erstellung der Anforderungen und des Konzepts hilfreich war.

## 4.2 Problemauswahl

### 4.2.1 Hierarchie der Reduktion

Wie bereits in der Einleitung beschrieben, muss für den Beweis der NP-Vollständigkeit ein bekanntes NP-vollständiges Problem auf das Bestehende reduziert werden können. In Abbildung 4.1 ist die Hierarchie der Reduktion für die ausgewählten Probleme aufgezeigt.

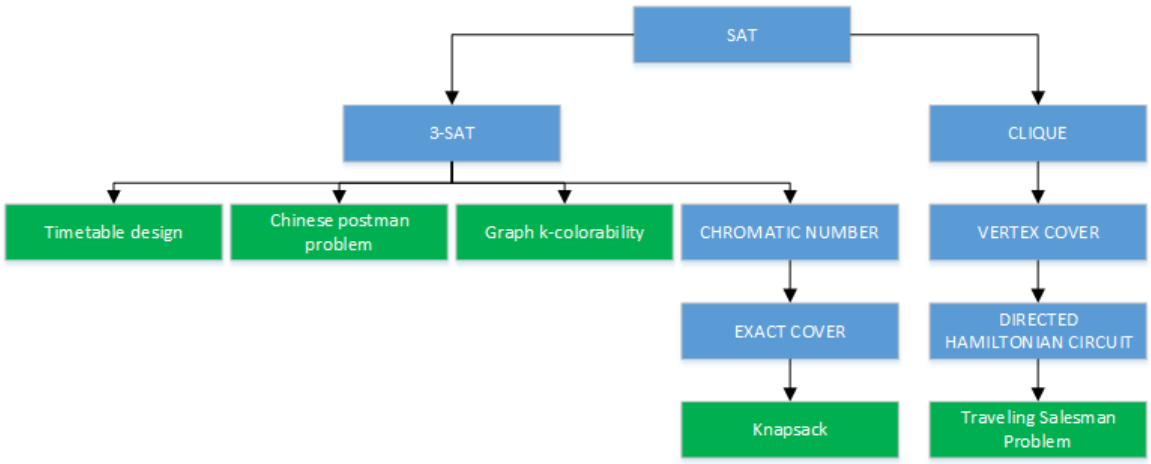



Abbildung 4.1: Hierarchie der Reduktion zum Beweis der NP-Vollständigkeit (eigene Darstellung, Daten aus [GJ79])

### 4.2.2 Graphentheorie

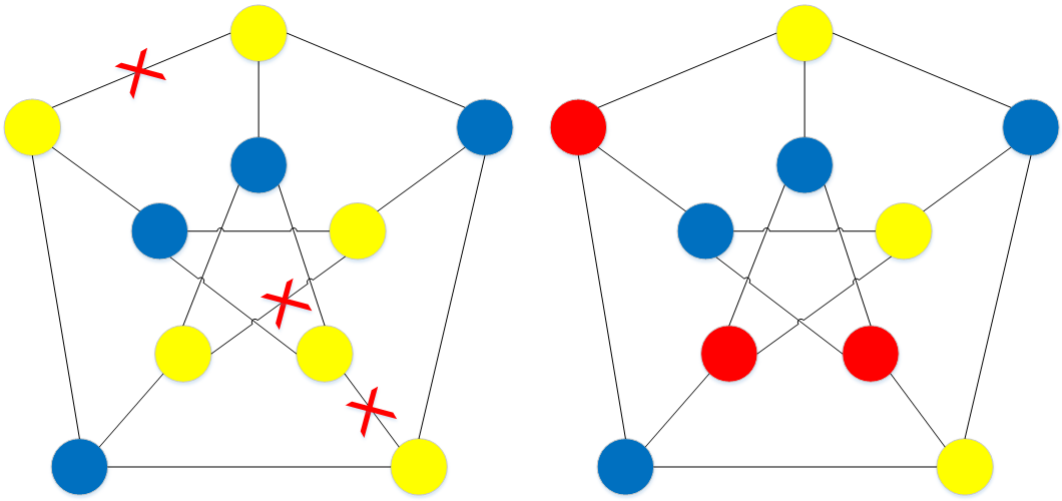
### Färbung (Graphtheorie)

Die Färbung in der Graphtheorie ist ein NP-vollständiges Problem. Dies wurde durch die Reduktion des 3-SAT Problemes bewiesen.

**Beschreibung** Englischer Name: Graph k-colorability

Bei diesem Problem geht es darum, die Knoten eines Graphens so zu färben, dass keine zwei benachbarten Knoten die gleiche Farbe tragen. Ein Graph heisst k-färbbar, wenn die Färbung mit k Farben korrekt durchgeführt werden kann. Dieses Problem ist für $k = 2$ in polynomieller Zeit lösbar, für $k \geq 2$ jedoch nicht mehr. Es gibt Spezialfälle, bei welchen auch ein Problem mit $k \geq 2$ in polynomieller Zeit lösbar sind (siehe [GJ79]).

**Beispiel** Gegeben sei ein Graph mit 10 Knoten mit einer vorgegebenen Konfiguration (siehe Abbildung 4.2).
Gesucht ist die Färbung der Knoten, damit keine zwei benachbarten Knoten die gleiche Farbe haben. Weiter die minimale Anzahl Farben ($k$), welche verwendet werden müssen, damit die Bedingung erfüllt ist. Der Graph 4.2(a) zeigt eine ungültige Lösung mit zwei Farben. Durch einen Algorithmus kann ermittelt werden, dass dieser Graph mindestens drei Fraben benötigt, um die Bedingung zu erfüllen (siehe Graph 4.2(b)). Somit ist $k = 3$.



(a) Inkorrekte Färbung (b) Korrekte Färbung

Abbildung 4.2: Kanten Färbung eines Graphen mit 10 Knoten (eigene Darstellung)

**Eingabe- und Ausgabedaten**
Eingabedaten: Knoten mit ihren Verbindungen zu anderen Knoten
Ausgabedaten: Knoten mit ihrer Färbung und $k$ (Anzahl benötigter Farben)

**Einfluss der Parameter auf die Komplexität**
Bei der Knotenfärbung steigert sich die Komplexität mit Anzahl Kanten. Die Knoten sind nur insofern relevant, dass sie neue Kanten aufspannen. Dass die Komplexität nur von der Anzahl Kanten abhängt, zeigt auch die Formel zur Berechnung des Maximums von $k$: $k \leq \frac{1}{2} + \sqrt{2m + \frac{1}{4}}$, wobei m für die Anzahl Kanten steht (siehe [Sch08]). Je grösser $k$, desto häufiger treten Kollisionen auf und umso mehr Varianten müssen ausprobiert werden.

**Bekannte Algorithmen** (siehe [Sie03a], [KN12] und [Sie03b])

- Spalten-Generierungs-Ansatz
- Sequentielles Färben (Sequential Coloring)
- Backtracking Algorithmus
- Greedy Algorithmus
- Johnson-Algorithmus

**Bekannte reale Probleme** Es gibt diverse reale Probleme, welche mit der Knotenfärbung gelöst werden können, hier sind nur einige davon aufgelistet:

- Stundenplan: Anhand der eingegebenen Daten wird ein Konfliktmatrix erstellt und diese dann in ein Färbungsproblem umgewandelt. Die Anzahl benötigten Farben sind dann die Anzahl der verschiedenen Perioden, welche es benötigt, um eine Stundenplan ohne Konflikte zu erstellen. [HBSA11] [KS] [Abd06]
- Frequenzverteilung (Mobilfunk): Im Mobilfunk hat jede Antenne einen Frequenzbereich. Im Graph werden die Antennen miteinander verbunden, bei denen sich die Reichweite überschneidet. Die Farben können durch Frequenzen ersetzt werden. Die Lösung stellt eine konfliktfreie Mobilnetzabdeckung dar. [Sie03b]
- Färben von Landkarten: Die Länder sind die Knoten, die Kanten die Verbindung zu den Nachbarländern und die errechnete Farbe entspricht der Einfärbung auf der Landkarte. [Sie03b]

### 4.2.3 Netzwerk Design

## Problem des Handlungsreisenden

Das Problem des Handlungsreisenden ist ein NP-vollständiges Problem. Dies wurde durch die Reduktion des Hamiltonkreisproblemes bewiesen.

**Beschreibung** Englischer Name: Traveling Salesman Problem
Beim Problem des Handlungsreisenden geht es darum, mit einer optimalen Route von einem Ausgangspunkt verschiedene Wegpunkte abzufahren und wieder zum Ausgangspunkt zurück zu kehren. Bei zehn Wegpunkten und ungleichen Hin- und Zurückwegen sind das über dreieinhalb Millionen Möglichkeiten. Bei der symmetrischen Variante sind die Verbindungen zwischen zwei Punkten gleich lang, was die Komplexität halbiert.

**Beispiel** Gegeben seien vier Wegpunkte (A, B, C, D), der Startpunkt sei A.
Gesucht ist die optimale Route, welche alle Wegpunkte beinhaltet und wieder bei A endet.
Die Abbildung 4.3 zeigt alle möglichen Lösungen mit ihren errechneten Werten. Die Route A-D-C-B-A ist mit einem Wert von 16 die beste Route. Weiter ist zu sehen, dass schon bei 4 Wegpunkten 6 Möglichkeiten vorhanden sind.

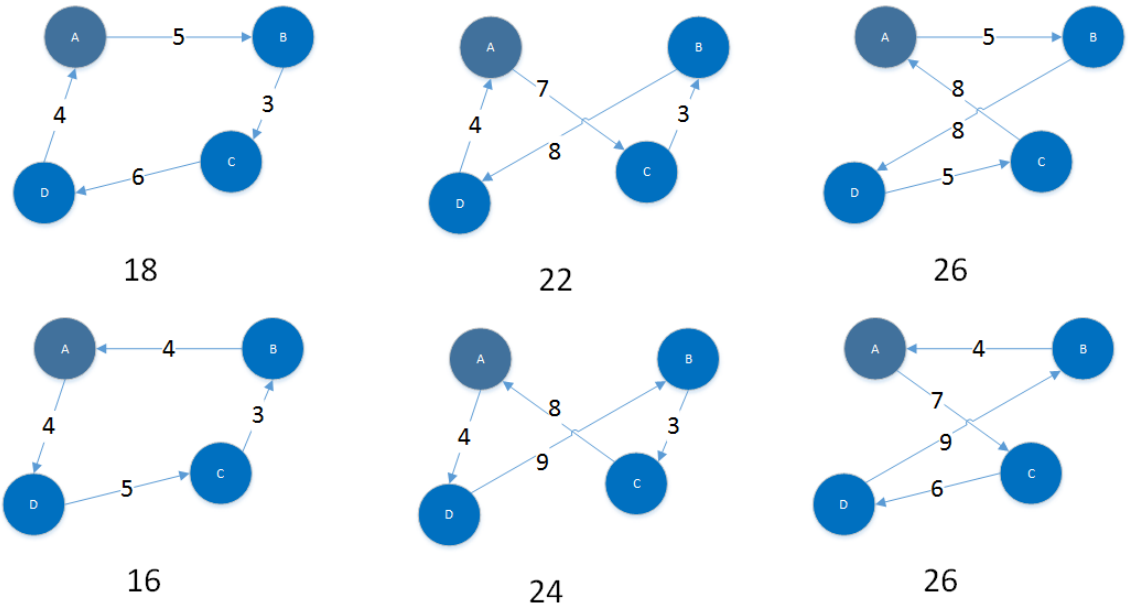



Abbildung 4.3: Problem des Handlungsreisenden mit 4 Wegpunkten (eigene Darstellung)

**Eingabe- und Ausgabedaten**
Eingabedaten: Startpunkt und Wegpunkte, welche passiert werden müssen
Ausgabedaten: Reihenfolge der Wegpunkte für eine optimale Route

**Einfluss der Parameter auf die Komplexität**
Die Komplexität beim Problem des Handlungsreisenden wird durch die Anzahl Wegpunkte bestimmt. Für ein asymmetrisches Problem ist die Formel zur Berechnung der Möglichkeit $(n-1)!$ für ein symmetrisches Problem hingegen $\frac{(n-1)!}{2}$.

**Bekannte Algorithmen** [Sau12] [LS]

- Branch and Bound
- Nearest-Neighbor
- 2Opt
- Christofides

## Briefträgerproblem

Das Briefträgerproblem ist ein NP-vollständiges Problem. Dies wurde durch die Reduktion des 3-SAT Problemes bewiesen.

**Beschreibung** Englischer Name: Chinese postman problem
Das Briefträgerproblem ist vergleichbar mit dem Problem des Handlungsreisenden, jedoch geht es darum jede Kante mindestens ein Mal abzufahren. Die Knoten stellen Kreuzungen dar, die Kanten entsprechen den Strassen. Die minimale Länge kann mit Hilfe des Eulerkreises relativ einfach berechnet werden. Falls der Graph die Kriterien des Eulerkreises nicht erfüllt, werden alle Knoten mit ungerader Anzahl Kanten mit einem anderen solchen Knoten über die kürzeste Route verbunden. Die minimal Läng ist die Summe aller Strecken und Hilfsstrecken. [PB04]

**Beispiel** Gegeben sei der Graph mit den Punkten A bis H und den Verbindungen in blau. Gesucht ist eine Route mit dem kürzesten Weg, welche jede Kante mindestens ein Mal abfährt. [PB04] Die grün eingezeichneten Verbindungen sind Hilfslinien, um den Graphen in einen *Eulerkreis* zu verwandeln. Eine mögliche Lösung mit der minimal Länge von 1000 ist die Route A-D-C-G-H-C-A-B-D-F-B-E-F-H-F-B-A.

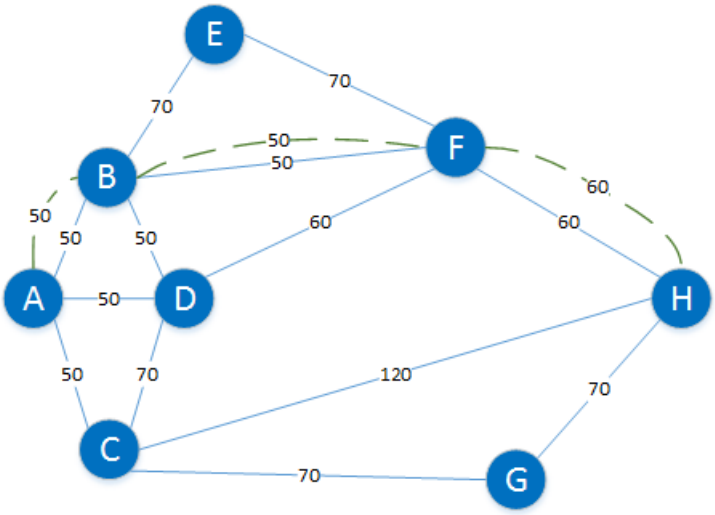



Abbildung 4.4: Beispiel für ein Briefträgerproblem (eigene Darstellung, Daten entnommen aus: [PB04])

**Eingabe- und Ausgabedaten**
Eingabedaten: Startpunkt und Knoten mit ihren Verbindungen mit einer Gewichtung
Ausgabedaten: Reihenfolge der Knoten für eine minimale Strecke

**Einfluss der Parameter auf die Komplexität**
Wie bei der Knotenfärbung spielt die Anzahl Kanten die Hauptrolle bei der Komplexität. Wie viele Knoten ein Graph hat und ob dieser bereits eulersch ist, hat einen geringeren Einfluss.

**Bekannte Algorithmen**
Briefträgeralgorithmus (Chinese postman algorithm)

### 4.2.4 Sequenzierung und Planung

## Stundenplan-Erstellung

Das Erstellen eines Stundenplans ist ein NP-vollständiges Problem. Dies wurde durch die Reduktion des 3-SAT-Problems bewiesen.

**Beschreibung** Englischer Name: Timetable design
Die Erstellung von Stundenplänen ist ein sehr komplexes Problem. Basierend auf Fächer, Lehrer, Klassen und Klassenzimmern wird versucht, eine optimale Verteilung der Stunden zu erreichen. Die wichtigste Bedingung ist, dass eine Ressource, sei das ein Lehrer, eine Klasse oder ein Schulzimmer, zu jedem Zeitpunkt höchstens ein Mal verplant ist. Zusätzlich kann es beliebige weitere Kriterien geben, beispielsweise dass eine Klasse nie mehr als neun Lektionen pro Tag haben soll oder ein Lehrer nie mehr als fünf Lektionen nacheinander unterrichten sollte. Bei der Erstellung von Stundenplänen wird oft von Hard Constraints und Soft Constraints gesprochen. Ein Hard Constraint ist im Gegensatz zu einem Soft Constraint unabdingbar. Es ist nicht möglich, dass ein Lehrer zwei verschiedene Fächer gleichzeitig unterrichtet, notfalls könnte er aber mehr als fünf Lektionen hintereinander unterrichten. [AA92] [Abr91] [GWB03] [Jaf14]

**Beispiel** Gegeben seien die Fächer, Lehrer, Klassen und Klassenzimmer aus den Tabellen 4.1, 4.2, 4.3 und 4.4.
Gesucht ist ein Stundenplan, bei welchem es keine Kollisionen für Lehrer, Klassen und Klassenzimmer gibt.

| Fachname | Kürzel |
|---|---|
| Mathematik | M |
| Deutsch | D |
| Englisch | E |
| Französisch | F |
| Sport | Sp |

Tabelle 4.1: Schulfächer

| Lehrername | Ausbildung |
|---|---|
| Angst | Deutsch, Mathematik, Sport |
| Arm | Sport |
| Bernasconi | Deutsch, Mathematik, Französisch |
| Müller | Deutsch, Mathematik, Englisch, Französisch |
| Pfister | Englisch, Französisch |

Tabelle 4.2: Lehrer

| Klassenname | Benötigte Fächer |
|---|---|
| Tja13 | Deutsch, Mathematik, Sport |
| Tja12 | Deutsch, Mathematik, Sport, Französisch |
| Tja11 | Deutsch, Mathematik, Sport, Französisch, Englisch |
| Tja10 | Deutsch, Mathematik, Sport, Französisch, Englisch |

Tabelle 4.3: Klassen

| Zimmername |
|---|
| Zimmer 101 |
| Zimmer 103 |
| Zimmer 201 |
| Turnhalle |

Tabelle 4.4: Klassenzimmer

Tabelle 4.5 zeigt eine mögliche Lösung. Es wurde beachtet, dass die Lehrer möglichst gleich viele Stunden unterrichten und die Schüler keine doppelten Freistunden haben. Es fällt auf, dass das Zimmer 201 nur selten besetzt ist. In der zweiten Variante (siehe Tabelle 4.6) wurden die Stunden der Klasse Tja11 so verschoben, dass das Zimmer 201 gar nicht mehr benötigt wird. Die Klasse Tja11 hat nun aber eine doppelte Freistunde. Die Eingabedaten sind im Beispiel noch sehr überschaubar, trotzdem gibt es bereits enorm viele verschiedenen Möglichkeiten und Ausprägungen.

| Uhrzeit | Turnhalle | Zimmer 101 | Zimmer 103 | Zimmer 201 |
|---|---|---|---|---|
| 0800-0900 | Sp / Tja13 / Arm | D / Tja11 / Müller | | |
| 0900-1000 | Sp / Tja13 / Arm | M / Tja11 / Müller | F / Tja12 / Pfister | |
| 1000-1100 | Sp / Tja12 / Arm | D / Tja10 / Bernasconi | M / Tja13 / Angst | F / Tja11 / Müller |
| 1100-1200 | Sp / Tja12 / Arm | M / Tja10 / Angst | D / Tja13 / Bernasconi | E / Tja11 / Pfister |
| 1300-1400 | Sp / Tja11 / Angst | F / Tja10 / Bernasconi | D / Tja12 / Müller | |
| 1400-1500 | Sp / Tja11 / Angst | E / Tja10 / Pfister | M / Tja12 / Bernasconi | |
| 1500-1600 | Sp / Tja10 / Angst | | | |
| 1600-1700 | Sp / Tja10 / Angst | | | |

Tabelle 4.5: Möglicher Stundenplan - Variante 1

| Uhrzeit | Turnhalle | Zimmer 101 | Zimmer 103 |
|---|---|---|---|
| 0800-0900 | Sp / Tja13 / Arm | D / Tja11 / Müller | |
| 0900-1000 | Sp / Tja13 / Arm | M / Tja11 / Müller | F / Tja12 / Pfister |
| 1000-1100 | Sp / Tja12 / Arm | D / Tja10 / Bernasconi | M / Tja13 / Angst |
| 1100-1200 | Sp / Tja12 / Arm | M / Tja10 / Angst | D / Tja13 / Bernasconi |
| 1300-1400 | Sp / Tja11 / Angst | F / Tja10 / Bernasconi | D / Tja12 / Müller |
| 1400-1500 | Sp / Tja11 / Angst | E / Tja10 / Pfister | M / Tja12 / Bernasconi |
| 1500-1600 | Sp / Tja10 / Angst | F / Tja11 / Müller | |
| 1600-1700 | Sp / Tja10 / Angst | E / Tja11 / Pfister | |

Tabelle 4.6: Möglicher Stundenplan - Variante 2

Eine sehr gut ausgearbeitete Software ist 'Units'[Gmb15], sie liefert umfangreiche Funktionen zur Erstellung von Stundenplänen.

#### Eingabe- und Ausgabedaten

Eingabedaten: Fächer, Lehrer, Klassen und Klassenzimmer mit verschiedenen Einschränkungen und Zusatzinformationen
Ausgabedaten: Einteilung der Fächer mit den dazugehörigen Klassen, Lehrern und Zimmer auf verschiedene Tage und Uhrzeiten, welche keine Konflikte enthält

#### Einfluss der Parameter auf die Komplexität

Beim Stundenplanproblem sind es nicht alle vier Listen, welche die Komplexität des Problems ausmachen. Den grössten Einfluss haben die Anzahl Stunden, welche verplant werden müssen. Die Anzahl Möglichkeiten berechnen sich wie folgt (vergleiche[Bra15]):

$$(AnzahlRäume * AnzahlZeitfenster * AnzahlLehrer)^{AnzahlVeranstaltungen}$$

**Bekannte Algorithmen** (siehe [GWB03])

- Backtracking Algorithmus
- Evolutionäre Algorithmen
- Constraint Logic Programming

**Bekannte ähnliche Probleme** Es gibt diverse andere Planungsprobleme, welche auf dem gleichen Konzept basieren, jedoch unterschiedliche Eingabedaten und Beschränkungen haben:

- Spielplan
- Prüfungsplan
- Schichtenplanung

### 4.2.5 Mathematisches Programmieren

### Rucksack-Problem

Das Rucksack-Problem ist ein NP-vollständiges Problem. Dies wurde durch die Reduktion des Problems der exakten Überdeckung bewiesen.

**Beschreibung** Englischer Name: Knapsack
Beim Rucksack-Problem geht es um einen Container, symbolisch der Rucksack, mit einer Gewichtsschranke. Zudem gibt es Objekte, welche in den Container gepackt werden können. Diese Objekte besitzen ein Gewicht und einen Profit. Das Ziel ist es, den grösstmöglichen Profit zu erlangen, ohne die Gewichtsschranke zu überschreiten. [Nag15]

**Beispiel** Gegeben sei ein Rakete mit der Gewichtsschranke 645 kg und die Objekte in Tabelle 4.7. Gesucht ist die Objektauswahl mit dem grössten Profit, welche die Gewichtsschranke nicht überschreitet.
In diesem Beispiel wäre die optimale Lösung 1, 2, 3 und 5. [Nag15]

| Objekt-Nr. | Gewicht in kg | Profit |
|---|---|---|
| 1 | 153 | 232 |
| 2 | 54 | 73 |
| 3 | 191 | 201 |
| 4 | 66 | 50 |
| 5 | 239 | 141 |
| 6 | 137 | 79 |
| 7 | 148 | 48 |
| 8 | 249 | 38 |

Tabelle 4.7: Knapsack Objekte mit Gewicht und Profit (Daten aus [Nag15])

**Eingabe- und Ausgabedaten**
Eingabedaten: Gewichtsschranke und Elemente mit Gewicht und Nutzwert
Ausgabedaten: Zusammenstellung der Elemente mit dem höchsten Nutzwert

**Einfluss der Parameter auf die Komplexität**
Die Komplexität des Rucksack-Problems hängt von der Anzahl der Objekte ab, die Gewichtsschranke trägt nicht zur Komplexität bei.

#### Bekannte Algorithmen

- Backtracking Algorithmus
- Greedy Algorithmus
- Algorithmus von Nemhauser und Ullmann [Nag15]

**Bekannte reale Probleme** Es gibt diverse reale Probleme, welche mit dem Rucksack-Problem gelöst werden können [KPP04], hier sind nur einige davon aufgelistet:

- Ausschneiden von verschiedenen Stücken aus einer Metall- oder Holzplatte: Optimale Nutzung der Fläche(n), damit am Schluss genug Einzelstücke für das Herstellen des Endproduktes vorhanden sind.
- Kreditvergabe: Optimale Ausnutzung des Kreditbudgets mit der Vergabe von Krediten an Kunden, welche einen Kredit über eine gewisse Höhe haben möchten und eine bestimmte Risikoeinstufung aufweisen.

### 4.2.6 Logik

Die in diesem Abschnitt aufgeführten Probleme sind der Vollständigkeit halber aufgelistet und beschrieben. Sie werden nicht für die weiteren Schritte der Schnittstelle verwendet. Sie sind die Grundpfeiler der NP-Vollständigkeit, auf welche zahlreiche Beweise der NP-Vollständigkeit von Probleme basieren.

#### Erfüllbarkeitsproblems der Aussagenlogik (SAT)

Das Erfüllbarkeitsproblem der *Aussagenlogik* ist ein NP-vollständiges Problem. Dies wurde durch Stephen A. Cook in den 1970er Jahren bewiesen [Coo71].

**Beschreibung** Englischer Name: Satisfiability (SAT)
Beim Erfüllbarkeitsproblem der Aussagenlogik ist zu überprüfen, ob eine beliebige aussagenlogische Formel erfüllbar ist.

**Beispiel** Gegeben sei eine aussagenlogische Formel 4.1.
Gesucht ist die Antwort auf die Erfüllbarkeit dieser Formel.
Die Tabelle 4.8 zeigt, dass die Formel für die Kombinationen 3, 4 und 8 erfüllbar ist.

$$(A \vee B) \wedge C \tag{4.1}$$

| Kombination-Nr. | A | B | C | $A \vee B$ | Resultat |
|---|---|---|---|---|---|
| 1 | wahr | falsch | falsch | wahr | falsch |
| 2 | wahr | wahr | falsch | wahr | falsch |
| **3** | wahr | falsch | wahr | wahr | **wahr** |
| **4** | wahr | wahr | wahr | wahr | **wahr** |
| 5 | falsch | falsch | falsch | falsch | falsch |
| 6 | falsch | wahr | falsch | wahr | falsch |
| 7 | falsch | falsch | wahr | falsch | falsch |
| **8** | falsch | wahr | wahr | wahr | **wahr** |

Tabelle 4.8: Wahrheitstabelle zur aussagenlogischen Formel

## 3-SAT

Das 3-SAT Problem ist ein NP-vollständiges Problem. Dies wurde durch die Reduktion des SAT Problems bewiesen.

**Beschreibung** Englischer Name: 3-Satisfiability (3-SAT)
Das 3-SAT Problem ist eine Spezialfall des SAT Problems, es dürfen maximal 3 Literale in einer Klausel enthalten sein.

**Beispiel** Gegeben sei eine 3-SAT Formel 4.2.
Gesucht ist die Antwort auf die Erfüllbarkeit dieser Formel.
Die Tabelle 4.9 beweist, dass die Formel erfüllbar ist, lediglich für die Kombinationen 5, 11, 13 und 15 ist sie nicht korrekt.

$$(A \wedge B \wedge C) \vee (B \wedge \neg C \wedge D) \tag{4.2}$$

| Kombination-Nr. | A | B | C | D | $A \wedge B \wedge C$ | $B \wedge \neg C \wedge D$ | Resultat |
|---|---|---|---|---|---|---|---|
| **1** | wahr | falsch | falsch | wahr | wahr | wahr | **wahr** |
| **2** | wahr | wahr | falsch | wahr | wahr | wahr | **wahr** |
| **3** | wahr | falsch | wahr | wahr | wahr | wahr | **wahr** |
| **4** | wahr | wahr | wahr | wahr | wahr | wahr | **wahr** |
| 5 | falsch | falsch | falsch | wahr | falsch | wahr | falsch |
| **6** | falsch | wahr | falsch | wahr | wahr | wahr | **wahr** |
| **7** | falsch | falsch | wahr | wahr | wahr | wahr | **wahr** |
| **8** | falsch | wahr | wahr | wahr | wahr | wahr | **wahr** |
| **9** | wahr | falsch | falsch | falsch | wahr | wahr | **wahr** |
| **10** | wahr | wahr | falsch | falsch | wahr | wahr | **wahr** |
| 11 | wahr | falsch | wahr | falsch | wahr | falsch | falsch |
| **12** | wahr | wahr | wahr | falsch | wahr | wahr | **wahr** |
| 13 | falsch | falsch | falsch | falsch | falsch | wahr | falsch |
| **14** | falsch | wahr | falsch | falsch | wahr | wahr | **wahr** |
| 15 | falsch | falsch | wahr | falsch | wahr | falsch | falsch |
| **16** | falsch | wahr | wahr | falsch | wahr | wahr | **wahr** |

Tabelle 4.9: Wahrheitstabelle zur 3-SAT Formel

## 4.3 Übersicht Eingabe- und Ausgabedaten [R2]

Die Eingabe- und Ausgabedaten der Probleme sind sehr unterschiedlich und weisen unterschiedliche Komplexität auf. Um einen besseren Überblick zu erhalten, wurden sie in der Tabelle 4.10 zusammengefasst.

| **Problem** | **Eingabedaten** | **Ausgabedaten** |
|---|---|---|
| Färbung (Graphtheorie) | 1. Knoten mit ihren Verbindungen zu anderen Knoten | 1. Knoten mit ihrer Färbung<br>2. $k$ (Anzahl benötigter Farben) |
| Problem des Handlungsreisenden | 1. Startpunkt<br>2. Wegpunkte, welche passiert werden müssen | 1. Wegpunkte in der Reihenfolge der optimalen Route<br>2. Länge der Strecke |
| Briefträgerproblem | 1. Startpunkt<br>2. Knoten mit ihren Verbindungen mit Gewichtung | 1. Knoten in der Reihenfolge der minimalen Strecke<br>2. Länge der Strecke |
| Stundenplan-Erstellung | 1. Fächer<br>2. Lehrer<br>3. Klassen<br>4. Klassenzimmer<br>5. Stundenplan Rahmenbedingungen | 1. Einteilung der Fächer mit den dazugehörigen Klassen, Lehrern und Zimmer auf verschiedene Tage und Uhrzeiten, welche keine Konflikte enthält |
| Rucksack-Problem | 1. Gewichtsschranke<br>2. Elemente mit Gewicht und Nutzwert | 1. Zusammenstellung von den Elementen mit höchstem Nutzwert<br>2. Nutzwert |

Tabelle 4.10: Eingabe- und Ausgabedaten der ausgewählten Probleme

## 4.4 Beispiel für den Einfluss der Parameter auf die Komplexität [R1a]

Bei den vorgestellten Problemen wurde der Einfluss der Parameter auf die Komplexität der Probleme erwähnt. Um dies zu verdeutlichen wurde ein praktisches Beispiel anhand des Rucksack-Problems erstellt.

### 4.4.1 Ausgangslage

Das Rucksack-Problem hat zwei Eingabeparameter, zum einen die Gewichtsschranke und zum anderen die Elemente. Im Beispiel wurden verschiedene Anzahl Elemente verwendet und dies mit unterschiedlichen Gewichtsschranken kombiniert. Zum Vergleich wurde die Zeit gestoppt, welche der Brute Force-Algorithmus benötigt, um die optimale Lösung herauszufinden. Bei dieser Methode wird das Ergebnis zwar durch andere Prozesse auf dem Testsystem verfälscht, reicht jedoch für eine Veranschaulichung der Einflüsse.

### 4.4.2 Resultat

Die Berechnungszeiten waren ungenau und unterschiedlich. Das Resultat bestätigt jedoch bei jedem Versuch die Theorie, dass die Komplexität nur durch die Anzahl an Elementen beeinflusst wird. Die verschiedenen Berechnungszeiten wurden in der Tabelle 4.11 festgehalten. Mit einer höheren Gewichtsschranke bleibt die Berechnungszeit in etwa gleich, bei einer Erhöhung der Anzahl Elemente nimmt die Komplexität jedoch sehr schnell zu.

| Anzahl an Elemente: / Gewichtsschranke: | **10** | **15** | **18** | **20** | **22** |
|---|---|---|---|---|---|
| **10** | 4ms | 23ms | 129ms | 1079ms | 7267ms |
| **100** | 1ms | 16ms | 149ms | 1761ms | 6054ms |
| **1000** | 1ms | 7ms | 595ms | 1038ms | 6331ms |
| **10000** | 1ms | 7ms | 48ms | 583ms | 7174ms |

Tabelle 4.11: Berechnungszeiten bei verschiedenen Eingabeparametern für das Rucksack-Problem

In Abbildung 4.5 wird die exponentielle Steigerung der Berechnungsdauer mit zunehmender Anzahl an Elementen sehr schön verdeutlicht.

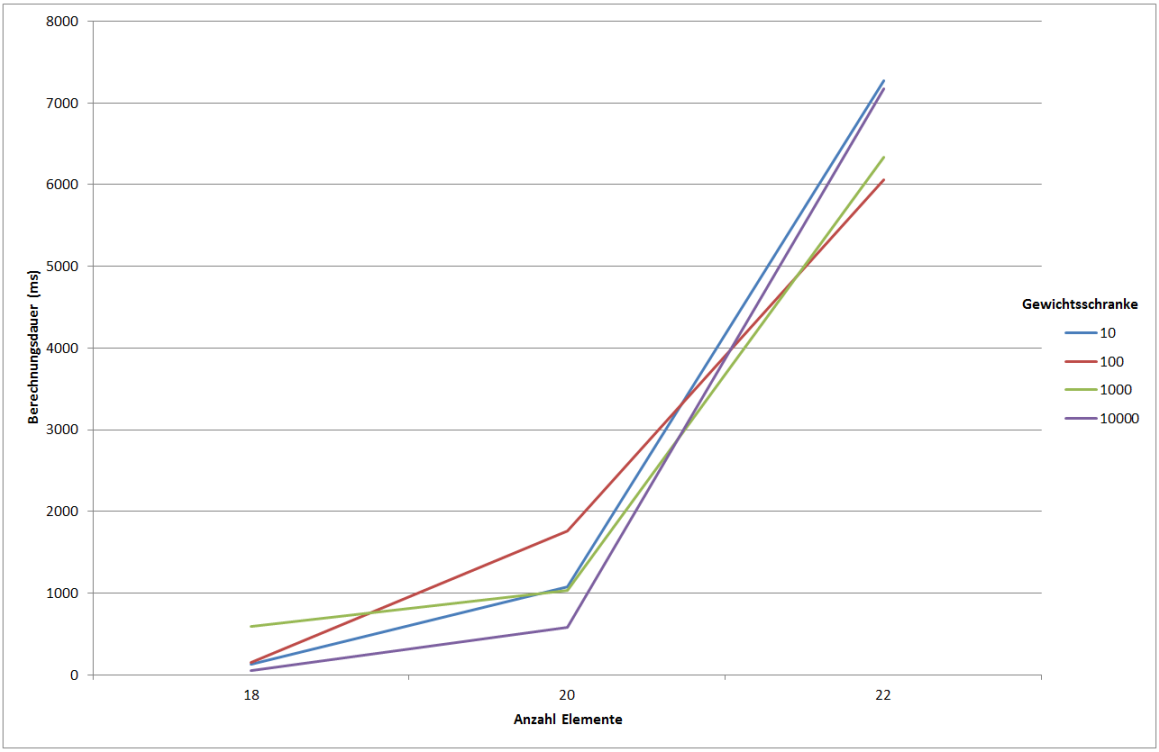



Abbildung 4.5: Berechnungszeiten für das Rucksack-Problem mit verschiedenen Eingabeparametern im Vergleich (eigene Darstellung)

# 5 Anforderungsdokument [R3]

Das Anforderungsdokument legt die Basis für die Implementation. Es ist für den Verlauf des Projekts wichtig, dass zu Beginn die Anforderungen aufgestellt werden. Bei der Erstellung des Anforderungsdokuments werden verschiedene Betrachtungsweisen aufgezeigt und die Anforderungen an das System in verschiedenen Detailstufen angeschaut.

## 5.1 Übersicht

In diesem Abschnitt wird die System- und Kontextabgrenzung dargelegt, die Systemumgebung beschrieben, die getroffenen Annahmen festgehalten und die verschiedenen Stakeholder mit ihren Erwartungen aufgelistet.

### 5.1.1 System- und Kontextabgrenzung

Der Systemkontext umfasst alle Aspekte, die für die Anforderungen des geplanten Systems relevant sind und nicht im Rahmen der Entwicklung dieses System gestaltet werden können. [PR11]

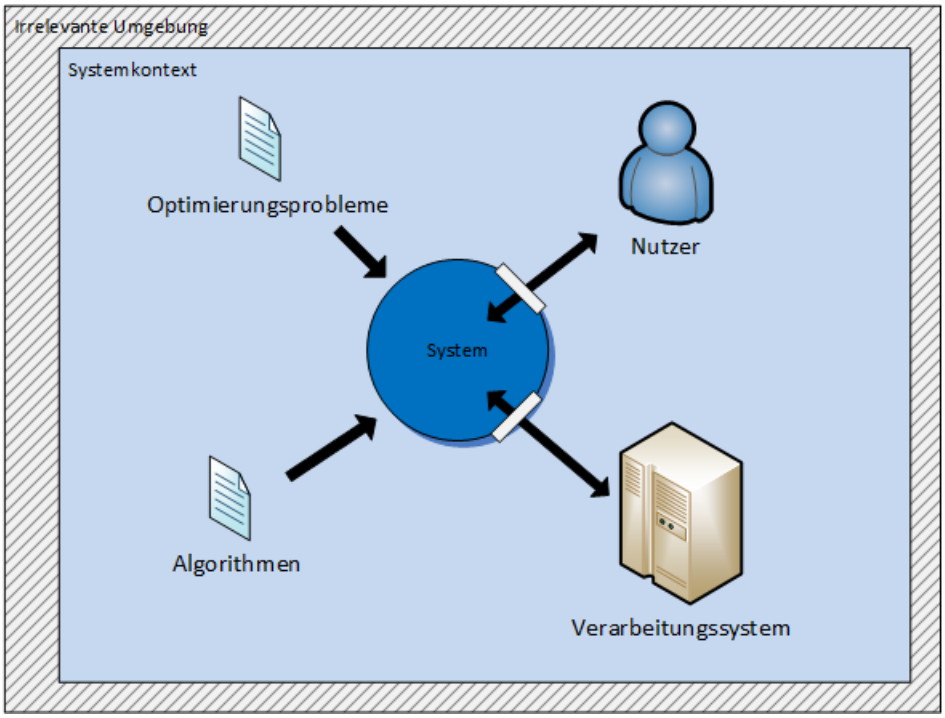



Abbildung 5.1: Systemkontext (eigene Darstellung)

Der Systemkontext (siehe Abbildung 5.1) zeigt, dass das System relevanten Schnittstellen zu den Nutzern und zum Verarbeitungssystem hat. Das System muss Daten für das Verarbeitungssystem zur Verfügung stellen und auch solche annehmen. Zudem wird das System von den Nutzern, Optimierungsproblemen und den dazugehörigen Algorithmen beeinflusst.

### 5.1.2 Systemumgebung

Die Systemumgebung (siehe Abbildung 5.2) definiert die Ausgangslage für das Projekt. Am Anfang des Projekts war bekannt, dass Nutzer und ein Verarbeitungssystem Dienste des zu erstellenden Systems beziehen wird. Die genaue Ausprägung dieser Dienste werden in diesem Kapitel behandelt.

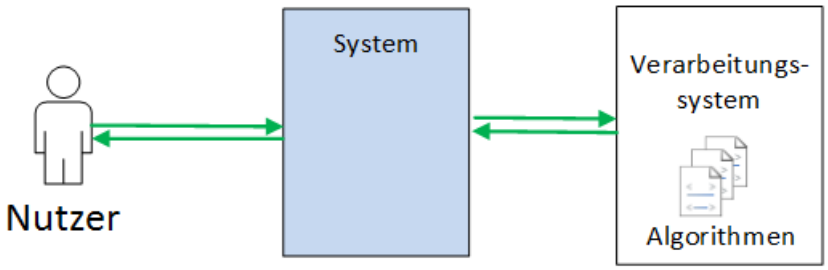



Abbildung 5.2: Systemumgebung (eigene Darstellung)

### 5.1.3 Annahmen

Zwischen dem Endnutzer und dem zu erstellenden System existiert noch eine Sicherheitsschicht, welcher nicht Teil dieser Arbeit ist. Später wird dieses Projekt allenfalls um diese Sicherheitsschicht erweitert oder eine übergeordnete Schnittstelle erstellt, welche diese anspricht.

Je nach Implementierung der Algorithmen ist das Ansteuern und die Aufbereitung der Daten unterschiedlich. Eine Referenzimplementierung zu jedem Problem zu finden, stellte sich als schwierig heraus. Deshalb wurden die Ein- und Ausgabe-Schemata der Algorithmen aus der Literaturrecherche abgeleitet.

### 5.1.4 Stakeholder

Die Stakeholder-Analyse dient dem Erfassen aller Nutzergruppen, die Einfluss auf das Projekt haben können. Zudem ermöglicht sie die Erfassung aller Gruppen, die potenziell Anforderungen an das Projekt stellen. In Tabelle 5.1 wurden die Stakeholder dieses Projekts zusammengetragen und ihre Erwartungen, ihre Einstellung und ihr Einfluss gegenüber dem Projekt festgehalten. Da es sich um eine Machbarkeitsanalyse handelt, befinden sich nur der Auftraggeber und ein einzelner potenzieller Kunde in der Auflistung.

| **Name** | **Erwartung** | **Einstellun** | **Einfluss** |
|---|---|---|---|
| | | -Positiv<br>-Neutral<br>-Negativ | -Hoch<br>-Mittel<br>-Niedrig |
| **Phil Hofmann** (Vorsteher der Geschäftsführung der 200ok GmbH) | Phil ist der Auftraggeber in diesem Projekt. Er erwartet von der Machbarkeitsanalyse Informationen für ein mögliches Projekt zur Umsetzung der Gesamtidee. | Positiv | Hoch |
| **Potenzieller Kunde** | Er wünscht sich eine einfache Abwicklung für seine Probleme, er möchte sich nicht mit Algorithmen und theoretischer Informatik herumschlagen. | Positiv | Hoch |

Tabelle 5.1: Liste der Stakeholder

## 5.2 Anforderungen

Zum Erfassen der Anforderungen an das System wurden zuerst verschiedene Use Cases definiert, mit deren Hilfe anschliessend der Anforderungskatalog erstellt werden konnte.

### 5.2.1 Use Cases

Das Use Case Diagramm (siehe Abbildung 5.3) zeigt einen Akteur, ein System und sechs Use Cases, welche für diese Arbeit relevant sind.

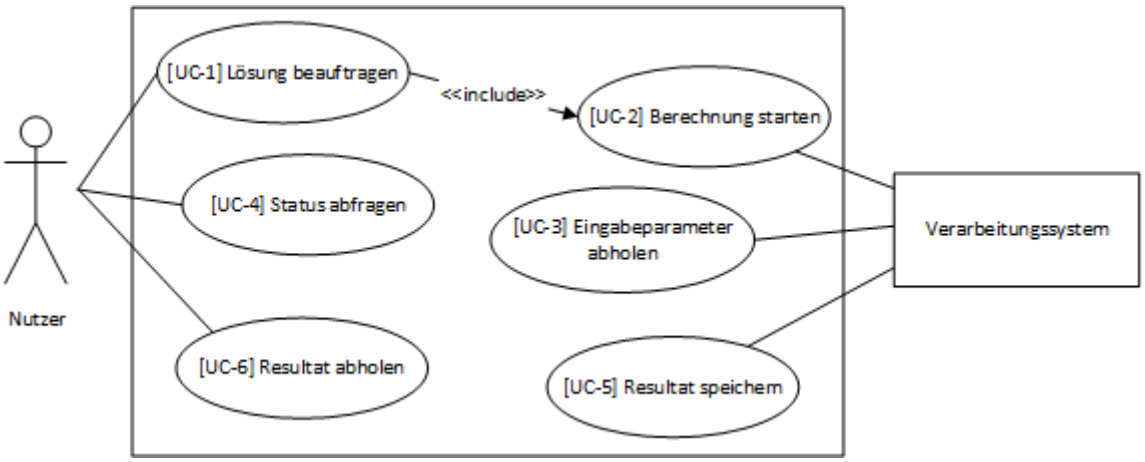



Abbildung 5.3: Use-Case Diagramm (eigene Darstellung)

Alle Use Cases wurden anhand der folgenden Vorlage (siehe Tabelle 5.2) spezifiziert. Diese Vorlage basiert auf Angaben von [PR11].

| **Bezeichner** | **Eindeutiger Bezeichner** |
|---|---|
| **Name** | Eindeutiger Name |
| **Beschreibung** | Komprimierte Beschreibung |
| **Auslösendes Ereignis** | Angabe des Ereignisses, das den Use Case auslöst. |
| **Akteure** | Auflistung der Akteure, die mit dem Use Case in Beziehung stehen. |
| **Vorbedingung** | Eine Liste notwendiger Voraussetzungen, die erfüllt sein müssen, bevor die Ausführung des Use Case beginnen kann. |
| **Nachbedingung** | Eine Liste von Zuständen, in denen sich das System unmittelbar nach der Ausführung des Hauptszenarios befindet. |
| **Ergebnis** | Beschreibung der Ausgaben, die während der Ausführung des Use Case erzeugt werden. |
| **Hauptszenario** | Beschreibung des Hauptszenarios eines Use Case |
| **Alternativszenarien** | Beschreibung von Alternativszenarien des Use Case oder lediglich Angabe der auslösenden Ereignisse. Hier gelten oftmals andere Nachbedingungen. |

Tabelle 5.2: Vorlage für Use Case Spezifikation

| **Bezeichner** | **UC-1** |
|---|---|
| **Name** | Lösung beauftragen |
| **Beschreibung** | Ein Nutzer möchte eine Lösung eines Problem mit spezifischen Parametern beauftragen. |
| **Auslösendes Ereignis** | Nutzer möchte ein Problem lösen. |
| **Akteure** | Nutzer |
| **Vorbedingung** | Das System bietet zur Lösung dieses Problems eine Schnittstelle. |
| **Nachbedingung** | Das System hat die nötigen Informationen für die Lösung des Problems und der Nutzer erhält eine ID, mit welcher er den Status bzw. das Resultat abfragen kann. |
| **Ergebnis** | Erfassung der Informationen für die Lösung des Problems |
| **Hauptszenario** | 1. Der Nutzer ruft die Funktion für das zu lösende Problem mit den Parametern auf.<br>2. Das System speichert die Parameter für die weitere Verarbeitung.<br>3. Der Nutzer erhält eine ID für das Abrufen des Status bzw. des Resultats. |
| **Alternativszenarien** | 3a Der Nutzer erhält eine Fehlermeldung, wenn das Problem nicht korrekt erfasst werden konnte. |

Tabelle 5.3: Use Case UC-1: Lösung beauftragen

| **Bezeichner** | **UC-2** |
|---|---|
| **Name** | Berechnung starten |
| **Beschreibung** | Das System startet die Berechnung beim Verarbeitungssystem. |
| **Auslösendes Ereignis** | Nutzer möchte ein Problem lösen. |
| **Akteure** | Verarbeitungssystem |
| **Vorbedingung** | Die Informationen für die Lösung des Problems sind erfasst. |
| **Nachbedingung** | Das Verarbeitungssystem beginnt mit der Berechnung. |
| **Ergebnis** | Starten der Berechnung |
| **Hauptszenario** | 1. Das System startet die Berechnung.<br>2. Das System übergibt eine ID für das Abrufen der abgelegten Daten. |
| **Alternativszenarien** | 2a Das System speichert die Fehlermeldung, falls das Starten der Berechnung fehlschlägt. |

Tabelle 5.4: Use Case UC-2: Berechnung starten

| **Bezeichner** | **UC-3** |
|---|---|
| **Name** | Eingabe Parameter abholen |
| **Beschreibung** | Das Verarbeitungssystem benötigt für die Lösung des Problems die Eingabeparameter. |
| **Auslösendes Ereignis** | Das Verarbeitungssystem startet eine neue Berechnung. |
| **Akteure** | Verarbeitungssystem |
| **Vorbedingung** | Das Verarbeitungssystem wurde angestossen, das Problem zu lösen. |
| **Nachbedingung** | Das Verarbeitungssystem hat die Eingabeparameter erhalten. |
| **Ergebnis** | Erhalt von Eingabeparametern |
| **Hauptszenario** | 1. Das Verarbeitungssystem fordert die Eingabeparameter für das zu lösende Problem an.<br>2. Das System leitet die Eingabeparameter weiter.<br>3. Das Verarbeitungssystem erhält die Eingabeparameter. |
| **Alternativszenarien** | 2a Das System liefert eine Fehlermeldung zurück, falls keine Eingabeparameter vorhanden sind. |

Tabelle 5.5: Use Case UC-3: Eingabe Parameter abholen

| **Bezeichner** | **UC-4** |
|---|---|
| **Name** | Status abfragen |
| **Beschreibung** | Der Nutzer kann den Status einer Berechnung abfragen, da die Verarbeitung einige Zeit benötigt. |
| **Auslösendes Ereignis** | Der Nutzer möchte den Status der Berechnung wissen. |
| **Akteure** | Nutzer |
| **Vorbedingung** | Der Nutzer hat bereits eine Berechnung beauftragt und kennt die ID. |
| **Nachbedingung** | Der Nutzer kennt den Status der Berechnung. |
| **Ergebnis** | Kenntnis des Status |
| **Hauptszenario** | 1. Der Nutzer fragt den Status einer Berechnung ab.<br>2. Das System fragt den Status ab.<br>3. Das System sendet den Status zurück.<br>4. Der Nutzer erhält den Status. |
| **Alternativszenarien** | 3a Das System sendet eine Fehlermeldung zurück, falls der Status nicht ermittelt werden kann. |

Tabelle 5.6: Use Case UC-4: Status abfragen

| **Bezeichner** | **UC-5** |
|---|---|
| **Name** | Resultat speichern |
| **Beschreibung** | Da die Aufrufe asynchron sind, muss das Ergebnis nach der Berechnung zwischengespeichert werden. |
| **Auslösendes Ereignis** | Das Verarbeitungssystem möchte das Resultat speichern. |
| **Akteure** | Verarbeitungssystem |
| **Vorbedingung** | Das Verarbeitungssystem hat ein Resultat berechnet. |
| **Nachbedingung** | Das Resultat ist gespeichert. |
| **Ergebnis** | Speicherung des Resultats |
| **Hauptszenario** | 1. Das Verarbeitungssystem schickt das Resultat der Berechnung an das System.<br>2. Das System erhält das Resultat.<br>3. Das System speichert das Resultat.<br>4. Das System quittiert das Erhalten des Resultats.<br>5. Das Verarbeitungssystem erhält die Bestätigung. |
| **Alternativszenarien** | 4a Das System liefert eine Fehlermeldung zurück, wenn das Resultat nicht korrekt gespeichert werden konnte. |

Tabelle 5.7: Use Case UC-5: Resultat speichern

| **Bezeichner** | **UC-6** |
|---|---|
| **Name** | Resultat abholen |
| **Beschreibung** | Der Nutzer holt das Resultat zu einem bestimmten Zeitpunkt ab. |
| **Auslösendes Ereignis** | Der Nutzer möchte das Resultat der Berechnung abholen. |
| **Akteure** | Nutzer |
| **Vorbedingung** | Der Nutzer hat bereits eine Berechnung beauftragt und kennt die ID. |
| **Nachbedingung** | Der Nutzer hat das Resultat erhalten. |
| **Ergebnis** | Erhalt des Resultats |
| **Hauptszenario** | 1. Der Nutzer fragt das Resultat der Berechnung ab.<br>2. Das System sucht das Resultat der Berechnung.<br>3. Das System sendet das Resultat.<br>4. Der Nutzer erhält das Resultat. |
| **Alternativszenarien** | 3a Das System sendet eine Fehlermeldung, wenn kein Resultat vorhanden ist. |

Tabelle 5.8: Use Case UC-6: Resultat abholen

### 5.2.2 Anforderungen

Alle Anforderungen wurden anhand der folgenden Vorlage (siehe Tabelle 5.9) erfasst. Diese Vorlage basiert auf Angaben von [PR11] und wurde um eigene Attribute erweitert.

| **Bezeichner** | **Eindeutiger Identifikator** |
|---|---|
| **Priorität** | Must, Should, Nice to have |
| **Anforderungstyp** | Funktionale Anforderung, Qualitätsanforderung, Randbedingung |
| **Name** | Eindeutiger, charakterisierender Name |
| **Use Case** | Referenz zum zugehörigen Use Case |
| **Beschreibung** | Beschreibung der Anforderung |
| **Begründung** | Bedeutung der Anforderung für das geplante System |
| **Akzeptanz Kriterium** | Messbare Abnahmekriterien |
| **Abhängigkeiten** | Referenz zu anderen Anforderungen |

Tabelle 5.9: Vorlage für Anforderungen

Die Beschreibung der Anforderungen wurden zusätzlich mit der Satzschablone in Abbildung 5.4) aus [PR11] erstellt. Dies hat den Vorteil, dass die Anforderungen normiert und exakt daherkommen. Die Abstufungen *muss*, *sollte* und *wird* werden verwendet, um die Wichtigkeit der Anforderungen auszudrücken.

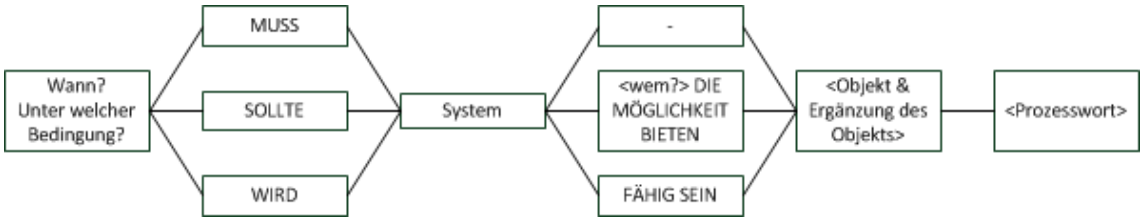



Abbildung 5.4: Satzschablone (Grafik entnommen aus [PR11])

### Funktionale Anforderungen

| **Bezeichner** | **RE-F1** |
|---|---|
| **Priorität** | Must |
| **Anforderungstyp** | Funktionale Anforderung |
| **Name** | Bereitstellung der Schnittstellen für Optimierungsprobleme |
| **Use Case** | Use Case UC-1: Lösung beauftragen |
| **Beschreibung** | Das System muss dem Nutzer die Möglichkeit bieten, die Lösung verschiedener Optimierungsprobleme zu beauftragen. |
| **Begründung** | Die Schnittstellen ist die Anlaufstelle des Nutzers. Er beauftragt das System, eine Berechnung zu starten. |
| **Akzeptanz Kriterium** | 1. Der Nutzer kann eine Schnittstelle für die bereitgestellten Berechnungsfunktionen ansprechen. |
| **Abhängigkeiten** | - |

Tabelle 5.10: Anforderung RF-F1

| **Bezeichner** | **RE-F2** |
|---|---|
| **Priorität** | Must |
| **Anforderungstyp** | Funktionale Anforderung |
| **Name** | Speicherung der Eingabeparameter |
| **Use Case** | Use Case UC-1: Lösung beauftragen |
| **Beschreibung** | Falls eine Berechnung in Auftrag gegeben wurde, muss das System fähig sein, die Eingabeparameter abzuspeichern. |
| **Begründung** | Die Berechnung wird von einem anderen System ausgeführt. Damit diese auf die Parameter zugreifen können, müssen die Daten persistiert werden. |
| **Akzeptanz Kriterium** | 1. Die Parameter sind persistiert.<br>2. Es gibt eine Fehlermeldung, falls bei der Speicherung etwas fehlschlägt oder die Eingabeparameter nicht gültig sind. |
| **Abhängigkeiten** | |

Tabelle 5.11: Anforderung RF-F2

| **Bezeichner** | **RE-F3** |
|---|---|
| **Priorität** | Must |
| **Anforderungstyp** | Funktionale Anforderung |
| **Name** | Rückgabe einer ID bei Beauftragung |
| **Use Case** | Use Case UC-1: Lösung beauftragen |
| **Beschreibung** | Falls eine Berechnung in Auftrag gegeben wurde, muss das System dem Ersteller eine ID zurückliefern. |
| **Begründung** | Die ID hilft dem Nutzer den Status der Berechnung abzuholen und wird am Schluss für das Resultat benötigt. |
| **Akzeptanz Kriterium** | 1. Der Nutzer erhält nach dem Starten einer Berechnung eine ID. |
| **Abhängigkeiten** | Anforderung RF-F2 |

Tabelle 5.12: Anforderung RF-F3

| **Bezeichner** | **RE-F4** |
|---|---|
| **Priorität** | Must |
| **Anforderungstyp** | Funktionale Anforderung |
| **Name** | Start der Berechnung |
| **Use Case** | Use Case UC-2: Berechnung starten |
| **Beschreibung** | Falls eine Berechnung in Auftrag gegeben wurde, muss das System fähig sein, die Berechnung beim Verarbeitungssystem zu starten. |
| **Begründung** | Der Nutzer kennt das Verarbeitungssystem nicht, das System muss dem Verarbeitungssystem den Start-Befehl geben. |
| **Akzeptanz Kriterium** | 1. Der Befehl für den Start wird versendet und die ID dabei übergeben.<br>2. Die Fehlermeldung bei einem Fehlversuch wird gespeichert. |
| **Abhängigkeiten** | Anforderung RF-F2 |

Tabelle 5.13: Anforderung RF-F4

| **Bezeichner** | **RE-F5** |
|---|---|
| **Priorität** | Must |
| **Anforderungstyp** | Funktionale Anforderung |
| **Name** | Abfrage der Eingabeparameter |
| **Use Case** | Use Case UC-3: Eingabe Parameter abholen |
| **Beschreibung** | Falls eine Berechnung in Auftrag gegeben wurde, muss das System dem Verarbeitungssystem die Möglichkeit bieten, die Eingabeparameter abzufragen. |
| **Begründung** | Damit das Verarbeitungssystem die Berechnung durchführen kann, braucht es die Eingabeparameter. |
| **Akzeptanz Kriterium** | 1. Das Verarbeitungssystem erhält die Eingabeparameter.<br>2. Das Verarbeitungssystem erhält eine Fehlermeldung, falls keine Eingabeparameter vorhanden sind. |
| **Abhängigkeiten** | Anforderung RF-F2 |

Tabelle 5.14: Anforderung RF-F5

| **Bezeichner** | **RE-F6** |
|---|---|
| **Priorität** | Should |
| **Anforderungstyp** | Funktionale Anforderung |
| **Name** | Abfrage des Status |
| **Use Case** | Use Case UC-4: Status abfragen |
| **Beschreibung** | Falls eine Berechnung in Auftrag gegeben wurde, sollte das System dem Nutzer die Möglichkeit bieten, den Status der Berechnung abzufragen. |
| **Begründung** | Da die Verarbeitung asynchron läuft, weiss der Benutzer nicht, wann seine Berechnung fertig ist. |
| **Akzeptanz Kriterium** | 1. Der Nutzer erhält einen Status seiner Berechnung. |
| **Abhängigkeiten** | Anforderung RF-F3 |

Tabelle 5.15: Anforderung RF-F6

| **Bezeichner** | **RE-F7** |
|---|---|
| **Priorität** | Nice to have |
| **Anforderungstyp** | Funktionale Anforderung |
| **Name** | Registrierung eines WebHooks für Statusänderungen |
| **Use Case** | Use Case UC-4: Status abfragen |
| **Beschreibung** | Falls eine Berechnung in Auftrag gegeben und dazu ein WebHook eingetragen wurde, sollte das System den Nutzer über eine Änderung des Status mittels WebHook informieren. |
| **Begründung** | Da die Verarbeitung asynchron läuft, weiss der Benutzer nicht, wann seine Berechnung fertig ist. Um ein ständiges Pollen zu vermeiden, können WebHooks verwenden werden. |
| **Akzeptanz Kriterium** | 1. Der Nutzer wird über die Änderung des Status auf dem eingetragen WebHook informiert. |
| **Abhängigkeiten** | Anforderung RF-F3 |

Tabelle 5.16: Anforderung RF-F7

| **Bezeichner** | **RE-F8** |
|---|---|
| **Priorität** | Must |
| **Anforderungstyp** | Funktionale Anforderung |
| **Name** | Speicherung des Resultats |
| **Use Case** | Use Case UC-5: Resultat speichern |
| **Beschreibung** | Nach der Berechnung muss das System dem Verarbeitungssystem die Möglichkeit bieten, das Resultat abspeichern zu können. |
| **Begründung** | Das Resultat muss, bis der Nutzer es abholt, zwischengespeichert werden. |
| **Akzeptanz Kriterium** | 1. Das Verarbeitungssystem kann das Resultat abspeichern.<br>2. Das Verarbeitungssystem erhält eine Fehlermeldung, falls das Speichern fehlgeschlagen ist. |
| **Abhängigkeiten** | Anforderung RF-F4 |

Tabelle 5.17: Anforderung RF-F8

| **Bezeichner** | **RE-F9** |
|---|---|
| **Priorität** | Must |
| **Anforderungstyp** | Funktionale Anforderung |
| **Name** | Abfrage des Resultats |
| **Use Case** | Use Case UC-6: Resultat abholen |
| **Beschreibung** | Das System muss dem Nutzer die Möglichkeit bieten, das Resultat der Berechnung abzufragen. |
| **Begründung** | Der Nutzer möchte das Resultat der Berechnung wissen. |
| **Akzeptanz Kriterium** | 1. Der Nutzer erhält das Resultat der Berechnung.<br>2. Der Nutzer erhält eine entsprechende Fehlermeldung, wenn beim Bereitstellen des Resultats ein Fehler aufgetreten ist. |
| **Abhängigkeiten** | Anforderung RF-F3 |

Tabelle 5.18: Anforderung RF-F9

### Qualitätsanforderung

| **Bezeichner** | **RE-NF1** |
|---|---|
| **Priorität** | Should |
| **Anforderungstyp** | Qualitätsanforderung |
| **Name** | Prozess-agnostische Schnittstelle |
| **Use Case** | Use Case UC-1: Lösung beauftragen |
| **Beschreibung** | Das System sollte fähig sein, die Lösung eines Problems so bereitzustellen, dass kein Wissen über den Verarbeitungsprozess erforderlich ist. |
| **Begründung** | Der Verarbeitungsprozess kann spezifisch und von Problem zu Problem unterschiedlich sein, der Nutzer sollte eine möglichst einfache Schnittstelle dafür haben. |
| **Akzeptanz Kriterium** | 1. Das Interface kann verwendet werden, ohne dass das Verarbeitungssystem bekannt ist. |
| **Abhängigkeiten** | - |

Tabelle 5.19: Qualitätsanforderung RF-NF1

| **Bezeichner** | **RE-NF2** |
|---|---|
| **Priorität** | Should |
| **Anforderungstyp** | Qualitätsanforderung |
| **Name** | Entgegennahme generischer Eingabeparameter |
| **Use Case** | Use Case UC-1: Lösung beauftragen |
| **Beschreibung** | Das System sollte in der Lage sein, unterschiedliche Ausprägungen von Eingabeparametern entgegenzunehmen. |
| **Begründung** | Da es bei den Problemen unterschiedliche Ausprägungen gibt, ist auf eine generische Deserialisierung der Eingabeparameter hinzuarbeiten. |
| **Akzeptanz Kriterium** | 1. Unterschiedliche Ausprägungen eines Problems benutzen die gleiche API. |
| **Abhängigkeiten** | - |

Tabelle 5.20: Qualitätsanforderung RF-NF2

| **Bezeichner** | **RE-NF3** |
|---|---|
| **Priorität** | Should |
| **Anforderungstyp** | Qualitätsanforderung |
| **Name** | Speicherung generischer Eingabeparameter |
| **Use Case** | Use Case UC-1: Lösung beauftragen |
| **Beschreibung** | Das System sollte Eingabeparameter einheitlich abspeichern. |
| **Begründung** | Da es viele unterschiedliche Probleme gibt, ist eine generische Persistierung anzustreben. |
| **Akzeptanz Kriterium** | 1. Unterschiedliche Probleme haben kein abweichendes Persistierungsschema. |
| **Abhängigkeiten** | - |

Tabelle 5.21: Qualitätsanforderung RF-NF3

### 5.2.3 Zusammenfassung der Anforderungen

Die Priorität der einzelnen Anforderungen ist wichtig, falls nicht alle Anforderungen umgesetzt werden können. Die Priorität wurde zusammen mit den Stakeholdern festgelegt und in Tabelle 5.22 zur besseren Übersicht zusammengetragen.

| **Bezeichner** | **Name** | **Priorität** |
|---|---|---|
| RE-F1 | Bereitstellung der Schnittstellen für Optimierungsprobleme | Must |
| RE-F2 | Speicherung der Eingabeparameter | Must |
| RE-F3 | Rückgabe einer ID bei Beauftragung | Must |
| RE-F4 | Start der Berechnung | Must |
| RE-F5 | Abfrage der Eingabeparameter | Must |
| RE-F6 | Abfrage des Status | Should |
| RE-F7 | Registrierung eines WebHooks für Statusänderungen | Nice to have |
| RE-F8 | Speicherung des Resultats | Must |
| RE-F9 | Abfrage des Resultats | Must |
| RE-NF1 | Prozess-agnostische Schnittstelle | Should |
| RE-NF2 | Entgegennahme generischer Eingabeparameter | Should |
| RE-NF3 | Speicherung generischer Eingabeparameter | Should |

Tabelle 5.22: Priorität der Anforderungen

# 6 Konzept der Schnittstelle [R4]

In diesem Kapitel wird auf die Strukur und das Konzept der Schnittstelle eingegangen. Dieses Konzept legt die Grundlage für die Umsetzung und entscheidet somit, ob die Aufgabe gelöst werden kann.

## 6.1 Übersicht

Die Systemumgebung (siehe Abbildung 6.1) hat zwei Berührungspunkte zur Aussenwelt. Der eine ist zum Nutzer hin, der andere zu einem Verarbeitungssystem. Um die Daten zu speichern, wird eine Datenbank benötigt. Das ermöglicht einen asynchronen Ablauf und ein mehrfaches Abfragen der Daten.

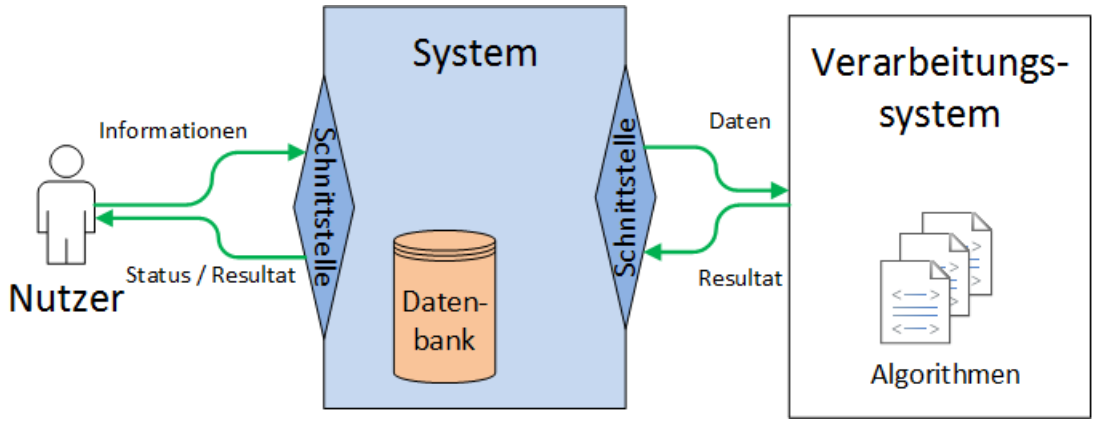



Abbildung 6.1: System Übersicht (eigene Darstellung)

## 6.2 Konzept

Das Konzept sollte simpel und flexibel sein, damit das System möglichst schnell erweitert werden kann. Beim Betrachten der Probleme und der Abläufe wurde bemerkt, dass die Vorgänge Ähnlichkeiten aufweisen. Die Daten werden angenommen und für einen bestimmten Algorithmus aufbereitet. Das Verarbeitungssystem schickt das Resultat zurück und dieses wird dann wieder umgewandelt, so dass es für den Nutzer brauchbar ist. Es finden zwei Umwandlungen statt. Die Umwandlungen an sich sind von Problem zu Problem verschieden, es gibt jedoch auch dort gewisse Ähnlichkeiten. In Abbildung 6.2 wird der Aufbau des Systems gezeigt. Die *Domänensprache* des Nutzers wird mit Hilfe der Controller, Entities und den Translators auf die Domänensprache der Algorithmen abgebildet.

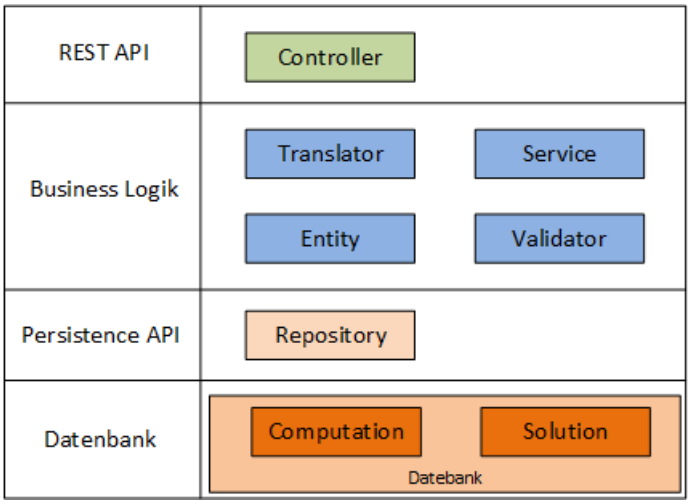



Abbildung 6.2: Architekturaufbau des Systems (eigene Darstellung)

### 6.2.1 REST API

Für die Schnittstelle wird ein *Representational State Transfer* (REST) *Application Programming Interface* (API) erstellt, welches die nötigen Funktionen bietet. Das API funktioniert nach dem De-facto-Standard (siehe [wik15b]). Falls Fehler auftreten, werden die *HTTP-Statuscode* verwendet, um diese an den Aufrufer weiterzugeben.

### 6.2.2 Business Logik

Die Business Logik besteht aus Translator, Service, Entity und Validator von den jeweiligen Problemen.

**Translator** Es gibt jeweils ein 'ComputationTranslator' für die Umwandlung vom Nutzer hin zum Algorithmus und einen 'SolutionTranslator' für die Umwandlung vom Algorithmus zurück in das System. In den Translators steckt die ganze Logik. Hier fliesst ein, wie der Algorithmus die Daten für die Verarbeitung benötigt und wie das Resultat zurück kommt. Es wäre zum Beispiel möglich, die Daten in Kombinationen für einen evolutionären Algorithmus umzuwandeln, so würde allenfalls ein generischer Algorithmus für alle Probleme ausreichen. Ebenfalls möglich wäre eine Umwandlung des Stundenplanproblems auf ein Knotenfärbungsproblem und somit könnte der gleiche Algorithmus angesprochen werden (vergleiche [Abd06]). Die Möglichkeiten mit dem Konzept der Translators ist sehr vielfälltig. Zusätzlich kann im Translator das Resultat mit zusätzlichen Informationen, zum Beispiel Statistiken, angereichert werden.

**Service** Der Service kann sehr generisch gehalten werden und benötigt keine problemspezifisches Methoden.

**Entity** Die Entitäten sind von Problem zu Problem unterschiedlich. Es muss analysiert werden, wie die Parameter am besten eingegeben werden und wie diese dann vom Algorithmus gebraucht werden.

**Validator** Der Validator entscheidet, ob eine Lösung gültig ist oder nicht. Wie bereits im Abschnitt 4.2 erklärt, kann jedes NP-vollständige Problem in polynomialer Zeit validiert werden.

### 6.2.3 Persistance API

Die Abstraktion der Datenbank wird mittels eines Persistance APIs, welches mit der Datenbank interagiert, realisiert. Dieses API ist für das Laden und Speichern der Daten verantwortlich und

bietet die Möglichkeit, spezifische Abfragen auszuführen.

### 6.2.4 Datenbank

Jedes Problem hat seine eigene Ausprägung von Computation und Solution, welche abgespeichert werden müssen. Die Datenbank sollte, wenn möglich, eine ähnliche Flexibilität, wie das Programm selber, aufweisen. Die Vorgänge benötigen keine Transaktionen und sind zum grossen Teil nur Einfüge-Operationen, nur selten wird ein Eintrag geändert. Die Daten werden immer in der Nutzer-Sicht gespeichert.

### 6.2.5 Ablauf

In Abbildung 6.3 wird der Ablauf des ganzen Vorganges und die Interaktion mit dem Nutzer und dem Verarbeitungssystem verdeutlicht. Die Translators, welche in dem Prototyp verwendet werden, sind nur eine Möglichkeit, welche dieses Konzept bietet. Generell baut dieses Konzept auf eine pre- und post-Aktion vor bzw. nach dem Starten des Algorithmus auf.

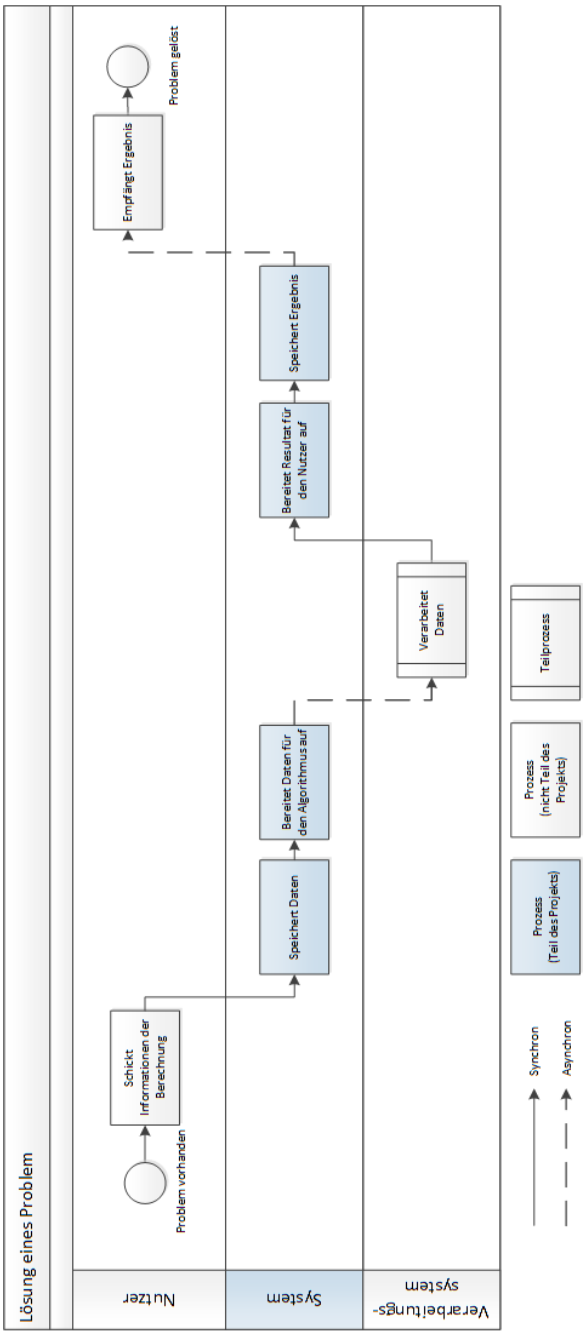



Abbildung 6.3: Flussdiagramm des Arbeitsablaufs (eigene Darstellung)

Die Abbildung 6.4 zeigt das Sequenzdiagramm für den Start eines beliebigen Problems, alle Komponenten mit '{Problem}' sind spezifische Problem-Komponenten, die anderen sind generische. Bei Speichern einer Berechnung wird der Status 'CREATED' gesetzt. Nach dem Speichern wird über die 'Solver'-Komponente asynchron das Verarbeitungssystem gestartet und der Status auf 'STARTED' gesetzt. Das Verarbeitungssystem holt sich die benötigten Informationen. Der Controller lädt das Problem vom Service, dieser wiederum lädt es vom Repository und wandelt es für den Algorithmus um.

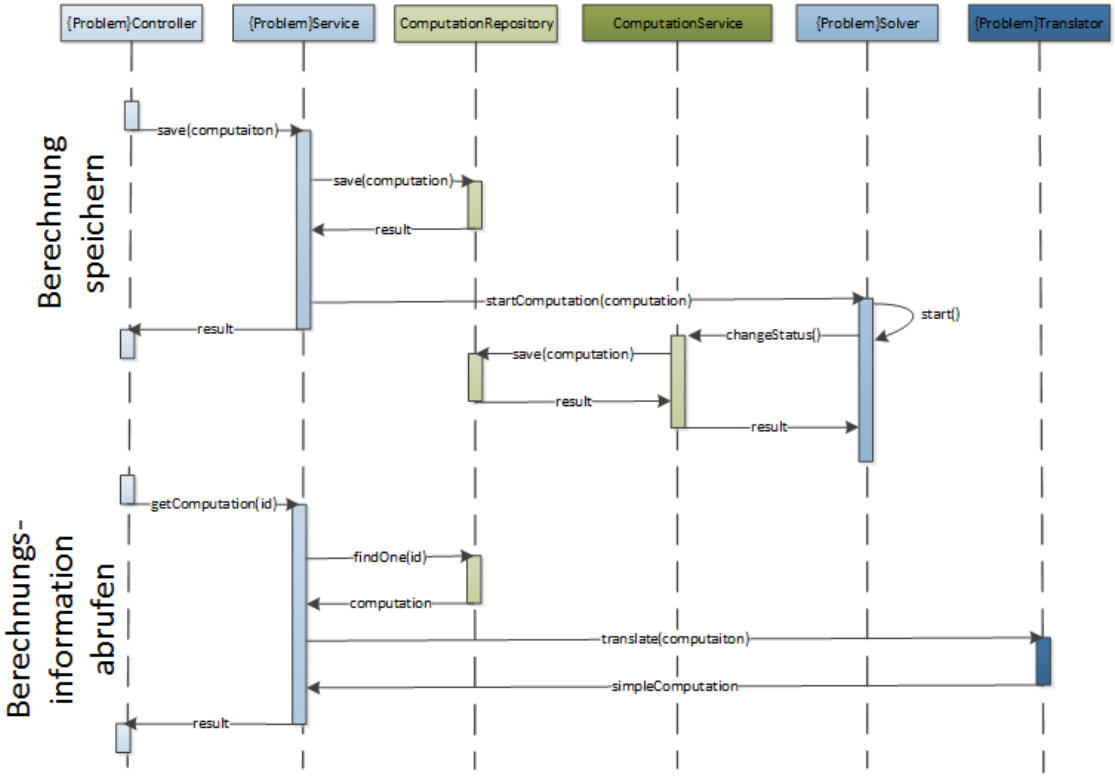



Abbildung 6.4: Start eines beliebigen Problemes (eigene Darstellung)

Die Abbildung 6.5 zeigt das Sequenzdiagramm für das Abspeichern eines Resultates einer beliebigen Berechnung. Bei einem Speicheraufruf wird zuerst das Problem geladen, danach wird die Lösung vom Algorithmus mit Hilfe der Eingabeparameter transferiert und vom Validator validiert. Nun wird anhand des Resultattypes der Status der Berechnung abgeändert. Das Resultat wird mit dem Ergebnis der Validierung in die Datenbank gespeichert. Wenn der Nutzer den Status einer Berechnung abruft, fragt der Controller über den Service den Status ab. Der Service lädt das Problem, fragt die vorhandenen Resultate ab, fügt diese zu einem Status zusammen und schickt den Status an den Controller zurück.

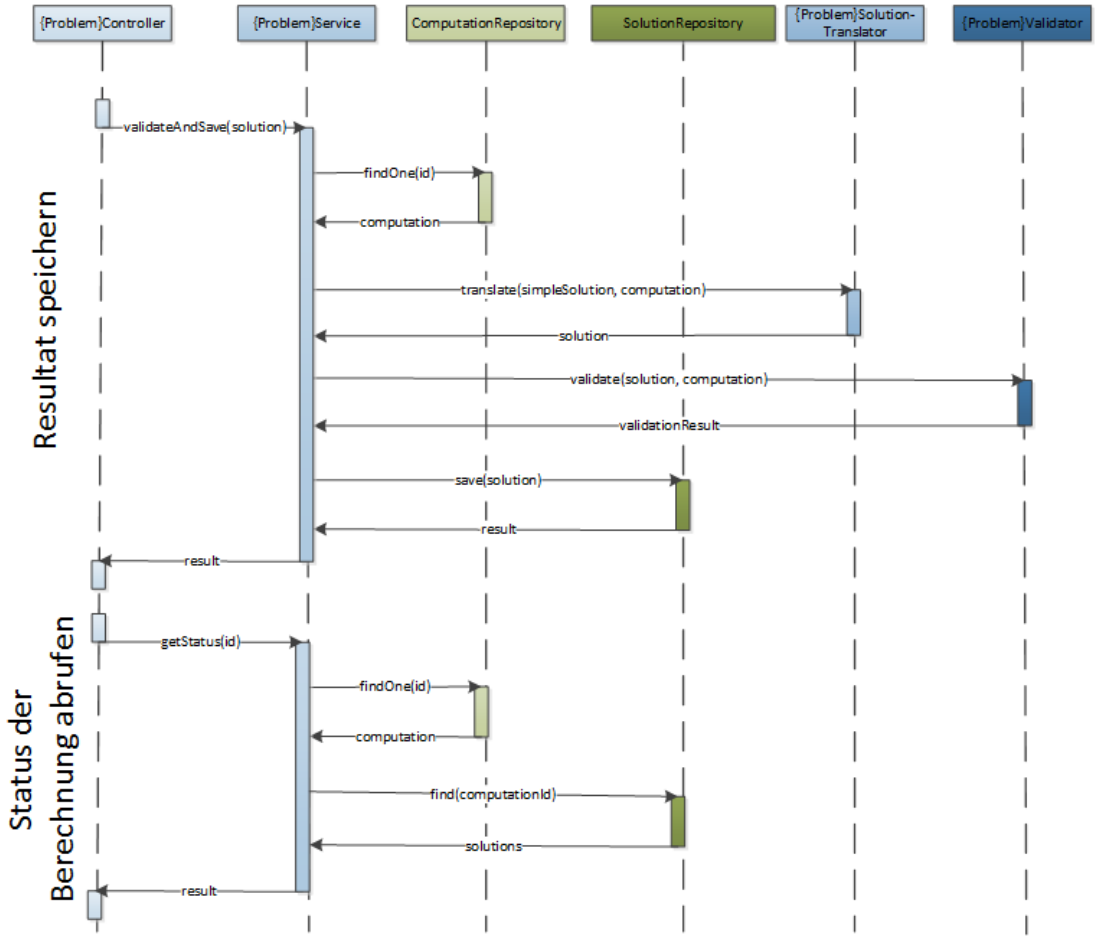



Abbildung 6.5: Abspeichern des Resultates eines beliebigen Problemes (eigene Darstellung)

## 6.3 Datenbank Varianten

Es gibt verschiedene Datenbanktypen und jede hat seine Vor- und Nachteile. In diesem Abschnitt werden vier verschiedenen Typen miteinander verglichen und der beste für diesen Anwendungszweck ausgewählt. Um die Eigenheiten der Datenbanken hervorzuheben, wird bei jeder Art das gleiche Beispiel mit der spezifischen Definitions– und Abfragesprache gemacht.

### 6.3.1 CAP-Theorem

Das CAP-Theorem ist im Jahr 2000 aus einer Vermutung von Eric Brewer entstanden [Bre00]. Das Theorem zeigt, dass die Werte Konsistenz (C), Verfügbarkeit (A) und Partitionstoleranz (P) ein Dreieck (siehe Abbildung 6.6) bilden und dass ein verteiltes System nur jeweils zwei dieser Eigenschaften gleichzeitig erfüllen kann.

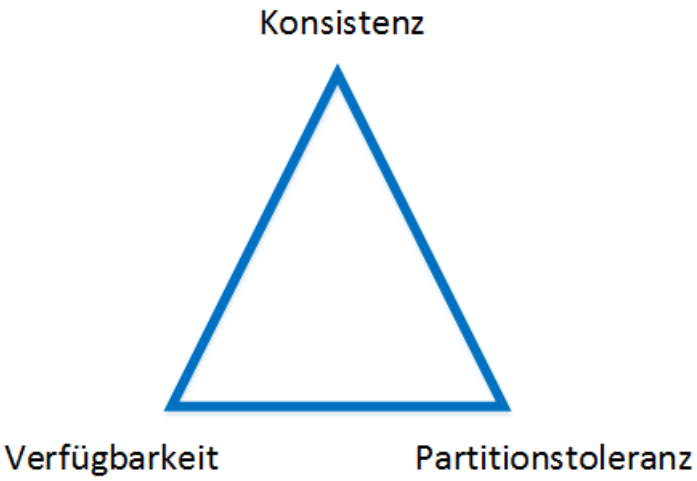



Abbildung 6.6: CAP-Theorem (eigene Darstellung)

### Beispiele

- **AP**: Domain Name System (DNS), NoSQL Datenbanken
- **CA**: Relationalen Datenbanken
- **CP**: Banken Anwendungen

## 6.3.2 Relationales Datenbanksystem

Eine *relationale Datenbank* (RDBMS) basiert auf Transaktionen und das dazugehörige *Atomicity, Consistency, Isolation, Durability* (ACID) Prinzip (vergleiche [Lim10]).

- **Atomicity**: Eine Reihe von Befehlen wird entweder ganz oder gar nicht ausgeführt.
- **Consistency**: Der Datenzustand ist nach jeder Veränderung wieder konsistent, wenn dies auch vorher der Fall war.
- **Isolation**: Unterschiedliche Befehlsketten beeinflussen sich nicht gegenseitig.
- **Durability**: Vollzogene Änderungen sind dauerhaft, auch bei einem Systemfehler.

Beziehungen zwischen Tabellen werden mit Fremdschlüsseln definiert, welche auf den Hauptschlüssel der referenzierten Tabelle zeigt. Listing 6.1 zeigt die Tabelle 'Person', welche eine Verbindung zur Tabelle 'Address' hat.

```
1   CREATE TABLE ADDRESS(
2             ID        INTEGER(11),
3             Street    VARCHAR(30),
4             Zip       INTEGER(6),
5             City      VARCHAR(12));
6
7   CREATE TABLE PERSON(
8               ID           INTEGER(11),
9               Name         VARCHAR(50),
10              FK_Address   INTEGER(11));
```

Listing 6.1: Tabellendefinition in relationalem Datenbanksystem

Die Abfrage der Strasse bei einer relationaler Datenbank mittels *Structured Query Language* (SQL) ist in Listing 6.2 dargestellt.

```
1  SELECT ADDRESS.Street
2  FROM PERSON
3  JOIN ADDRESS on PERSON.FK_Address = ADDRESS.ID;
4  WHERE PERSON.Name = 'Thomas';
```

Listing 6.2: Abfrage in relationalem Datenbanksystem

### 6.3.3 Objektrelationales Datenbanksystem

Objektrelationale Datenbankmanagementsysteme (ORDBMS) wurden entwickelt, um das relationale System mit den Funktionen des objektorientierten Stiles zu erweitern. Die folgende Beschreibung der Eigenschaften ist aus [Lim10] abgeleitet. In objektrelationalen Datenbanksystemen werden komplexe Datentypen unterstützt, so können zusätzliche Typen definiert werden und diese können dann bei einer Tabellendefinition verwendet werden. In Listing 6.3 ist eine Typendefinition von 'AddressType' zu sehen, welche wiederum in Listing 6.4 in der Tabelle 'Person' verwendet wird.

```
1  CREATE TYPE AddressType  AS
2           (Street    VARCHAR(30),
3            Zip       INTEGER(6),
4            City      VARCHAR(12));
```

Listing 6.3: Typendefinition in objektrelationalem Datenbanksystem

```
1  CREATE TABLE PERSON(
2              ID        VARCHAR(20),
3              Name      VARCHAR(50),
4              Address   AddressType);
```

Listing 6.4: Verwendung von Typendefinition in objektrelationalem Datenbanksystem

Wenn nun die Strasse einer bestimmten Person abgefragt werden möchte, sieht dies in einer objektrelationaler Datenbank wie in Listing 6.5 aus.

```
1  SELECT Address.Street
2  FROM PERSON;
3  WHERE Name = 'Thomas';
```

Listing 6.5: Abfrage in objektrelationalem Datenbanksystem

Zusätzlich können auch Arrays von Typen definiert werden, in Listing 6.6 ist die Definition der Tabelle 'Person' mit bis zu fünf E-Mail-Adressen gezeigt.

```
1  CREATE TABLE PERSON(
2              ID        VARCHAR(20),
3              Name      VARCHAR(50),
4              Mail      VARCHAR(40) ARRAY[5]);
```

Listing 6.6: Verwendung von Array in objektrelationalem Datenbanksystem

Weiter bieten die Systeme die Möglichkeiten, Methoden auf Typen und Tabellen und Vererbungshierarchien zu definieren. Der Anwendungsbereich liegt bei Applikationen, welche viele kurzlebige Transaktionen mit komplexen Objekten durchführen.

### 6.3.4 Objekorientiertes Datenbanksystem

Eine *objektorientierte Datenbank* (OODBMS) hat als Ziel eine bessere und nähere Zusammenarbeit mit objektorientierten Sprachen. Die Informationen über OODBMS sind aus [Lim10] entnommen. Der grosse Unterschied zu anderen Datenbanksystemen ist die Persistierung, bei OODBMS wird bereits beim Erstellen eines neuen persistenten Objekts im Code ein Pointer auf das Objekt in der Datenbank zurückgegeben. Weiter unterstützen die Systeme Versionierung von Objekten, die Datenbank kann somit mit verschiedenen Versionen der Objekte gleichzeitig umgehen. Dies kann bei

einer geplanten Umstrukturierung sehr hilfreich sein, wenn die neue Version erst für Tests verwendet wird und die bestehenden Objekte erst nach dem Release migriert werden sollen. Listing 6.7 zeigt, wie das Objekt 'Person' in *Object Definition Language* (ODL) definiert wird.

```
class PERSON
{
      attribute string Id;
      attribute string Name;
      attribute struct Address
          { string Street,
            short Zip,
            string City} address;
}
```

Listing 6.7: Objektdefinition in objektorientierem Datenbanksystem

Für die Abfrage der Strasse einer Person wird ein Statement wie in Listing 6.8 verwendet. Diese Abfragesprache nennt sich *Object Query Language* (OQL). Die Attribute der Objekte und Structs können mit einem Punkt separiert verkettet werden.

```
SELECT p.Address.Street
FROM persons p
WHERE p.Name = 'Thomas';
```

Listing 6.8: Abfrage in objektorientierem Datenbanksystem

Die objektorientierten Datenbanksystemen bietet die aus den RDBMS bekannten Abfragefunktionen (Avg, Max, Min, Distinct). Zusätzlich können bei diesen Systemen Vererbungshierarchien und Methoden in Objekten definiert werden. Sie werden bei Applikationen verwendet, welche viele und lange Transaktionen mit komplexen Objekten haben.

### 6.3.5 NoSQL Datenbanksystem

NoSQL Datenbanksysteme folgen nicht dem ACID Prinzip, sie sind entwickelt worden, um den erforderlichen Geschwindigkeiten und Grössen von Anwendungen wie Google, Facebook, Yahoo und Twitter zu genügen. Die Systeme sind nicht relational aufgebaut und verwenden oft eine andere Abfragesprache als SQL (siehe [Vai13]). Die meisten NoSQL Datenbanksystem setzen auf das *Basically Available, Soft state, Eventual consistency* (BASE)-Prinzip von Eric Brewer.

> „**Basic availability**: Each request is guaranteed a response—successful or failed execution.
> **Soft state**: The state of the system may change over time, at times without any input (for eventual consistency).
> **Eventual consistency**: The database may be momentarily inconsistent but will be consistent eventually.“ (Gaurav Vaish [Vai13])

Nahe zu alle NoSQL Datenbanksysteme sind schemalos, dass heisst, die Tabellen müsse nicht im Vorherein definiert werden. Die Abfragen gestalten sich einfach, JOIN Befehle entfallen komplett. Durch den lockeren Aufbau kann eine höhere Performance erzielt werden, die Unterschiede bei der Geschwindigkeit werden bereits bei kleineren Datenmengen bemerkbar.

#### NoSQL Datenbanktypen

NoSQL ist eine Übergruppe von unterschiedlichen Datenbanktypen. In diesem Abschnitt wird auf drei verschiedene NoSQL Datenbanktypen eingegangen, die Eigenschaften der jeweiligen Typen sind aus [Vai13] entnommen.

**Document** Bei dokumentorientierten Datenbanken werden *semistrukturierte Daten* abgespeichert, diese verwenden meistens die Dateiformate *JavaScript Object Notation* (JSON), *Extensible Markup Language* (XML) oder *YAML*. Die meisten Datenbanksysteme dieser Art haben kein definiertes Schema für Einträge oder nur ein teilweise definiertes Schema. Es ist somit möglich ganz unterschiedliche Elemente in eine Tabelle bzw. Collection, so werden Tabellen in dieser Welt genannt, zu speichern. Diese Erklärung erinnert an *Binary Large Object* (BLOB) in relationalen Datenbanken, jedoch können hier Indexes auf bestimmte Attribute gesetzt und innerhalb der Elemente effizient gesucht werden.

In Listing 6.9 wird ein Personen Element gezeigt, welches in der Collection 'Persons' gespeichert ist. Dieses Beispiel ist basierend auf einer MongoDB Instanz, die Definition für die Collection entfällt komplett.

```
{
    _id: 1,
    name: "Thomas",
    address: {
                street: "Bahnhofstrasse 45",
                zip: 8315,
                city: "Lindau"}
}
```

Listing 6.9: Personen Element in JSON Format

Listing 6.10 zeigt die Abfrage der Strasse bei MongoDB, welche das Resultat aus Listing 6.11 liefert. Bei MongoDB wird bei einer Abfrage standardmässig immer das ganze Element zurückgegeben, falls nur gewiese Attribute abgefragt werden sollen, können diese in dem Projektionsabschnitt definiert werden. Die ID des Elements wird, so lang sie nicht explizit ausgeschlossen wird, immer mit selektiert.

```
db.persons.find({name: "Thomas"}, {_id: 0, "address.street": 1})
```

Listing 6.10: Abfrage in MongoDB

```
{
    address: {
                street: "Bahnhofstrasse 45"}
}
```

Listing 6.11: Resultat der Abfrage in MongoDB

**Column-oriented** Anders als bei relationalen Datenbanken, welche zeilenorientierte sind, werden die Daten spaltenorientiert gespeichert. Eine Lohndatenbank würde serialisiert beim zeilenorientierten Konzept wie in Listing 6.12 und beim spaltenorientierten Ansatz wie in Listing 6.13 aussehen.

```
1,Thomas,100000
2,Patrick,120000
3,Andreas,200000
```

Listing 6.12: Serialisierung zeilenorientierte Datenbank

```
1,2,3
Thomas,Patrick,Andreas
100000,120000,200000
```

Listing 6.13: Serialisierung spaltenorientierte Datenbank

Die Definitions- und Abfragesprache ist gleich wie bei relationalen Datenbanken, deshalb wird hier auf Beispiele verzichtet. Der Vorteil einer spaltenorientierten Datenbank liegt in der Performance bei *Aggregationsfunktionen*. Weiter führt dieser Datenbanktyp Veränderungen, welche alle Werte einer Spalte betreffen, effizient durch. Dafür ist das Einfügen einer neu Zeile oder das Auslesen einer ganze Zeile aufwendiger als bei dem zeilenorientierten Ansatz.

**Graph** In Graphdatenbanken sind die abgespeicherten Objekte die Knoten und die Beziehung der Objekte bilden die Kanten. Die Kanten können zusätzlich mit Eigenschaften versehen werden. Dieser Datenbanktyp kann für komplexe Netzwerke verwendet werden. Der Vorteil liegt bei der Handhabung der Beziehungen. Das Herausfinden der kürzesten Route zwischen zwei Elementen über ihre definierten Beziehungen stellt für diesen Datenbanktyp keine Herausforderung dar.

Die Daten werden je nach Implementation auch in Dokumenten abgespeichert, somit entfällt auch hier eine Definition des Schemas. In Listing 6.14 ist das Beispiel für das Abfragen der Strasse einer Person in Neo4j zu sehen.

```
1        MATCH (person:Person)
2        WHERE person.name = "Thomas"
3        RETURN person.address.street;
```

Listing 6.14: Abfrage in Neo4j

### 6.3.6 Vorselektierung

Um nicht alle sechs vorgestellten Datenbanksysteme in der Nutzwertanalyse miteinander vergleichen zu müssen, wurde eine Vorselektierung aufgrund der Ist-Analyse gemacht. Da objektrelationale und objektorientierte Datenbanksystem eher akademischer Natur sind und keine detaillierte Dokumentation darüber gefunden wurde, werden sie von der Nutzwertanalyse ausgeschlossen. Die Anforderungen lassen auf keinen Anwendungsfall für eine spaltenorientierte Datenbank schliessen. In den einzelnen Problemen spielen Beziehungen zwischen Objekten zwar oft eine Rolle, doch das würde eher in den Bereich des Algorithmus fallen, aus diesem Grund werden Graphdatenbanken ebenfalls ausgeschlossen. Somit bleiben noch die relationale Datenbank und die dokumentorientierten Datenbanken für die Nutzwertanalyse übrig.

Da es innerhalb der ausgewählten Datenbanksystemen sehr viele verschiedene Implementationen gibt, wurde von jedem Typ eine ausgewählt. Bei den relationalen Datenbanken fiel die Entscheidung auf MySQL, da diese Implementation kostenlos ist und bereits Erfahrung vorhanden war. Da MongoDB sehr verbreitet und eine gute Integration in die Java Frameworks hat, wurde diese Implementation unter den dokumentorientierten Datenbanken ausgewählt.

## 6.4 Nutzwertanalyse

### 6.4.1 Bewertungskriterien

In der Nutzwertanalyse werden folgende Punkte betrachtet und nach dem angegebenen Schema bewertet und dann gewichtet. Die Kriterien sind grösstenteils aus den *CAP-Theorem* abgeleitet.

#### Aufwand

- **Beschreibung**: Wie gross ist der geschätzte Aufwand mit diesem Datenbanktyp?
- **Bewertung**: 1: sehr hoch, 10: sehr niedrig
- **Gewichtung**: 5 (Die Zeit für dieses Projekt ist beschränkt und die Entscheidung könnte zu einem Risiko werden.)

#### Änderbarkeit

- **Beschreibung**: Wie flexibel ist diese Variante in Bezug auf Erweiterungen/Vererbung?
- **Bewertung**: 1: sehr spezifisch, 10: sehr flexibel
- **Gewichtung**: 5 (Die Anforderungen geben vor, dass das System einfach zu erweitern sein sollte.)

#### Konsistenz

- **Beschreibung**: Wie gut ist die Datenbank in Bezug auf Konsistenz?
- **Bewertung**: 1: gar nicht 10: sehr gut
- **Gewichtung**: 2 (Die Daten haben untereinander so gut wie keine Abhängigkeiten, es werden fast nur Einfüge-Operationen benützt.)

#### Verfügbarkeit

- **Beschreibung**: Wie hoch ist die Verfügbarkeit der Datenbank-Operationen?
- **Bewertung**: 1: niedrig, 10: sehr hoch
- **Gewichtung**: 4 (Die Schnittstelle wird für möglichst viele gleichzeitige Requests ausgelegt, eine Warteschlaufe von Operationen ist nicht gewünscht.)

#### Partitionstoleranz

- **Beschreibung**: Kann die Datenbank verteilt betrieben werden?
- **Bewertung**: 1: nur eingeschränkt, 10: sehr einfach
- **Gewichtung**: 4 (Mit zunehmender Grösse wird die Datenbank sehr wahrscheinlich verteilt betrieben.)

### 6.4.2 Bewertung

Anhand der zuvor definierten Kriterien wurde eine Bewertung vorgenommen. Diese Bewertung ist nicht generell gültig, sie bezieht sich nur auf dieses Projekt.

| **Kriterium** | **Relationale Datenbank** | **Dokumentorientierte Datenbank** |
|---|---|---|
| Aufwand | 7 (35) | 7 (35) |
| Begründung | MySQL ist bekannt und das Einrichten ist relativ schnell gemacht. | Dokumentorientierten Datenbank noch nie benutzt. Laut Informanten ziemlich simpel. |
| Änderbarkeit | 3 (15) | 10 (50) |
| Begründung | Jede Anpassung an den Entities Attributen erfordert auch eine Änderung des Schemas. | Sehr flexibel, bei Änderungen der Entities oder neuen Probleme keine Anpassung nötig. |
| Konsistenz | 10 (20) | 4 (8) |
| Begründung | Konsistenz ist einer der Hauptgründe für relationale Datenbanksysteme. | Die Stärke liegt nicht in der Konsistenz. |
| Verfügbarkeit | 6 (24) | 9 (36) |
| Begründung | Bei Transaktionen sind oft Teile der Datenbank gesperrt und nicht verfügbar. | Keine Sperrzeiten von Einträgen, somit keine Verzögerungen von Aktionen. |
| Partitionstoleranz | 7 (28) | 9 (36) |
| Begründung | Benötigt aufwendige Konfiguration um die Datenbank verteilt zu betreiben. | Kann gut und einfach verteilt betrieben werden. |
| **Total (gewichtet)** | **33 (122)** | **39 (165)** |

Tabelle 6.1: Nutzwertanalyse - Datenbank Varianten

### 6.4.3 Fazit

Das Resultat der Nutzwertanalyse hat ergeben, dass für dieses Projekt am besten eine dokumentorientierte Datenbank Lösung verwendet wird. Die Schnittstelle benötigt keine Transaktionen und profitiert von einer hohen Verfügbarkeit und Partitionstoleranz. Die Flexibilität von dokumentorientierten Datenbanksystemen ist ideal für dieses Projekt, da sie eine schnelle und unkomplizierte Erweiterung bzw. Anpassung ermöglicht. Zusätzlich werden auch die verschiedenen Ausprägungen der einzelnen Problemen gut unterstützt.

## 6.5 Datendiagramm des Datenspeichers

Da sich bei der Nutzerwertanalyse ergeben hat, dass eine dokumentorientierte Datenbank verwendet wird, entfällt ein Datendiagramm der Tabellen, wie es von relationalen Datenbanken bekannt ist. Die Collections benutzen kein vordefiniertes Schema, jedoch legen die verwendeten Klassen, welche in die Collections gespeichert werden, das Schema fest. Aus diesem Grund reicht ein Klassendiagramm, um zu wissen, wie die Daten abgespeichert werden.

In Abbildung 6.7 ist die Klassen-Hierarchie der Berechnungstypen dargestellt und in Abbildung 6.8 werden die verschiedenen Resultatklassen gezeigt.

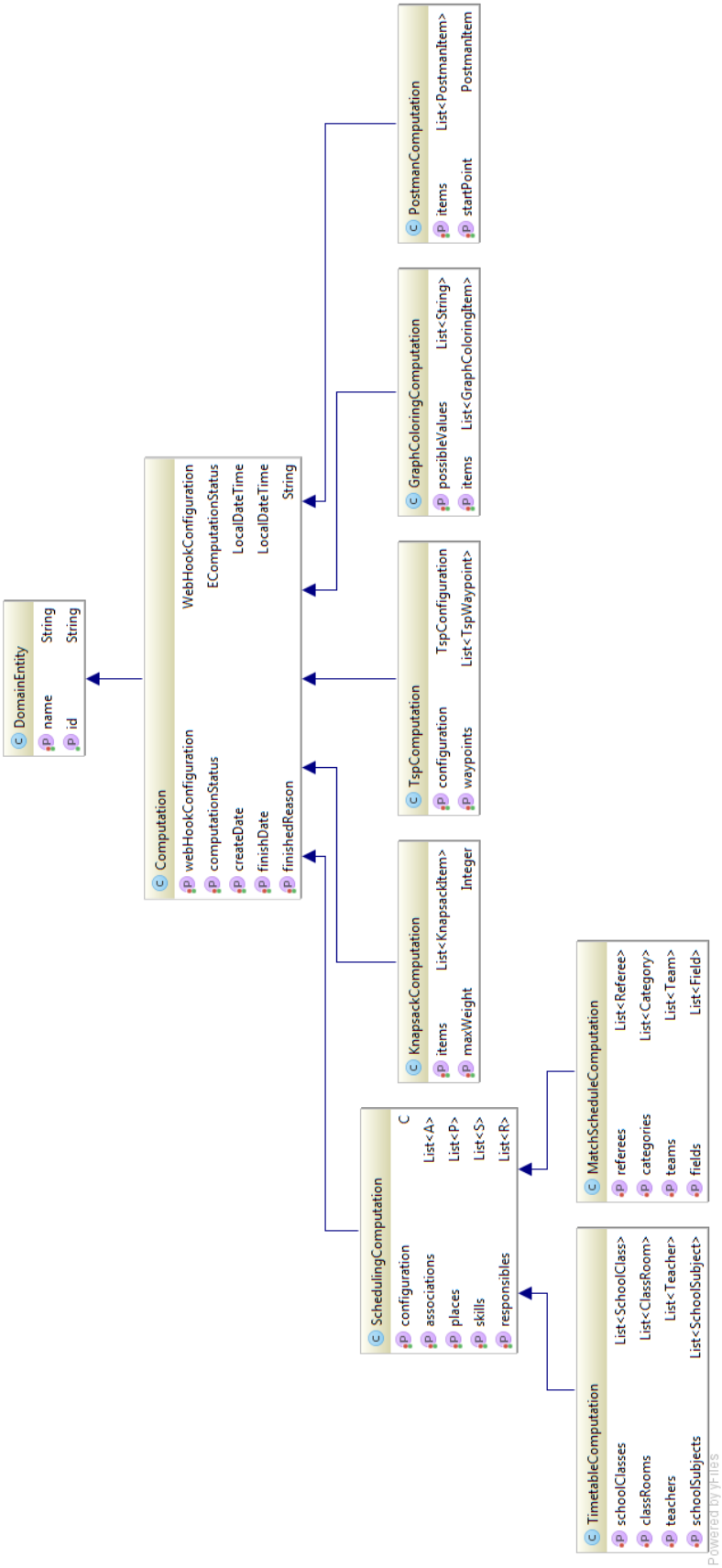



Abbildung 6.7: Klassendiagramm der verschiedenen Problemklassen (eigene Darstellung)

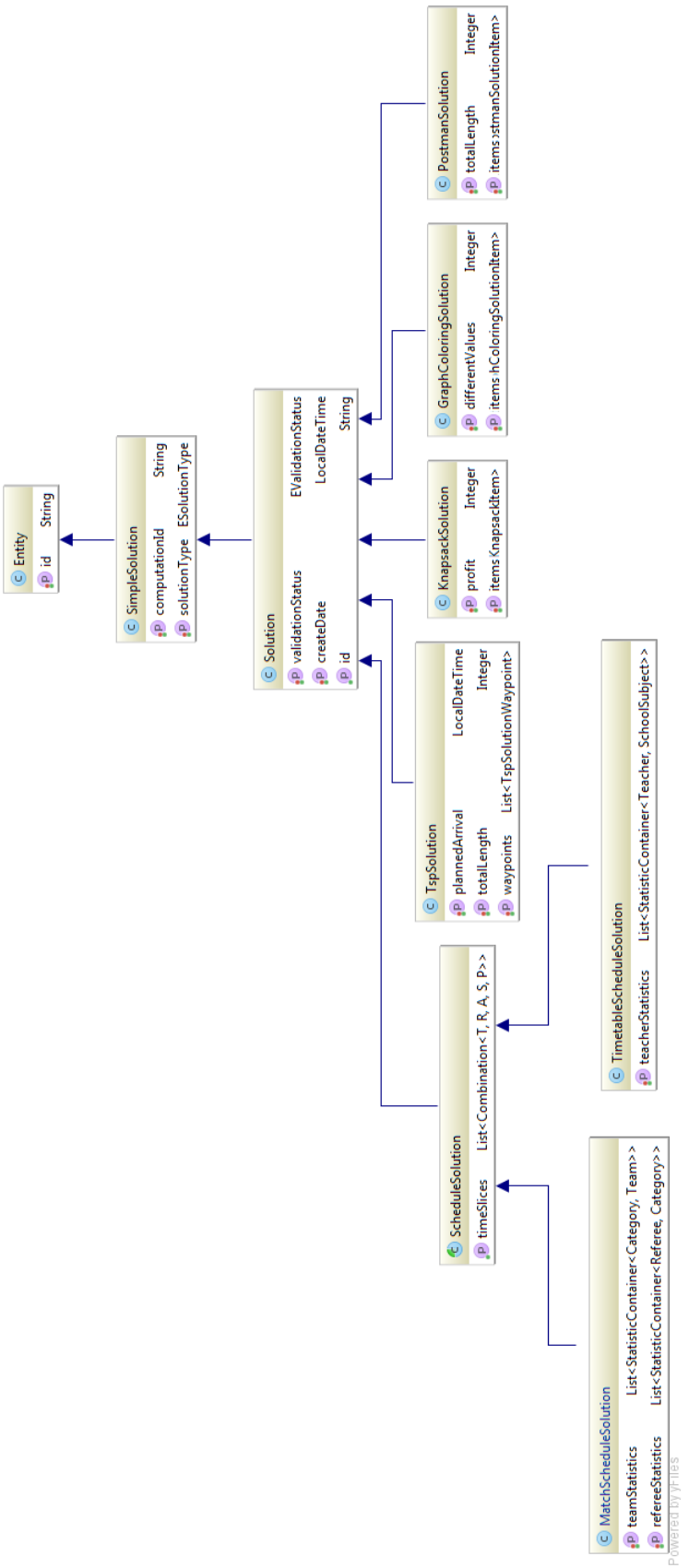



Abbildung 6.8: Klassendiagramm der verschiedenen Resultatklassen (eigene Darstellung)

# 7 Umsetzung des Prototyps [R5]

In diesem Kapitel wird kurz auf die Erkenntnisse aus dem ersten Durchstich eingegangen. Danach wird erklärt, wie die Umsetzung für die ausgewählten Problemtypen schliesslich durchgeführt wurde. Zu guter Letzt wird noch die Entwicklungsumgebung für dieses Projekt aufgezeigt.

## 7.1 Erster Durchstich

Zum Start der Umsetzung wurde ein erster vertikaler Durchstich anhand des Rucksack-Problems gemacht, um zu sehen, ob sich das Konzept bewährt. Während der Implementation wurde bereits überlegt, wie die Logik möglichst generisch gehalten werden kann.

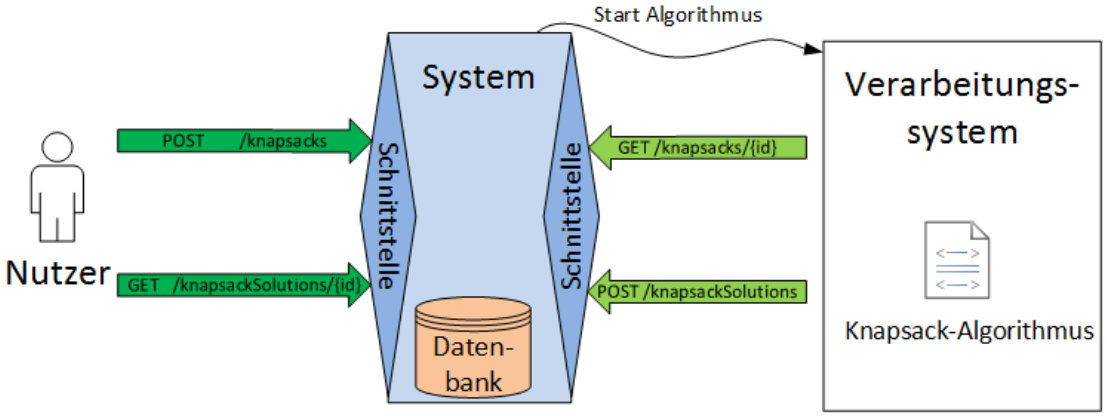



Abbildung 7.1: Vertikaler Durchstich mit dem Rucksack-Problem (eigene Darstellung)

### 7.1.1 Erkenntnisse nach dem ersten Durchstich

Die Kontaktschnittstelle nach Aussen generisch zu halten, macht aus Gründen der Benutzerfreundlichkeit keinen Sinn. Für den Kunden ist es transparenter, wenn er eine Schnittstelle aufruft, welche zum Beispiel 'timetabelScheduleComputations' heisst, als eine generische Schnittstelle, welche 'computations' heisst. Zudem ist nicht möglich, nur aus den Daten herauszufinden, was für ein Problem gelöst werden möchte. Dazu würde es einen zusätzlichen Parameter benötigen. Weiter stellt es sich in Java mit Spring als schwierig heraus, einen Endpunkt bereitzustellen, welcher verschiedene Ressourcentypen akzeptiert. Die Validierung der Eingabeparameter würde dadurch zusätzlich erschwert werden.

Das Konzept mit den beiden Translators bietet sehr viel Möglichkeiten und Flexibilität. Es entkoppelt die Nutzer-Schnittstelle komplett von der Schnittstelle für die Algorithmen. Um diese Flexibilität ausnutzen zu können, müssen jedoch oft verschiedene Entities für die Nutzer- und Algorithmus-Sicht erstellt werden.

In der Business Logik kann vieles generisch gehalten werden. Die Repositories, Services und die Interfaces können generisch programmiert werden und bleiben für alle Probleme gleich. Beim Controller kann die Logik in einer abstrakten Klasse definiert werden. Somit müssen nur noch die Namen der Schnittstellen in der spezifischen Implementation definiert werden. Es zeigte sich, dass sich das Konzept bewährt und die weiteren Probleme dementsprechend gelöst werden können.

### 7.1.2 Anpassung des Konzepts nach dem ersten Durchstich

Das Konzept wurde nach dem ersten Durchstich dahingegen abgeändert, dass die Schnittstellen auf der Nutzer-Seite umbenannt wurden und die Algorithmus-Seite einen anderen Namespace erhielt. Weiter verschwand die separate Schnittstelle für das Abfragen des Resultats, die Funktionalität wurde in die Status-Abfrage integriert.

## 7.2 Implementierung der Schnittstelle

In diesem Abschnitt wird über die Implementation der Schnittstelle im Allgemeinen geschrieben. Wie bereits erwähnt ist der Ablauf bei jedem Problem gleich, dementsprechend verhält sich die Schnittstelle im Allgemeinen gleich.

### 7.2.1 Statusabfrage

Damit der Nutzer möglichst wenig Schnittstellen ansprechen muss, erhält er bei einer Statusabfrage zugleich die vorhandenen Resultate. Der Nutzer erhält nur Resultate mit dem Status 'FINAL'. Resultate mit dem Status 'PARTIAL' werden zwar beim Status und der Zeit des zuletzt empfangenen Resultats berücksichtigt, aber nicht herausgegeben. Der Status besitzt neben den Resultaten, die ID und den Namen, weiter werden Start- und Endzeit angeben und zusätzlich noch die Zeit des zuletzt empfangenen Resultats. Falls eine Berechnung beendet ist, wird eine Begründung der Beendigung angezeigt. Dies kann zum Beispiel ein aufgetretener Fehler während des Starts oder der Berechnung sein.

```
{
  "id: "<Berechnung ID>",
  "name": "<Berechnungsname>",
  "computationStatus": "<Status der Berechnung>",
  "createDate": <Erstelldatum>,
  "finishDate": <Enddatum>,
  "finishedReason": "<Begruendung der Beendigung>",
  "lastResultReceived": <Datum des zuletzt empfangenen
     Resultats>,
  "solutions": <Resultate>
}
```

Listing 7.1: Aufbau einer Antwort auf eine Statusabfrage

### 7.2.2 WebHook Möglichkeit

Die Berechnungen können je nach Komplexität sehr lange dauern. Der Nutzer müsste immer wieder den Status abfragen, um zu sehen, ob ein Resultat vorhanden ist. *WebHooks* bieten Abhilfe für dieses Problem. Das Verfahren ist nicht standardisiert, ist aber sehr simpel und hilfreich. Beim Start einer Berechnung kann eine URL mitgegeben werden, zu welcher bei jeder Statusänderung ein POST-Request abgeschickt wird. Um diese Schnittstelle möglichst generisch zu halten, wurde eine Möglichkeit geschaffen, eine beliebige Payload für den POST-Request anzugeben. Die Nachricht der Statusänderung kann nach belieben mit dem Platzhalter '__MESSAGE__' irgendwo in die Payload eingebunden werden. Das Attribut 'message' wird in der Nachricht gesetzt, wenn der Platzhalter nirgends verwendet wird. Eine Konfiguration für die Chat Applikation 'Slack' würde wie in Listing 7.2 aussehen. Die Nachrichten bei den Statusänderungen sähen dann wie in Abbildung 7.2 aus.

```
  ...
  "webHookConfiguration": {
    "url": "https://hooks.slack.com/services/T0000/B0000/XXX",
    "payload": {
        "text": "__MESSAGE__",
```

```
6           "channel": "#simplatyser",
7           "username": "simplatyser",
8           "icon_emoji": ":squirrel:"
9       }
10    }
11  ...
```

Listing 7.2: Beispiel einer WebHook Konfiguration für Slack

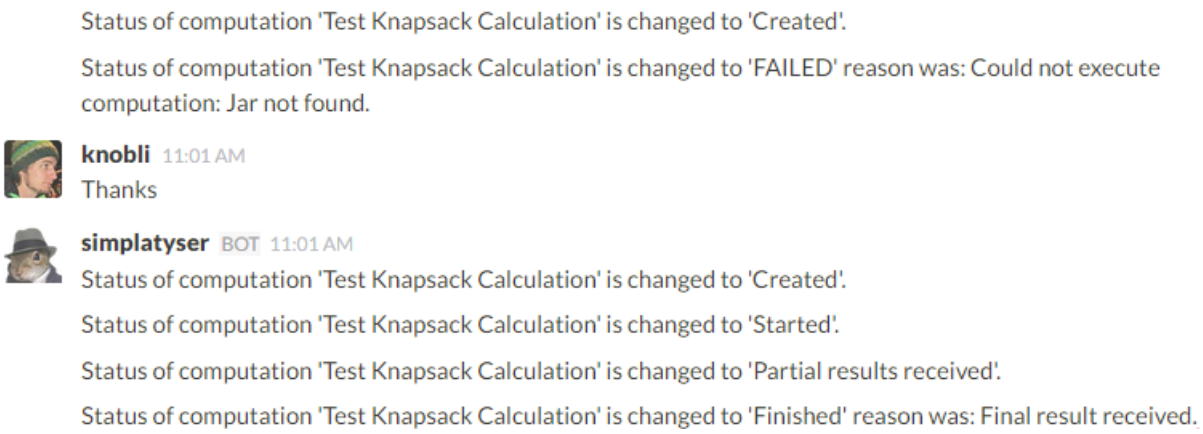



Abbildung 7.2: Nachrichten der Statusänderungen von Berechnungen (eigene Darstellung)

### 7.2.3 Technische Umsetzung und Probleme

Das Fundament der Software war mit Spring-Boot [spr15a] sehr schnell gebaut, die Anbindung an die MongoDB wurde mit SpringData [spr15b] realisiert. Leider waren oftmals die Dokumentationen nicht ausreichend oder nur für die XML-Konfiguration von Spring ausgelegt. Weiter gab es fehlende Teile in SpringData, welche gefunden werden mussten. So gibt es zum Beispiel zu diesem Zeitpunkt standardmässig keine Möglichkeit ein 'LocalTime'-Objekt oder ein 'LocalDateTime'-Objekt von Java 8 zu persistieren. Dies führte zu einem Stackoverflow anstatt einem Fehler, was wiederum die Suche nach dem Fehler erheblich erschwerte. Um diese Probleme zu beheben, musste ein eigener Converter geschrieben und dieser bei der MongoDB-Konfiguration angegeben werden.

Bei der Business Logik wurde geschaut, dass vieles mit *Generics* gelöst werden konnte, vor allem bei den Services konnte viel Code für alle Probleme verwendet werden. Auch bei den beiden Scheduling-Problemen, konnte viel Code-Duplizierung umgangen werden. Zusätzlich wurden häufig verwendete Funktionen, wie das Abbilden auf die Input Objekte, in Helfer-Klassen ausgelagert, damit sie nur ein Mal implementiert werden mussten.

## 7.3 Implementierung der Probleme

In diesem Unterkapitel wird für jedes Problem kurz erklärt, was beim Prototyp implementiert wurde, um die Möglichkeiten des Konzepts aufzuzeigen. Eine ausführliche Schnittstellen-Dokumentation ist in Abschnitt A.2 zu finden. Zusätzlich wurde noch eine elektronische Dokumentation der Schnittstelle mit Hilfe von Swagger UI erstellt. Diese stellt die angebotenen Schnittstellen, die Eingabeparameter und Resultat sehr übersichtlich dar.

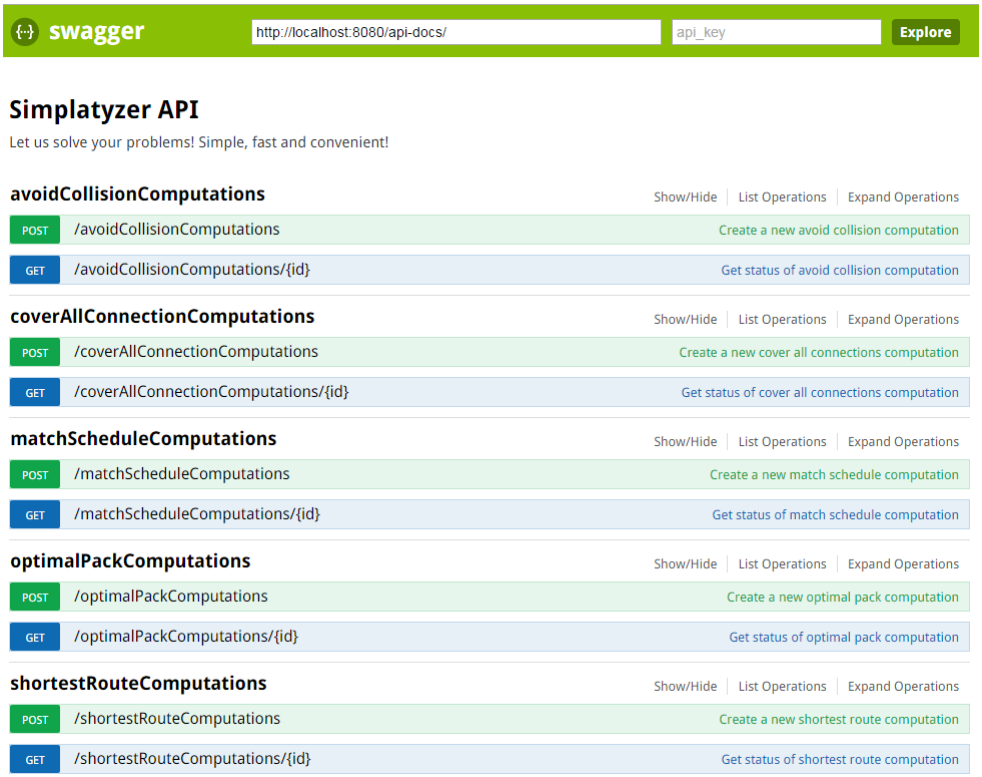



Abbildung 7.3: Nutzer-Schnittstellenbeschreibung von Swagger (eigene Darstellung)

Beim Evaluieren des Stundenplan-Problems wurde bemerkt, dass es sehr viele verschiedene Planungsprobleme gibt. Deshalb wurde ein zusätzliches Planungsproblem gewählt, um zu schauen, wie sich die Schnittstelle bei sehr ähnlichen Problemen verhält.

### 7.3.1 Rucksack

Das Rucksack-Problem ist aus Sicht der Schnittstelle ein relativ einfaches Problem. Aus Benutzerfreundlichkeit kann der Nutzer bei den Elementen eine Anzahl definieren und muss sie nicht doppelt angeben. Der Algorithmus hingegen bekommt eine Liste mit allen Elementen, es gibt nur noch einzelne Elemente und der Name wird nicht weiter gegeben, da er vom Algorithmus nicht benötigt wird. Der Algorithmus liefert als Resultat eine Liste von Boolean-Werten zurück, diese müssen zuerst wieder auf die ursprünglichen Elemente abgebildet werden. Bei der Validierung wird überprüft, ob die Gewichtsschranke nicht überschritten wurde und ob ein Element nicht zu oft verwendet wurde. Letzteres ist zwar bei dem Test-Algorithmus nicht möglich, könnte jedoch bei einer anderen Implementation der Fall sein. Der Benutzer erhält als Resultat eine Liste mit allen Objekten, welche verwendet wurden. Wenn das Objekt mehrmals verwendet wurde, wird die entsprechende Anzahl angegeben.

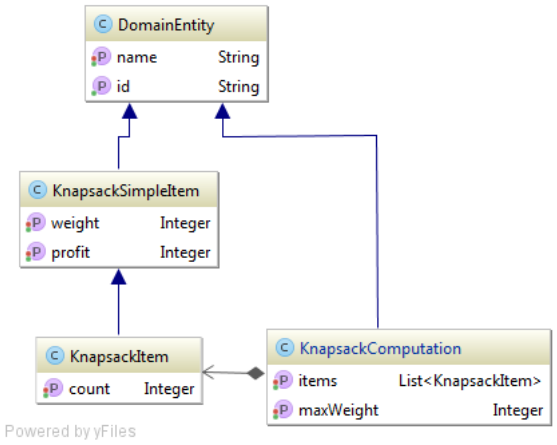



Abbildung 7.4: Eingabeparameter für eine Rucksack-Berechnung (eigene Darstellung)

### 7.3.2 Knotenfärbung

Bei der Knotenfärbung geht es darum, Kollisionen zu vermeiden. Dies wurde für den Nutzer möglichst transparent umgesetzt. Der Nutzer kann mögliche Werte (z.B. Farben oder Frequenzen) angeben, welche die Elemente zugeteilt bekommen sollen. Der Algorithmus erhält eine Liste mit allen Elementen und ihren Nachbarn. Der Name der Elemente und die möglichen Werte werden nicht weiter gegeben. Es wird davon ausgegangen, dass der Algorithmus eine Liste von Elementen mit den zugewiesenen Werten zurückgibt. Beim Übersetzen werden, falls vorhanden, die Werte vom Algorithmus mit den möglichen Werten aus der Benutzereingabe ausgetauscht. Bei der Validierung wird überprüft, ob kein Element den gleichen Wert wie einer seiner Nachbarn hat. Als Resultat erhält der Benutzer eine Liste mit allen Elementen und ihren zugewiesenen Werten.

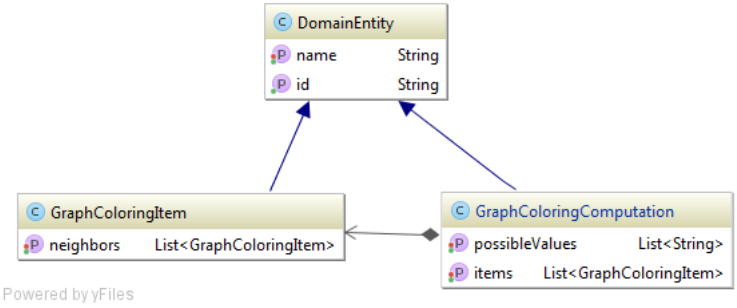



Abbildung 7.5: Eingabeparameter für eine Knotenfärbung-Berechnung (eigene Darstellung)

### 7.3.3 Problem des Handlungsreisenden

Der Service bietet nicht nur die kürzeste Route für eine Liste von Wegpunkten an, es ist auch möglich für jeden Wegpunkt eine gewünschte Ankunftszeit und Aufenthaltszeit anzugeben. Der Nutzer kann eine Liste von Wegpunkten mit gewünschter Ankunftszeit und Aufenthaltszeit angeben. Der Algorithmus erhält vom System eine Liste mit allen Wegpunkten, bei diesem Schritt wird nichts übersetzt. Es wäre möglich, dass die Schnittstelle bereits die Distanzen zwischen den Wegpunkten berechnet oder die Daten sonst irgendwie für den Algorithmus aufbereitet. Es wird davon ausgegangen, dass der Algorithmus eine Liste der Wegpunkte in der berechneten Reihenfolge zurückgibt. Beim Übersetzen werden die geplanten Ankunftszeiten berechnet, welche dann während der Validierung mittels der maximal angegebenen Abweichung überprüft werden. Der Benutzer erhält als

Resultat eine Liste mit der berechneten Reihenfolge der Wegpunkte. Jeder Wegpunkt besitzt die gewünschte und geplante Ankunftszeit.

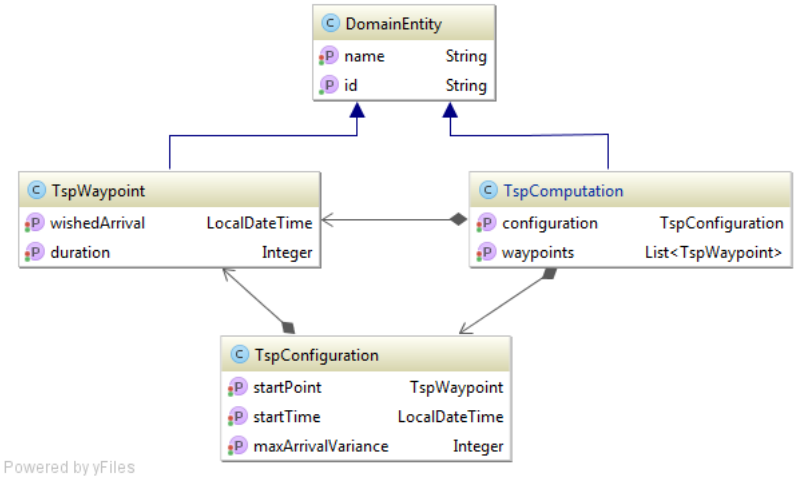



Abbildung 7.6: Eingabeparameter für eine Routen-Berechnung (eigene Darstellung)

### 7.3.4 Briefträgerproblem

Das Briefträgerproblem berechnet eine Route, welche alle bekannten Wege einer Strecke abfährt. Der Nutzer kann eine Liste von Wegpunkten mit den bekannten Verknüpfungen zu anderen Wegpunkten und ihrer Distanz angeben. Der Algorithmus erhält vom System eine Liste mit allen Wegpunkten, die Namen der Wegpunkte werden nicht weitergegeben. Es wird davon ausgegangen, dass der Algorithmus eine Liste der Wegpunkte in der berechneten Reihenfolge zurückgibt. Beim Übersetzen werden die Wegpunkte wieder auf die Eingabewerte abgebildet und die komplette Länge der Route berechnet. Die Validierung überprüft, ob der Weg möglich ist und ob jede Verbindung mindestens ein Mal benutzt wurde. Der Benutzer erhält als Resultat eine Liste mit der berechneten Reihenfolge der Wegpunkte und die totale Länge der Strecke.

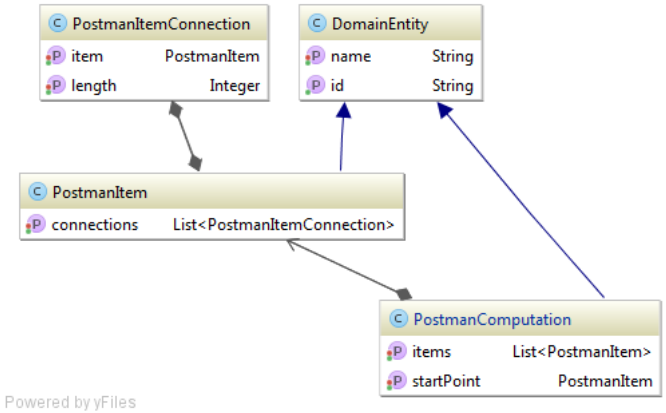



Abbildung 7.7: Eingabeparameter für eine Briefträger-Routen-Berechnung (eigene Darstellung)

### 7.3.5 Stundenplan Erstellung

Das Erstellen eines Stundenplans ist ein Planungsproblem, welches sehr vielen Möglichkeiten und Einschränkungen besitzt und somit sehr komplex ist. In dieser Implementation sind bei weitem

nicht alle Spezialitäten abgedeckt. Es wurde geschaut, dass bereits einige Einschränkungen, wie zum Beispiel freie Tage von Lehrern und blockierte Schulzimmer, miteinbezogen werden. Der Nutzer gibt eine Liste von Klassen, Lehrern, Schulzimmern und Schulfächern an. Die Klassen haben eine Grösse und definieren auch, welche Fächer sie besuchen müssen. Die Lehrer haben eine Liste mit Schulfächern, welche sie unterrichten können, eine List mit zugehörigen Klassen und eine Definition ihrer freien Tage. Die Klassenzimmer besitzen eine Liste mit möglichen Schulfächern und eine Sperrlist, in welcher definiert ist, wann der Raum nicht verfügbar ist. Die Schulfächer können definieren, ob sie eine Raum brauchen, welcher explizit dafür bestimmt ist, zum Beispiel Sport in der Turnhalle. Neben den Elementen, welche verplant werden, gibt es Randbedingungen wie die Pausenzeiten, die Lektionsdauer und die Definition, wann unterrichtet werden soll. Der Algorithmus erhält eine Liste mit allen Klassen, Lehrern, Schulzimmern und Schulfächern. Die Rahmenbedingungen werden in Zeitschlitze umgewandelt, welche vom Algorithmus verplant werden können. Zusätzlich werden die Elemente generischer benannt, damit der Algorithmus nicht verschiedene Konfigurationen für verschiedene Planungsprobleme besitzen muss. In Listing 7.3 ist ein Beispiel für eine Eingabe der Rahmenbedingungen gezeigt, welche durch den Translator in die 36 Zahl umgewandelt werden würde. Die Zahl wird aus der Lektionsdauer, den Pausen und den einzelnen Zeitfenstern pro Tag berechnet, die Tabelle 7.1 stellt dies visuell dar.

```
{
  ...
  "configuration": {
    "breakTimeSliceSize": [5, 20, 5, 90, 5, 20, 5],
    "dayTimeSlots": [
      {
        "monday": {"defaultTimes": true},
        "tuesday": {"defaultTimes": true},
        "wednesday": {"from": [8, 20, 0], "to": [12,0,0]},
        "thursday": {"defaultTimes": true},
        "friday": {"defaultTimes": true}
      }
    ],
    "lessonDuration": 45
  }
}
```

Listing 7.3: Ausschnitt einer Eingabe für das Stundenplanproblem für die Rahmenbedingungen

| Uhrzeit | Mo | Di | Mi | Do | Fr |
|---|---|---|---|---|---|
| 0820-0905 | 1 | 9 | 17 | 21 | 29 |
| 0910-0955 | 2 | 10 | 18 | 22 | 30 |
| 1015-1100 | 3 | 11 | 19 | 23 | 31 |
| 1105-1150 | 4 | 12 | 20 | 24 | 32 |
| 1320-1405 | 5 | 13 | - | 25 | 33 |
| 1410-1455 | 6 | 14 | - | 26 | 34 |
| 1515-1600 | 7 | 15 | - | 27 | 35 |
| 1605-1650 | 8 | 16 | - | 28 | 36 |

Tabelle 7.1: Visuelle Darstellung der Zeitfenster-Berechnung

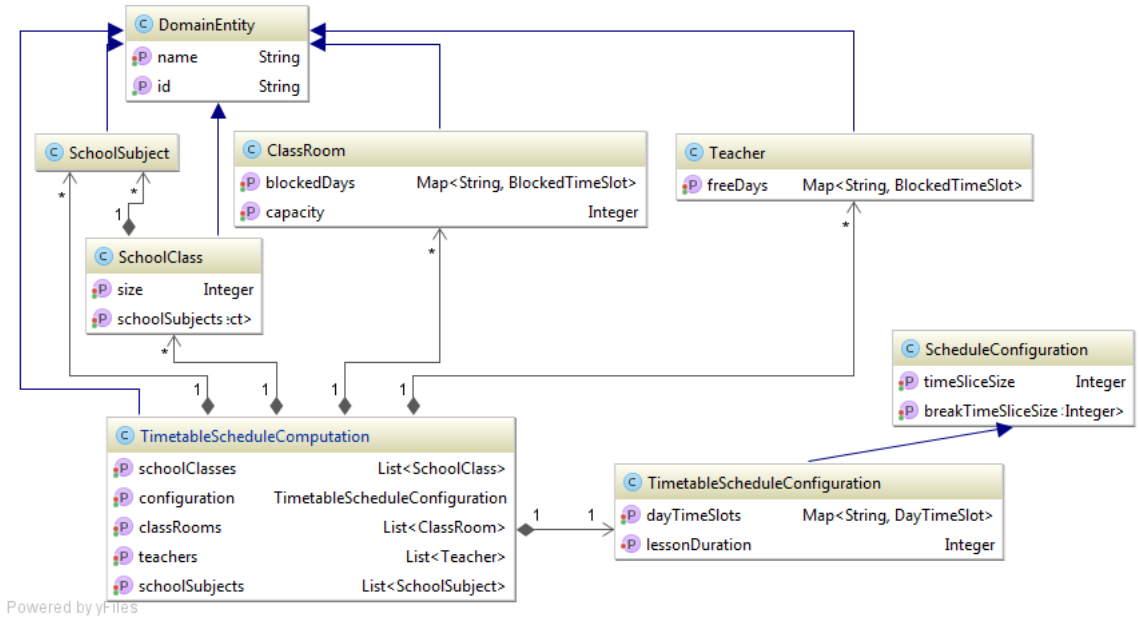



Abbildung 7.8: Eingabeparameter für eine Stundenplan-Berechnung (eigene Darstellung)

Es wird davon ausgegangen, dass der Algorithmus eine Liste von Planungskombinationen zurück gibt. Eine Kombination besteht aus einer Zeitfensternummer, einem Lehrer, einer Klasse, einem Schulfach und einem Klassenzimmer. Beim Übersetzen werden die IDs auf die ursprünglichen Elemente abgebildet, die Zeitfensternummern werden in Uhrzeiten umgewandelt. Zusätzlich wird eine Statistik für die Lehrer geführt. Bei der Validierung wird geschaut, ob ein Element zu einer Zeit mehrfach verplant ist, ob der Lehrer die nötigen Fähigkeiten hat und ob ein Klassenzimmer für das Fach ausgelegt ist. Das Resultat für den Nutzer enthält eine Liste mit den berechneten Kombination, sortiert nach Wochentag und Uhrzeit. Die Lehrerstatistik zeigt, wie oft ein Lehrer ein bestimmtes Fach unterrichtet und wie viel Stunden er insgesamt unterrichtet.

### 7.3.6 Spielplan Erstellung

Eine weitere Ausprägung des Planungsproblems ist das Erstellen eines Spielplans. Der Nutzer gibt eine Liste von Teams, Schiedsrichter, Spielfelder und Kategorien an. Die Teams haben eine bestimmte Kategorie. Die Schiedsrichter haben eine Liste mit Kategorien, welche sie leiten können und eine Liste von Teams, zu welchen sie gehören. Die Spielfelder besitzen eine Liste mit möglichen Kategorien. Neben den Elementen, welche verplant werden, gibt es Randbedingungen wie die Spieldauer, die Startzeit und die Pausenzeiten. Der Algorithmus erhält eine Liste mit allen Teams, Schiedsrichter, Spielfelder und Kategorien. Zusätzlich werden die Elemente generischer benannt, damit der Algorithmus nicht verschiedene Konfigurationen für verschiedene Planungsprobleme besitzen muss.

Es wird davon ausgegangen, dass der Algorithmus eine Liste von Planungskombinationen zurück gibt. Eine Kombination besteht aus einer Zeitfensternummer, einem Schiedsrichter, einer Kombination von zwei Teams und einem Spielfeld. Beim Übersetzen werden die IDs auf die ursprünglichen Elemente abgebildet, die Zeitfensternummern werden in Uhrzeiten umgewandelt. Zusätzlich wird eine Statistik für die Schiedsrichter und die Teams geführt, wie viele Einsätze sie jeweils haben. Bei der Validierung wird geschaut, ob ein Element zu einer bestimmten Zeit mehrfach verplant ist, ob die Schiedsrichter die nötigen Fähigkeiten haben und ob ein Spielfeld für die Kategorie ausgelegt ist. Der Benutzer erhält als Resultat eine Liste mit den berechneten Kombination, sortiert nach Uhrzeit. Die Schiedsrichterstatistik zeigt, wie oft ein Schiedsrichter eine bestimmte Kategorie pfeift und für wie viel Spiele er insgesamt verantwortlich ist. Die Teamstatistik zeigt, wie viele Spiele eine Kategorie insgesamt hat und wie viele Spiele ein Team bestreitet. In Listing 7.4 wird ein Ausschnitt

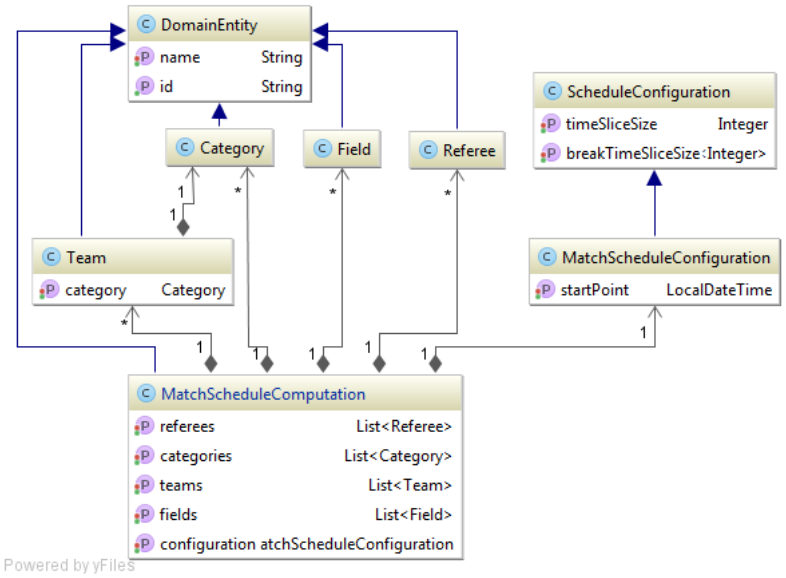



Abbildung 7.9: Eingabeparameter für eine Spielplan-Berechnung (eigene Darstellung)

der Schiedsrichterstatistik dargestellt.

```
{
  ...
  name: "Sabine Pfister"
  statisticMap: {
    Knaben: 1
    Damen Kat A: 3
    Damen Kat B: 2
    _TOTAL: 6
  }
  ...
}
```

Listing 7.4: Ausschnitt eines Resultats einer Spielplan Erstellung

### 7.3.7 Übersicht der Schnittstellen

Für den Nutzer wurde ein problem-agnostischer Name für die Schnittstelle gewählt. Die Namen unterscheiden sich somit zum Teil zwischen Nutzer- und Algorithmus-Sicht voneinander. Damit die Verknüpfung der beiden Schnittstellen nicht verloren geht und um eine Übersicht über alle angebotenen Schnittstellen zu haben, wurden sie in der Tabelle 7.2 zusammengetragen. Die Schnittstellen für die Algorithmen sind unter einem anderen Namespace '/algorithm', damit diese sauber voneinander getrennt sind.

| Nutzer | Algorithmus |
|---|---|
| / | /algorithm/ |
| **Allgemein** | |
| Berechnung starten | Berechnungsinformationen abholen |
| Status bzw. Lösung abholen | Lösung bekannt geben |
| **Rucksack** | |
| POST /optimalPackComputations | GET /knapsackComputations/{ID} |
| GET /optimalPackComputations/{ID} | POST /knapsackComputations/{ID}/solutions |
| **Knotenfärbung** | |
| POST /avoidCollisionComputations | GET /graphColoringComputations/{ID} |
| GET /avoidCollisionComputations/{ID} | POST /graphColoringComputations/{ID}/solutions |
| **Problem des Handlungsreisenden** | |
| POST /shortestRouteComputations | GET /tspComputations/{ID} |
| GET /shortestRouteComputations/{ID} | POST /tspComputations/{ID}/solutions |
| **Briefträgerproblem** | |
| POST /coverAllConnectionComputations | GET /postmanComputations/{ID} |
| GET /coverAllConnectionComputations/{ID} | POST /postmanComputations/{ID}/solutions |
| **Stundenplan Erstellung** | |
| POST /timetableComputations | GET /timetableComputations/{ID} |
| GET /timetableComputations/{ID} | POST /timetableComputations/{ID}/solutions |
| **Spielplan Erstellung** | |
| POST /matchesScheduleComputations | GET /matchesScheduleComputations/{ID} |
| GET /matchesScheduleComputations/{ID} | POST/matchesScheduleComputations/{ID}/solutions |

Tabelle 7.2: Übersicht der angebotenen Schnittstellen

### 7.3.8 Erstellung eines neuen Problems

Das Ziel dieser Arbeit war die Erstellung einer Schnittstelle, welche einfach zu erweitern ist. Dieser Abschnitt soll zeigen, wie dies erreicht wurde und wie sie erweitert werden kann. Das Software Projekt ist in Packages gegliedert. Auf der ersten Ebenen befinden sich die generischen Klassen. Die spezifischen Implementierung sind unter dem Package 'problem' angesiedelt. Die Gliederung ist so gewählt, damit ein Problem nur an einem Ort im Projekt verhängt ist und nicht an vielen verschiedenen Stellen gesucht werden muss. Es wäre möglich, die verschiedenen Problem-Implementationen in ein anderes Projekt auszulagern.

Um ein neues Problem in den Katalog aufzunehmen, müsste ein neues Package mit dem Problemnamen erstellt werden. Darin sind weitere Packages zur besseren Übersicht definiert. Als erstes müsste das Problem analysiert werden und dementsprechend die Entities für die Eingabe und das Resultat bereit gestellt werden.

Sind die Entities definiert, muss ein Controller für den Algorithmus und einer für den Nutzer erstellt werden. Die Controller leiten von einer abstrakten Klasse ab und dienen lediglich zur Definition der neuen Endpunkte. Die Dokumentation der Controller wird auch in die jeweiligen Klassen geschrieben, dies erweitert automatisch die elektronische Dokumentation von Swagger UI. Nun fehlen noch die beiden Translator für das Umwandeln der Nutzer-Sicht in die Algorithmus-Sicht, der Validator für das Problem und der Solver, welcher den Algorithmus startet. Alles andere ist generischer Code, welcher nur noch mit den jeweiligen Entity-Typen spezifiziert werden muss.

Zu guter Letzt wird noch ein Beispiel für eine Eingabe aus Nutzer-Sicht und ein Resultat aus Algorithmus-Sicht in JSON unter den Ressourcen abgelegt. Diese Beispiele können für Tests verwendet werden.

## 7.4 Entwicklungsumgebung

Ein Softwareprojekt benötigt immer eine gewisse Entwicklungsumgebung. Bei der Entwicklung mit Spring-Boot sind die Anforderungen minimal, da Spring-Boot bereits einen eigenen Webserver mitbringt.

### 7.4.1 IDE - Integrated Development Environment

Als *Integrated Development Environment* (IDE) wurde IntelliJ verwendet, IntelliJ bietet gute Refactoring-Methoden an und ist eine grosse Unterstützung beim Programmieren von Java-Code. Die IDE bietet auch die Möglichkeit Klassen-Diagramme zu erstellt und hat ein *VIM*-Plugin. Mit diesem Plugin können VIM-Befehle benutzt werden, was die Geschwindigkeit beim Programmieren enorm erhöht.

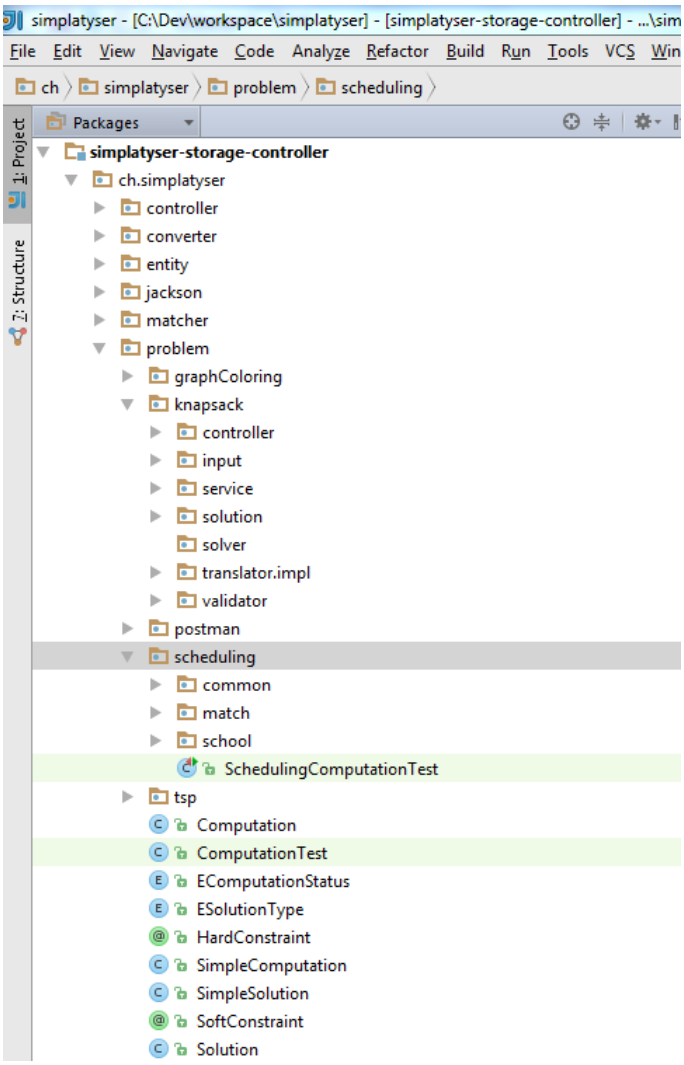



Abbildung 7.10: Darstellung der Package-Struktur in IntelliJ (eigene Darstellung)

### 7.4.2 Versionierung

Für die Versionierung der Software wurde git [git15a] verwendet. Das Remote Repository wurde auf Github [git15b] erstellt. Es wurde darauf geachtet, dass der Code oft ins Repository geladen wurde, damit ein Backup existiert. Die Dokumentation wurde in Dropbox gespeichert, damit auf verschiedenen Computern darauf zugegriffen werden konnte und immer ein Backup vorhanden war. Zu Korrekturzwecken wurde die Arbeit ebenfalls auf github hochgeladen und die Änderungsvorschläge wurden mit Latexdiff verglichen.

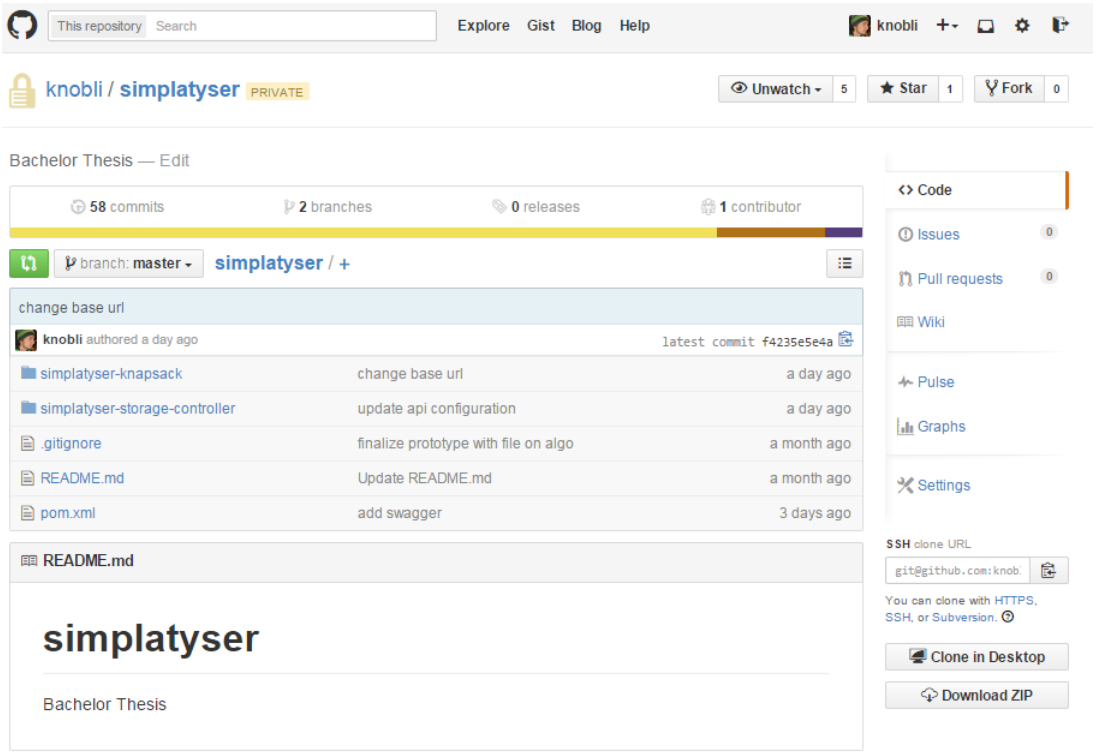



Abbildung 7.11: Github Repository des Simplatyser Projekts (eigene Darstellung)

### 7.4.3 Testen - Analysieren

Über den Chrome App 'Advanced REST client' (siehe Abbildung 7.12) wurde die Schnittstelle manuell getestet. Für die Regression-Tests wurde Junit und Mockito verwendet. Die statische Code Analyse wurde mit *Sonar* [son15] durchgeführt.

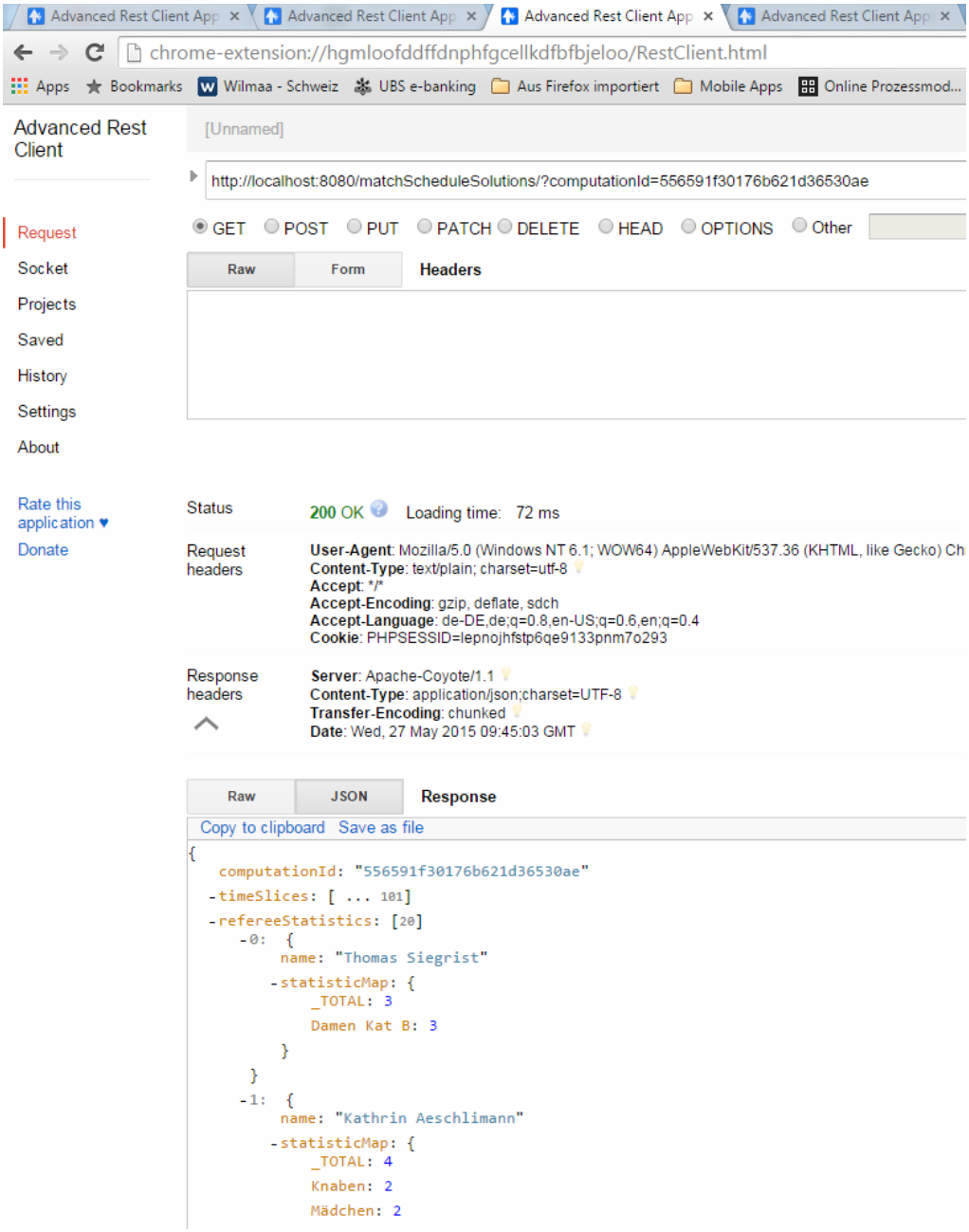



Abbildung 7.12: Advanced Rest client (eigene Darstellung)

# 8 Tests [R6]

In diesem Kapitel wird auf das Testverfahren und die Tests für dieses Projekts eingegangen.

## 8.1 Einführung

In diesem Projekt wurden die sieben Grundsätze aus [SL12] als Leitlinie zum Testen verwendet:

1. **Testen zeigt die Anwesenheit von Fehlern**: Die Testabdeckung wurde mit Hilfe von IntelliJ ermittelt und anhand der Anforderungen überprüft.
2. **Vollständiges Testen ist nicht möglich**: Es wurde solange getestet bis alle Akzeptanzkriterien abgedeckt waren und die Testabdeckung gut genug war.
3. **Mit dem Testen frühzeitig beginnen**: Die Unit Tests für neuen Funktionen wurden möglichst zeitnah oder zur gleichen Zeit geschrieben.
4. **Häufung von Fehlern**: Wenn Fehler auftraten, dann wurde diese Komponente nach der Behebung nochmals intensiv getestet und zusätzliche Tests erfasst.
5. **Zunehmende Testresistenz (Pesticide paradox)**: Jede Änderungen am Code hatte auch eine Anpassung oder Erweiterung der Tests zur Folge.
6. **Testen ist abhängig vom Umfeld**: Der Prototyp ist nicht sicherheitskritisch, jedoch wurde Wert auf eine hohe Testabdeckung gelegt. Es wurden zusätzlich immer wieder End-zu-End Tests mit den verschiedenen Problemen durchgeführt.
7. **Trugschluss: Keine Fehler bedeutet ein brauchbares System**: Die Schnittstelle wurde dem Stakeholder, bevor das Projekt zu Ende war, demonstriert und erklärt. So konnten allfällige Missverständnis frühzeitig erkannt werden. Die hohe Testabdeckung trägt ebenfalls zur Aufdeckung von unbemerkten Fehler bei.

## 8.2 Testing

Komponententests und Integrationstests, bekannt aus [SL12], wurden bei der Schnittstelle mit JUnit und Mockito durchgeführt. Bei den Integrationstests wurde eine Testdatenbank verwendet, welche bei jedem Testlauf neu initialisiert wird. Die Ausgangslage ist somit für jeden Durchlauf identisch. Für die Analyse der Testabdeckung wurde IntelliJ verwendet und die statische Code Analyse wurde mit Sonar überprüft. Neben den automatischen Tests wurden immer wieder manuelle Tests durchgeführt, um auch das Nutzererlebnis zu testen.

## 8.3 Systemtest

Als Abschluss des Projekts wurde ein Systemtest vorgesehen, für welcher ein Testprotokoll erstellt wurde. Das Protokoll zeigt dem Kunden was getestet wurden und das Ergebnis.

### 8.3.1 Testprotokoll

Das Testprotokoll basiert auf den Use Cases (siehe Abschnitt 5.2.1) und den Anforderungen (Abschnitt 5.2.2). Akzeptanz-Kriterien mit UND- oder ODER-Verknüpfung wurden aufgesplittet, um sicher zu gehen, dass beide Bedingungen erfüllt sind.

Tabelle 8.1: Testprotokoll

| **ID** | **Test** | **Herkunft** | **nicht / teilweise / erfüllt** |
|---|---|---|---|
| 1 | Der Nutzer kann eine Schnittstelle für die bereitgestellte Berechnungsfunktionen ansprechen. | Anforderung RF-F1 | erfüllt |
| 2 | Der Nutzer erhält nach dem Starten einer Berechnung eine ID. | Anforderung RF-F3 | erfüllt |
| 3 | Die Parameter sind persistiert. | Anforderung RF-F2 | erfüllt |
| 4a | Es gibt eine Fehlermeldung, falls bei der Speicherung etwas fehlschlägt | Anforderung RF-F2 | erfüllt |
| 4b | oder die Eingabeparameter nicht gültig sind. | Anforderung RF-F2 | erfüllt |
| 5a | Der Befehl für den Start wird versendet | Anforderung RF-F4 | erfüllt |
| 5b | und die ID dabei übergeben. | Anforderung RF-F4 | erfüllt |
| 6 | Die Fehlermeldung bei einem Fehlversuch wird gespeichert. | Anforderung RF-F4 | erfüllt |
| 7 | Das Verarbeitungssystem erhält die Eingabeparameter. | Anforderung RF-F5 | erfüllt |
| 8 | Das Verarbeitungssystem erhält eine Fehlermeldung, falls keine Eingabeparameter vorhanden sind. | Anforderung RF-F5 | erfüllt |
| 9 | Der Nutzer erhält einen Status seiner Berechnung. | Anforderung RF-F6 | erfüllt |
| 10 | Der Nutzer wird über die Änderungen des Status auf dem eingetragen Web-Hook informiert. | Anforderung RF-F7 | erfüllt |
| 11 | Das Verarbeitungssystem kann das Resultat abspeichern. | Anforderung RF-F8 | erfüllt |
| 12 | Das Verarbeitungssystem erhält eine Fehlermeldung, falls das Speichern fehlgeschlagen ist. | Anforderung RF-F8 | erfüllt |
| 13 | Der Nutzer erhält das Resultat der Berechnung. | Anforderung RF-F9 | erfüllt |
| 14 | Der Nutzer erhält eine entsprechende Fehlermeldung, wenn beim Bereitstellen des Resultats ein Fehler aufgetreten ist. | Anforderung RF-F9 | erfüllt |

*Fortsetzung auf der nächsten Seite*

**Tabelle 8.1 (Fortsetzung):** Testprotokoll

| **ID** | **Test** | **Herkunft** | **nicht / teilweise / erfüllt** |
|---|---|---|---|
| 15 | Das Interface kann verwendet werden, ohne dass das Verarbeitungssystem bekannt ist. | Qualitätsanforderung RF-NF1 | erfüllt |
| 16 | Unterschiedliche Ausprägungen eines Problems benutzen die gleiche API. | Qualitätsanforderung RF-NF2 | teilweise erfüllt |
| 17 | Unterschiedliche Probleme haben kein abweichendes Persistierungsschema. | Qualitätsanforderung RF-NF3 | erfüllt |

# 9 Schlussfolgerungen

In der Schlussfolgerung wird kurz auf die Verwendung des Produktes, das Fazit des Verfassers und den Ausblick eingegangen.

## 9.1 Verwendung

Während der Projektzeit konnte der Spielplan des Auffahrtsturniers des Turnvereins Grafstals validiert werden. Dieser Spielplan wird von Hand erstellt und immer wieder unterlaufen kleine Patzer, was bei einer Planung von etwa 110 Spielen und Schiedsrichtern, welche Coaches für bis zu drei Teams sind, nicht verwunderlich ist. Diese Aufgabe hat zum einen die Schnittstelle getestet, hat jedoch auch dem Verein geholfen, da auch dieses Jahr zwei kleine Fehler auftauchten. Der Turnverein hat sich bereits sehr für die vollautomatisierte Lösung interessiert.

## 9.2 Fazit

Das Erstellen eines Konzept einer solchen Schnittstelle stellte sich als sehr spannend heraus. Die Recherche und die Analyse der Probleme mit hoher Laufzeitkomplexität war sehr fesselnd und es bestand die Gefahr sich darin zu verlieren. Das Gebiet der theoretischen Informatik ist extrem weitreichend und die Probleme können beliebig komplex werden. Es war schnell klar, dass es schwierig sein wird, eine generische Lösung für die Probleme zu finden. Das Konzept mit dem pre- und post-Aktionen ist eine elegante Lösung und bietet sehr viel Flexibilität. Der Ablauf einer Berechnung konnte generisch gehalten werden und somit ist die Machbarkeitsstudie als erfolgreich zu betrachten.

Die Auswahl der Probleme hat sich als facettenreich herausgestellt und das Auswahlverfahren hat sich bewährt. Es ist spannend, dass die beiden Probleme aus dem Bereich ‘Netzwerk Design’ viele Ähnlichkeiten aufzeigen, jedoch mit komplett anderen Algorithmen gelöst werden. Die beiden Planungsprobleme sind sich sehr ähnlich und könnten ziemlich sicher mit einem generischen Algorithmus gelöst werden. Der grösste Unterschied liegt darin, dass bei einem Spielplan zwei Mannschaften gleichzeitig auf einem Platz sind und bei einem Stundenplan jeweils nur eine Klasse. Der Code der Schnittstelle konnte für die beiden Planungsprobleme auch viel generischer gehalten werden als für diejenigen aus dem Bereich ‘Netzwerk Design’.

Der Entscheid, eine dokumentorientierte Datenbank zu verwenden, war ideal. Die Flexibilität der Datenbank kam dem Projekt sehr zugute und es wurde keine Funktion vermisst. Spring Boot ist für einen Prototyp ideal, da es schon sehr viel mitbringt. Dies kann jedoch auch zu Problemen führen, da die Versionen von den Libraries nicht selber verwaltet werden können.

Ich bin mit dem Konzept und dem entstanden Prototypen sehr zufrieden und freue mich nun auf die Planung der nächsten Schritte. Mit Spannung erwarte ich die ersten realen Kunden, welche diese Schnittstelle verwenden und bin gespannt, wohin sich diese entwickeln wird. Ich bin mir bewusst, dass die Schnittstelle noch keine Produktionsreife hat. Es benötigt jedoch nicht mehr viel, bis die ersten Turnierpläne vollautomatisch erstellt werden können.

## 9.3 Ausblick

Das Konzept ist nun ausgearbeitet, jetzt wird eine automatisierte Lösung angestrebt. Als nächstes wird die Integration von zwei Teilprojekten, welche sich um die Entwicklung eines Verarbeitungssystem mit einer Cloud Lösung und um die Ansteuerung eines generischen Planungsalgorithmus handeln, geplant. Danach wird es weiteren Entwicklungsaufwand geben, da ein Webseite für die Eingabe für Kleinkunden angedacht ist. Da die Endlösung schliesslich dem Nutzer eine ungefähre

Berechnungszeit angeben möchte, wird eine Berechnungskomponente benötigt. Diese Komponente wird wahrscheinlich mittels Heuristiken und alten Resultaten eine Zeitextrapolation erstellen. Die Grundsteine für diese Berechnung sind mit dem vorgestellten Konzept bereits gelegt. Die Zwischenresultate und Endresultate enthalten einen Zeitstempel. Dadurch kann berechnet werden, wie lange die Berechnung zu einem gewissen Resultat gedauert hat. In Abbildung 9.1 ist das Konzept des übergeordneten Projekts dargestellt, von welchem Teile des User API und der Data Controller in diesem Projekt behandelt wurden.

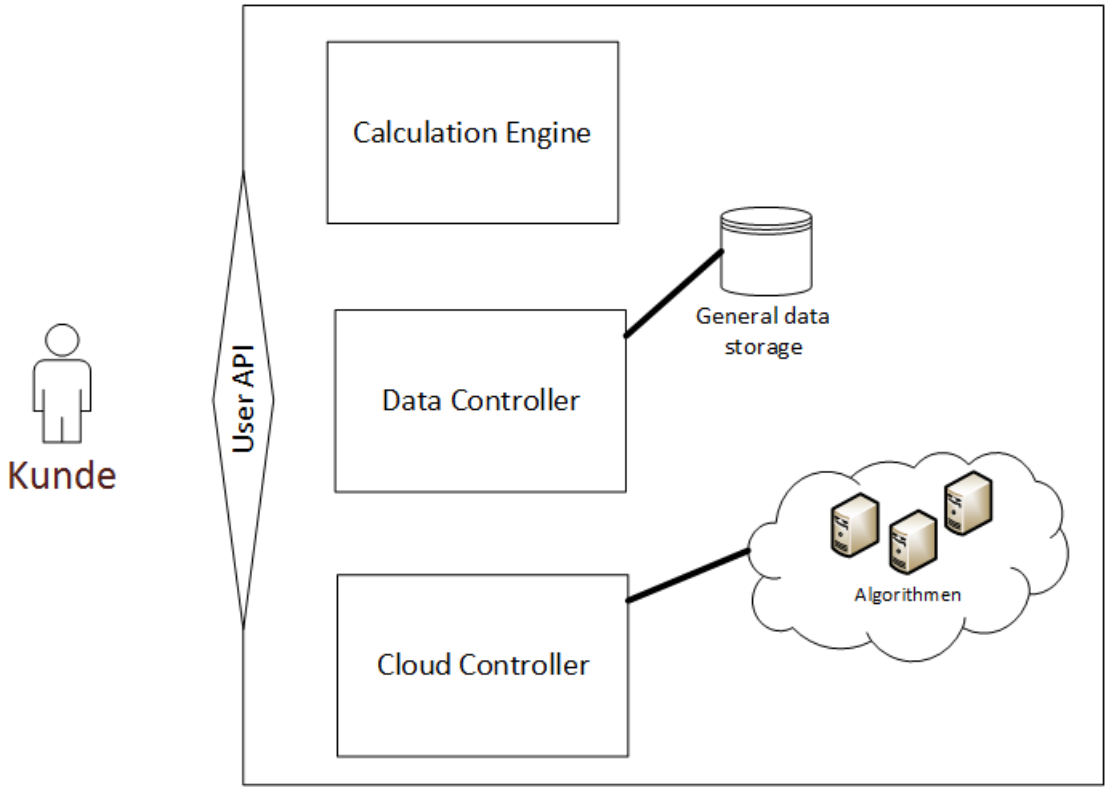



Abbildung 9.1: Konzept des übergeordneten Projekts (eigene Darstellung)

# A Anhang

## A.1 Risikoanalyse des Projekts

Jedes Projekt birgt Risiken. Werden diese nicht am Anfang analysiert und über die gesamte Projektlaufzeit überwacht, kann es zu grossen Schwierigkeiten kommen. Die angewandten Methoden sind aus [HHMS09] und aus dem dem Management Fach Projekt und Prozessmanagement bekannt.

### A.1.1 Risikoermittlung

Die Risikoermittlung wird zur Bestimmung und Folgenabschätzung möglicher Risiken angewendet.

| **Risiko** | **Mögliche Folgen** |
|---|---|
| Implementationsschwierigkeiten | • Zeitplan nicht einhaltbar<br>• Einige Anforderungen müssen zurückgestellt werden<br>• Projektarbeit kann nicht durchgeführt werden |
| Zeitengpässe | • Zeitplan nicht einhaltbar<br>• Einige Anforderungen müssen zurückgestellt werden<br>• Projektarbeit kann nicht durchgeführt werden |
| Mangelhaftes Endprodukt | • Produkt muss überarbeitet werden, da keine Abnahme durch den Stakeholder erfolgt |
| Anforderungen nicht vollständig | • Wichtige Funktionen stehen den Benutzern nicht zur Verfügung |

Tabelle A.1: Risikoermittlung

### A.1.2 Risikobewertung

Das Schadensausmass und die Eintrittswahrscheinlichkeit der Risiken sind nach folgendem Schema bewertet worden:

| Wert | Eintrittswahrscheinlichkeit | Schadensausmass |
|---|---|---|
| 1 | sehr unwahrscheinlich | vernachlässigbar |
| 2 | unwahrscheinlich | spürbar |
| 3 | wenig wahrscheinlich | verkraftbar |
| 4 | ziemlich wahrscheinlich | gefährlich |
| 5 | sehr wahrscheinlich | katastrophal |

Tabelle A.2: Risikobewertungsschema

Die Risiken aus der Risikobewertung (siehe A.1.1) wurden anhand dieses Schemas bewertet.

$$Risikofaktor = Eintrittswahrscheinlichkeit * Schadensausmass$$

| Risiko | Eintrittswahr-scheinlichkeit | Schadensausmass | Risikofaktor |
|---|---|---|---|
| Implementationsschwierigkeiten | 3 | 4 | **12** |
| Zeitengpässe | 2 | 5 | **10** |
| Mangelhaftes Endprodukt | 2 | 4 | **8** |
| Anforderungen nicht vollständig | 2 | 2 | 4 |

Tabelle A.3: Risikobewertung

### A.1.3 Risikomatrix

Anhand der Risikobeurteilung konnten die Risiken in eine Risikomatrix eingesetzt werden. Diese Matrix bietet einen guten Überblick über die Risiken und zeigt schnell, welche Risiken beachtet werden müssen.

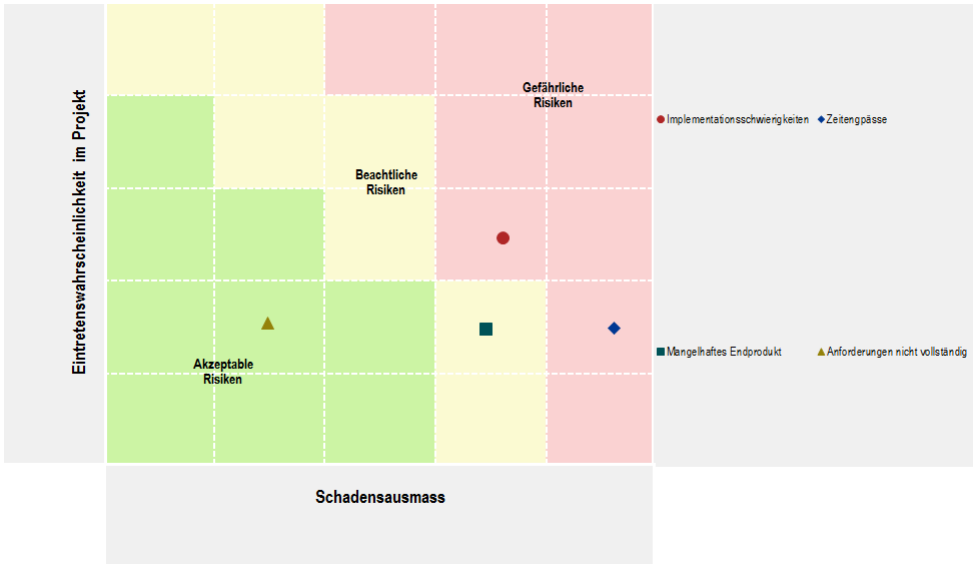



Abbildung A.1: Risikomatrix

### A.1.4 Massnahmen

Es wurden Massnahmen für die gefundenen Risiken gesucht und festgehalten. Die Massnahmen sind wiederum in vorbeugende Massnahmen und Eventualmassnahmen unterteilt.

| **Risiko** | **Massnahmen** | |
|---|---|---|
| | Vorbeugende Massnahmen | Eventualmassnahmen |
| Implementationsschwierigkeiten | • Im Zeitplan genügen Reserver einrechnen<br>• Kontaktpersonen zum Thema suchen<br>• Zeitplan einhalten, pünktlich mit den Arbeiten beginnen | • Betreuer / Schulleitung informieren<br>• Verschiebungsgesuch stellen |
| Zeitengpässe | • Fixe Zeiten einplanen<br>• Tätigkeiten priorisieren | • Vorgezogene Präsentation verschieben<br>• Verschiebungsgesuch stellen |
| Mangelhaftes Endprodukt | • Anforderungskatalog sauber erstellen<br>• Kunden laufend über den Stand des Produktes informieren | • Lösung mit dem Kunden suchen<br>• Projekt verlängern |
| Anforderungen nicht vollständig | • Review des Anforderungskataloges<br>• Kunden laufend über den Stand des Produktes informieren | • Anforderungen werden aufgenommen und in einen nächsten Release geplant |

Tabelle A.4: Risikoanalyse – Massnahmen

## A.2 Schnittstellen Dokumentation

Die Beispiele sind in JSON, es ist jedoch auch sehr einfach möglich die Schnittstelle für XML, YAML oder andere Formate zu erweitern

### A.2.1 Rucksack

**POST /optimalPackComputations** - Erstellung einer neuen Rucksack-Berechnung
Erstellt eine neue Berechnung des Rucksack-Problems und speichert sie in die Datenbank.
**Status Codes:**
200 - OK: Berechnung wurde erstellt, zusätzlich erhält der Nutzer noch die ID der Berechnung.
400 - Bad Request: Die Validierung der Eingabedaten ist fehlgeschlagen.

```
{
  "name": "<Berechnungsname>",
  "maxWeight": <Gewichtsschranke>,
  "items": [
    {
      "id": "<Objekt ID (optional)>",
      "name": "<Objekt Name>",
      "weight": <Gewicht>,
      "profit": <Profit>,
      "count": <Anzahl (optional)>
    }
  ]
}
```

Listing A.1: Beispiel einer Eingabe für das Rucksack-Problem

**GET /algorithm/knapsackComputations/{ID}** - Abrufen der Eingabedaten einer Rucksack-Berechnung
Ruft die Eingabedaten einer bestimmten Berechnung des Rucksack-Problems ab. Die Daten werden in der Schnittstelle zuerst für den Algorithmus umgewandelt.
**Status Codes:**
200 - OK: Die Eingabedaten wurden gefunden und zurückgeben.
404 - Not Found: Die Eingabedaten wurden nicht gefunden.

```
{
  "id": "<Berechnung ID>",
  "maxWeight": <Gewichtsschranke>,
  "items": [
    {
      "id": "<Objekt ID>",
      "weight": <Gewicht>,
      "profit": <Profit>
    }
  ]
}
```

Listing A.2: Beispiel für Eingabedaten des Rucksack-Problems für den Algorithmus

**POST /algorithm/knapsackComputations/{ID}/solutions** - Speichern einer Lösung für eine Rucksack-Berechnung
Speichert eine mögliche Lösung für eine Berechnung des Rucksack-Problems. Die Lösung wird zuerst für den Nutzer umgewandelt und danach noch validiert.

**Status Codes:**
200 - OK: Lösung wurde gespeichert.
400 - Bad Request: Die Validierung der Lösung zeigt Fehler auf, zusätzlich werden die Validierungsfehler mitgegeben.

```
{
  "solutionType": "<Resultat Typ (PARTIAL | FINAL)>",
  "fitness": <Profit des Resultates>,
  "items": [<Liste der Boolean-Werte>]
}
```

Listing A.3: Beispiel eines Resultates für das Rucksack-Problem aus Algorithmus-Sicht

**GET /optimalPackComputations/{ID}** - Abrufen des Status einer Rucksack-Berechnung
Ruft den Status und die Endresultate einer bestimmten Berechnung des Rucksack-Problems ab, die Daten werden direkt aus der Datenbank genommen, da sie dort in Nutzer-Sicht abgespeichert sind.
**Status Codes:**
200 - OK: Das Endresultat wurde gefunden und zurückgeben.
404 - Not Found: Keine Berechnung gefunden.

```
{
  "computationId": "<Berechnung ID>",
  "profit": <Profit des Resultates>,
  "items": [
    {
      "id": "<Objekt ID>",
      "name": "<Objekt Name>",
      "weight": <Gewicht>,
      "profit": <Profit>,
      "count": <Anzahl>
    }
  ]
}
```

Listing A.4: Beispiel eines Endresultates für das Rucksack-Problem

### A.2.2 Knotenfärbung

**POST /avoidCollisionComputations** - Erstellung einer neue Knotenfärbungsberechnung
Erstellt eine neue Berechnung des Knotenfärbungsproblems und speichert sie in die Datenbank.
**Status Codes:**
200 - OK: Berechnung wurde erstellt, zusätzlich erhält der Nutzer noch die ID der Berechnung.
400 - Bad Request: Die Validierung der Eingabedaten ist fehlgeschlagen.

```
{
  "name": "<Berechnungsname>",
  "items": [
    {
      "id": "<Element ID (optional)>",
      "name": "<Element Name>",
      "neighbors": [<Benachbarte Elemente>]
    }
  ],
```

```
  "possibleValues": [<Moegliche Werte>]
}
```

Listing A.5: Beispiel einer Eingabe für das Knotenfärbungsproblem

**GET /algorithm/graphColoringComputations/{ID}** - Abrufen der Eingabedaten einer Knotenfärbungsberechnung
Ruft die Eingabedaten einer bestimmten Berechnung des Knotenfärbungsproblems ab. Die Daten werden in der Schnittstelle zuerst für den Algorithmus umgewandelt.
**Status Codes:**
200 - OK: Die Eingabedaten wurden gefunden und zurückgeben.
404 - Not Found: Die Eingabedaten wurden nicht gefunden.

```
{
  "id": "<Berechnung ID>",
  "items": [
    {
      "id": "<Element ID>",
      "neighbors": [<Benachbarte Elemente>]
    }
  ]
}
```

Listing A.6: Beispiel für Eingabedaten des Knotenfärbungsproblems für den Algorithmus

**POST /algorithm/graphColoringComputations/{ID}/solutions** - Speichern einer Lösung für eine Knotenfärbungsberechnung
Speichert eine mögliche Lösung für eine Berechnung des Knotenfärbungsproblems. Die Lösung wird zuerst für den Nutzer umgewandelt und danach noch validiert.
**Status Codes:**
200 - OK: Lösung wurde gespeichert.
400 - Bad Request: Die Validierung der Lösung zeigt Fehler auf, zusätzlich werden die Validierungsfehler mitgegeben.

```
{
  "solutionType": "<Resultat Typ (PARTIAL | FINAL)>",
  "differentValues": <Anzahl verschiedene Werte>,
  "items": [
    {
      "id": "<Element name>",
      "value": "<Zugewiesener Wert>"
    }
  ]
}
```

Listing A.7: Beispiel eines Resultates für das Knotenfärbungsproblems aus Algorithmus-Sicht

**GET /graphColoringComputations/{ID}** - Abrufen des Status einer Knotenfärbungsberechnung
Ruft den Status und die Endresultate einer bestimmten Berechnung des Knotenfärbungsproblems ab, die Daten werden direkt aus der Datenbank genommen, da sie dort in Nutzer-Sicht abgespeichert sind.
**Status Codes:**
200 - OK: Das Endresultat wurde gefunden und zurückgeben.
404 - Not Found: Keine Berechnung gefunden.

```
{
  "computationId": "<Berechnung ID>",
  "differentValues": <Anzahl verschiedene Werte>,
  "items": [
    {
      "id": "<Element name>",
      "value": "<Zugewiesener Wert (ersetzt durch angegebene
         Werte)>"
    }
  ]
}
```

Listing A.8: Beispiel eines Endresultates für das Knotenfärbungsproblems

### A.2.3 Problem des Handlungsreisenden

**POST /shortestRouteComputations** - Erstellung einer neue Routenberechnung
Erstellt eine neue Berechnung des Problems des Handlungsreisenden und speichert sie in die Datenbank.
**Status Codes:**
200 - OK: Berechnung wurde erstellt, zusätzlich erhält der Nutzer noch die ID der Berechnung.
400 - Bad Request: Die Validierung der Eingabedaten ist fehlgeschlagen.

```
{
  "name": "<Berechnungsname>",
  "waypoints": [
    {
      "id": "<Wegpunkt ID (optional)>",
      "name": "<Wegpunkt Name>",
      "duration": <Aufenthaltszeit>,
      "wishedArrival": <Gewuenschte Ankunftszeit>
    }
  ],
  "configuration": {
    "startPoint": {
      "id": "<Wegpunkt ID (optional)>",
      "name": "<Wegpunkt Name>"
    },
    "startTime": <Startzeit>,
    "maxArrivalVariance": <Maximale Abweichung bei den
       Akunftszeiten>
  }
}
```

Listing A.9: Beispiel einer Eingabe für das Problem des Handlungsreisenden

**GET /algorithm/tspComputations/{ID}** - Abrufen der Eingabedaten einer Routenberechnung
Ruft die Eingabedaten einer bestimmten Berechnung des Problems des Handlungsreisenden ab. Die Daten werden in der Schnittstelle zuerst für den Algorithmus umgewandelt.
**Status Codes:**
200 - OK: Die Eingabedaten wurden gefunden und zurückgeben.
404 - Not Found: Die Eingabedaten wurden nicht gefunden.

```
{
  "id": "<Berechnung ID>",
  "waypoints": [
    {
      "id": "<Wegpunkt ID>",
      "name": "<Wegpunkt Name>",
      "duration": <Aufenthaltszeit>,
      "wishedArrival": <Gewuenschte Ankunftszeit>
    }
  ],
  "configuration": {
    "startPoint": {
      "id": "<Wegpunkt ID>",
      "name": "<Wegpunkt Name>"
    },
    "startTime": <Startzeit>,
    "maxArrivalVariance": <Maximale Abweichung bei den
       Akunftszeiten>
  }
}
```

Listing A.10: Beispiel für Eingabedaten des Problem des Handlungsreisenden für den Algorithmus

**POST /algorithm/tspComputations/{ID}/solutions** - Speichern einer Lösung für eine Routenberechnung
Speichert eine mögliche Lösung für eine Berechnung des Problems des Handlungsreisenden. Die Lösung wird zuerst für den Nutzer umgewandelt und danach noch validiert.
**Status Codes:**
200 - OK: Lösung wurde gespeichert.
400 - Bad Request: Die Validierung der Lösung zeigt Fehler auf, zusätzlich werden die Validierungsfehler mitgegeben.

```
{
  "solutionType": "<Resultat Typ (PARTIAL | FINAL)>",
  "totalLength": <Totale Laenge>,
  "waypoints": [
    {
      "name": "<Wegpunkt Name>"
    }
  ]
}
```

Listing A.11: Beispiel eines Resultates für das Problem des Handlungsreisenden aus Algorithmus-Sicht

**GET /shortestRouteComputations/{ID}** - Abrufen des Status einer Routenberechnung
Ruft den Status und die Endresultate einer bestimmten Berechnung des Problems des Handlungsreisenden ab, die Daten werden direkt aus der Datenbank genommen, da sie dort in Nutzer-Sicht abgespeichert sind.
**Status Codes:**
200 - OK: Das Endresultat wurde gefunden und zurückgeben.
404 - Not Found: Keine Berechnung gefunden.

```
{
  "computationId": "<Berechnung ID>",
  "totalLength": <Totale Laenge>,
  "plannedArrival": <Geplante Endankunftszeit>,
  "waypoints": [
    {
      "id": "<Wegpunkt ID>",
      "name": "<Wegpunkt Name>",
      "duration": <Aufenthaltszeit>,
      "wishedArrival": <Gewuenschte Ankunftszeit>,
      "plannedArrival": <Geplante Ankunftszeit>
    }
  ]
}
```

Listing A.12: Beispiel eines Endresultates für das Problem des Handlungsreisenden

### A.2.4 Briefträgerproblem

**POST /coverAllConnectionComputations** - Erstellung einer neue Routenberechnung
Erstellt eine neue Berechnung des Briefträgerproblems und speichert sie in die Datenbank.
**Status Codes:**
200 - OK: Berechnung wurde erstellt, zusätzlich erhält der Nutzer noch die ID der Berechnung.
400 - Bad Request: Die Validierung der Eingabedaten ist fehlgeschlagen.

```
{
  "name": "<Berechnungsname>",
  "items": [
    {
      "id": "<Wegpunkt ID (optional)>",
      "name": "<Wegpunkt Name>",
      "connections": [
        {
          "item": {<Wegpunkt>},
          "length": <Distanz>
        }
      ]
    }
  ],
  "startPoint": {
    "id": "<Wegpunkt ID (optional)>",
    "name": "<Wegpunkt Name>",
    "connections": [
      {
        "item": {<Wegpunkt>},
        "length": <Distanz>
      }
    ]
  }
}
```

Listing A.13: Beispiel einer Eingabe für das Briefträgerproblem

**GET /algorithm/postmanComputations/{ID}** - Abrufen der Eingabedaten einer Routenberechnung

Ruft die Eingabedaten einer bestimmten Berechnung des Briefträgerproblems ab. Die Daten werden in der Schnittstelle zuerst für den Algorithmus umgewandelt.
**Status Codes:**
200 - OK: Die Eingabedaten wurden gefunden und zurückgeben.
404 - Not Found: Die Eingabedaten wurden nicht gefunden.

```
{
  "id": "<Berechnung ID>",
  "items": [
    {
      "id": "<Wegpunkt ID>",
      "name": "<Wegpunkt Name>",
      "connections": [
        {
          "item": {<Wegpunkt>},
          "length": <Distanz>
        }
      ]
    }
  ],
  "startPoint": {
    "id": "<Wegpunkt ID (optional)>",
    "name": "<Wegpunkt Name>",
    "connections": [
      {
        "item": {<Wegpunkt>},
        "length": <Distanz>
      }
    ]
  }
}
```

Listing A.14: Beispiel für Eingabedaten des Briefträgerproblems für den Algorithmus

**POST /algorithm/postmanComputations/{ID}/solutions** - Speichern einer Lösung für eine Routenberechnung
Speichert eine mögliche Lösung für eine Berechnung des Briefträgerproblems. Die Lösung wird zuerst für den Nutzer umgewandelt und danach noch validiert.
**Status Codes:**
200 - OK: Lösung wurde gespeichert.
400 - Bad Request: Die Validierung der Lösung zeigt Fehler auf, zusätzlich werden die Validierungsfehler mitgegeben.

```
{
  "solutionType": "<Resultat Typ (PARTIAL | FINAL)>",
  "items": [
    {
      "id": "<Wegpunkt ID>"
    }
  ]
}
```

Listing A.15: Beispiel eines Resultates für das Briefträgerproblem aus Algorithmus-Sicht

**GET /coverAllConnectionComputations/{ID}** - Abrufen des Status einer Routenberechnung
Ruft den Status und die Endresultate einer bestimmten Berechnung des Briefträgerproblems ab, die Daten werden direkt aus der Datenbank genommen, da sie dort in Nutzer-Sicht abgespeichert sind.
**Status Codes:**
200 - OK: Das Endresultat wurde gefunden und zurückgeben.
404 - Not Found: Keine Berechnung gefunden.

```
{
  "computationId": "<Berechnung ID>",
  "items": [
    {
      "id": "<Wegpunkt ID>",
      "name": "<Wegpunkt Name>"
    }
  ],
  "totalLength": <Totale Laenge>
}
```

Listing A.16: Beispiel eines Endresultates für das Briefträgerproblem

### A.2.5 Stundenplan Erstellung

**POST /timetableComputations** - Erstellung einer neue Stundenplan-Berechnung
Erstellt eine neue Berechnung des Stundenplanproblems und speichert sie in die Datenbank.
**Status Codes:**
200 - OK: Berechnung wurde erstellt, zusätzlich erhält der Nutzer noch die ID der Berechnung.
400 - Bad Request: Die Validierung der Eingabedaten ist fehlgeschlagen.

```
{
  "name": "<Berechnungsname>",
  "schoolClasses": [
    {
      "id": "<ID Klasse>",
      "name": "<Klassename>",
      "schoolSubjects": [
        {
          "id": "<ID Schulfach>"
        }
      ],
      "size": <Groesse der Klasse>
    }
  ],
  "teachers": [
    {
      "id": "<ID Lehrer>",
      "name": "<Name des Lehrers>",
      "skills": [
        {
          "id": "<ID Schulfach>"
        }
      ],
      "associations": [
        {
          "id": "<ID Klasse (optional)>"
```

```
        }
      ],
      "freeDays": [
        {
          "<Wochentag>": {
            "from": <Startzeit (optional)>,
            "to": <Endzeit (optional)>,
            "morning": <Am Morgen (optional)>,
            "afternoon": <Am Nachmittag (optional)>,
            "wholeDay": <Ganzer Tag (optional)>
          }
        }
      ]
    }
  ],
  "classRooms": [
    {
      "id": "<ID Klassenzimmer>",
      "name": "<Klassenzimmername>",
      "allowedSkills": [
        {
          "id": "<ID Schulfach (optional)>"
        }
      ],
      "blockedDays": [
        {
          "<Wochentag>": {
            "from": <Startzeit (optional)>,
            "to": <Endzeit (optional)>,
            "morning": <Am Morgen (optional)>,
            "afternoon": <Am Nachmittag (optional)>,
            "wholeDay": <Ganzer Tag (optional)>
          }
        }
      ],
      "capacity": <Fassungsvermoegen>
    }
  ],
  "schoolSubjects": [
    {
      "id": "<ID Schulfach>",
      "name": "<Schulfachname>",
      "explicitLocation": <Explizites Schulzimmer (optional)>
    }
  ],
  "configuration": {
    "breakTimeSliceSize": [<Pausenzeiten>],
    "dayTimeSlots": [
      {
        "<Wochentag>": {
          "from": <Startzeit (optional)>,
          "to": <Endzeit (optional)>,
          "defaultTimes": <Standard Werte (optional)>
        }
      }
    ],
```

```
    "lessonDuration": <Lektionsdauer>
  }
}
```

Listing A.17: Beispiel einer Eingabe für das Stundenplanproblem

**GET /algorithm/timetableComputations/{ID}** - Abrufen der Eingabedaten einer Stundenplan-Berechnung
Ruft die Eingabedaten einer bestimmten Berechnung des Stundenplanproblems ab. Die Daten werden in der Schnittstelle zuerst für den Algorithmus umgewandelt.
**Status Codes:**
200 - OK: Die Eingabedaten wurden gefunden und zurückgeben.
404 - Not Found: Die Eingabedaten wurden nicht gefunden.

```
{
  "id": "<Berechnung ID>",
  "associations": [
    {
      "id": "<ID Klasse>",
      "name": "<Klassename>",
      "schoolSubjects": [
        {
          "id": "<ID Schulfach>"
        }
      ],
      "size": <Groesse der Klasse>
    }
  ],
  "responsibles": [
    {
      "id": "<ID Lehrer>",
      "name": "<Name des Lehrers>",
      "skills": [
        {
          "id": "<ID Schulfach>"
        }
      ],
      "associations": [
        {
          "id": "<ID Klasse>"
        }
      ],
      "freeDays": [
        {
          "<Wochentag>": {
            "from": <Startzeit>,
            "to": <Endzeit>,
            "morning": <Am Morgen>,
            "afternoon": <Am Nachmittag>,
            "wholeDay": <Ganzer Tag>
          }
        }
      ]
    }
  ],
```

```
  "places": [
    {
      "id": "<ID Klassenzimmer>",
      "name": "<Klassenzimmername>",
      "allowedSkills": [
        {
          "id": "<ID Schulfach>"
        }
      ],
      "blockedDays": [
        {
          "<Wochentag>": {
            "from": <Startzeit>,
            "to": <Endzeit>,
            "morning": <Am Morgen>,
            "afternoon": <Am Nachmittag>,
            "wholeDay": <Ganzer Tag>
          }
        }
      ],
      "capacity": <Fassungsvermoegen>
    }
  ],
  "skills": [
    {
      "id": "<ID Schulfach>",
      "name": "<Schulfachname>",
      "explicitLocation": <Explizites Schulzimmer>
    }
  ],
  "timeSlices": [
    {
      "number": <Zeitschlitz Nummer>
    }
  ]
}
```

Listing A.18: Beispiel für Eingabedaten des Stundenplanproblems für den Algorithmus

**POST /algorithm/timetableComputations/{ID}/solutions** - Speichern einer Lösung für eine Stundenplan-Berechnung
Speichert eine mögliche Lösung für eine Berechnung des Stundenplanproblems. Die Lösung wird zuerst für den Nutzer umgewandelt und danach noch validiert.
**Status Codes:**
200 - OK: Lösung wurde gespeichert.
400 - Bad Request: Die Validierung der Lösung zeigt Fehler auf, zusätzlich werden die Validierungsfehler mitgegeben.

```
{
  "solutionType": "<Resultat Typ (PARTIAL | FINAL)>",
  "timeSlices": [
    {
      "timeSlice": {
        "number": <Zeitschlitz Nummer>
      },
```

```
      "responsible": {
        "id": "<ID Lehrer>"
      },
      "association": {
        "id": "<ID Schulklasse>"
      },
      "skill": {
        "id": "<ID Schulfach>"
      },
      "place": {
        "id": "<ID Klassenzimmer>"
      }
    }
  ]
}
```

Listing A.19: Beispiel eines Resultates für das Stundenplanproblem aus Algorithmus-Sicht

**GET /timetableScheduleComputations/{ID}** - Abrufen des Status einer Stundenplan-Berechnung
Ruft den Status und die Endresultate einer bestimmten Berechnung des Stundenplanproblems ab, die Daten werden direkt aus der Datenbank genommen, da sie dort in Nutzer-Sicht abgespeichert sind.
**Status Codes:**
200 - OK: Das Endresultat wurde gefunden und zurückgeben.
404 - Not Found: Keine Berechnung gefunden.

```
{
  "computationId": "<Berechnung ID>",
  "plan": [
    {
      "<Wochentag>": [
        {
          "timeSlice": {
            "from": <Startzeit>,
            "to": <Endzeit>
          },
          "teacher": {
            "id": "<ID Lehrer>",
            "name": "<Name des Lehrers>",
          },
          "schoolClass": {
            "id": "<ID Klasse>",
            "name": "<Klassenname>",
          },
          "schoolSubject": {
            "id": "<ID Schulfach>",
            "name": "<Schulfachname>",
          },
          "classRoom": {
            "id": "<ID Klassenzimmer>",
            "name": "<Klassenzimmername>",
          }
        }
      ]
    }
```

```
  ],
  "teacherStatistics": [
    {
      "name": "<Lehrername>",
      "statisticMap": [
        {
          "<Schulfach>": <Anzahl Stunden>
        }
      ]
    }
  ]
}
```

Listing A.20: Beispiel eines Endresultates für das Stundenplanproblem

### A.2.6 Spielplan Erstellung

**POST /matchScheduleComputations** - Erstellung einer neue Spielplan-Berechnung
Erstellt eine neue Berechnung des Spielplanproblems und speichert sie in die Datenbank.
**Status Codes:**
200 - OK: Berechnung wurde erstellt, zusätzlich erhält der Nutzer noch die ID der Berechnung.
400 - Bad Request: Die Validierung der Eingabedaten ist fehlgeschlagen.

```
{
  "name": "<Berechnungsname>",
  "teams": [
    {
      "id": "<ID Team>",
      "name": "<Teamname>",
      "category": {
          "id": "<ID Kategorie>"
      }
    }
  ],
  "referees": [
    {
      "id": "<ID Schiedsrichter>",
      "name": "<Name des Schiedsrichters>",
      "skills": [
        {
          "id": "<ID Kategorie>"
        }
      ],
      "associations": [
        {
          "id": "<ID Team (optional)>"
        }
      ]
    }
  ],
  "fields": [
    {
      "id": "<ID Spielfeld>",
      "name": "<Spielfeldname>",
      "allowedSkills": [
```

```
        {
          "id": "<ID Kategorie (optional)>"
        }
      ]
    }
  ],
  "categories": [
    {
      "id": "<ID Kategorie>",
      "name": "<Kategoriename>"
    }
  ],
  "configuration": {
    "breakTimeSliceSize": [<Pausenzeiten>],
    "timeSliceSize": <Spieldauer>
    "startPoint": <Startpunkt>
  }
}
```

Listing A.21: Beispiel einer Eingabe für das Spielplanproblem

**GET /algorithm/matchScheduleComputations/{ID}** - Abrufen der Eingabedaten einer Spielplan-Berechnung
Ruft die Eingabedaten einer bestimmten Berechnung des Spielplanproblems ab. Die Daten werden in der Schnittstelle zuerst für den Algorithmus umgewandelt.
**Status Codes:**
200 - OK: Die Eingabedaten wurden gefunden und zurückgeben.
404 - Not Found: Die Eingabedaten wurden nicht gefunden.

```
{
  "id": "<Berechnung ID>",
  "associations": [
    {
      "id": "<ID Team>",
      "name": "<Teamname>",
      "category": {
          "id": "<ID Kategorie>"
      }
    }
  ],
  "responsibles": [
    {
      "id": "<ID Schiedsrichter>",
      "name": "<Name des Schiedsrichters>",
      "skills": [
        {
          "id": "<ID Kategorie>"
        }
      ],
      "associations": [
        {
          "id": "<ID Team (optional)>"
        }
      ]
    }
```

```
  ],
  "places": [
    {
      "id": "<ID Spielfeld>",
      "name": "<Spielfeldname>",
      "allowedSkills": [
        {
          "id": "<ID Kategorie (optional)>"
        }
      ]
    }
  ],
  "skills": [
    {
      "id": "<ID Kategorie>",
      "name": "<Kategoriename>"
    }
  ]
}
```

Listing A.22: Beispiel für Eingabedaten des Spielplanproblems für den Algorithmus

**POST /algorithm/matchScheduleComputations/{ID}/solutions** - Speichern einer Lösung für eine Spielplan-Berechnung
Speichert eine mögliche Lösung für eine Berechnung des Spielplanproblems. Die Lösung wird zuerst für den Nutzer umgewandelt und danach noch validiert.
**Status Codes:**
200 - OK: Lösung wurde gespeichert.
400 - Bad Request: Die Validierung der Lösung zeigt Fehler auf, zusätzlich werden die Validierungsfehler mitgegeben.

```
{
  "solutionType": "<Resultat Typ (PARTIAL | FINAL)>",
  "timeSlices": [
    {
      "timeSlice": {
        "number": <Zeitschlitz Nummer>
      },
      "responsible": {
        "id": "<ID Schiedsrichter>"
      },
      "association": {
        "team1": {
          "id": "<ID Team>"
        },
        "team2": {
          "id": "<ID Team>"
        }
      },
      "place": {
        "id": "<ID Spielfeld>"
      }
    }
  ]
}
```

Listing A.23: Beispiel eines Resultates für das Spielplanproblem aus Algorithmus-Sicht

**GET /matchScheduleComputations/{ID}** - Abrufen des Status einer Spielplan-Berechnung
Ruft den Status und die Endresultate einer bestimmten Berechnung des Spielplanproblems ab, die Daten werden direkt aus der Datenbank genommen, da sie dort in Nutzer-Sicht abgespeichert sind.
**Status Codes:**
200 - OK: Das Endresultat wurde gefunden und zurückgeben.
404 - Not Found: Keine Berechnung gefunden.

```
{
  "computationId": "<Berechnung ID>",
  "timeSlices": [
    {
      "timeSlice": {
        "from": <Startzeit>,
        "to": <Endzeit>
      },
      "referee": {
        "id": "<ID Schiedsrichter>",
        "name": "<Name des Schiedsrichters>",
      },
      "teams": {
        "team1": {
          "id": "<ID Team>",
          "name": "<Teamname>",
        },
        "team2": {
          "id": "<ID Team>",
          "name": "<Teamname>",
        }
      },
      "categorie": {
        "id": "<ID Kategorie>",
        "name": "<Kategoriename>",
      },
      "field": {
        "id": "<ID Klassenzimmer>",
        "name": "<Klassenzimmername>",
      }
    }
  ],
  "refereeStatistics": [
    {
      "name": "<Schiedsrichtername>",
      "statisticMap": [
        {
          "<Kategoriename>": <Anzahl Spiele>
        }
      ]
    }
  ],
  "teamStatistics": [
    {
      "name": "<Kategoriename>",
```

```
46        "statisticMap": [
47          {
48            "<Teamname>": <Anzahl Spiele>
49          }
50        ]
51      }
52    ]
53  }
```

Listing A.24: Beispiel eines Endresultates für das Spielplanproblem

# Acronyms

**ACID** Atomicity, Consistency, Isolation, Durability

**API** Application Programming Interface

**BASE** Basically Available, Soft state, Eventual consistency

**BLOB** Binary Large Object

**IDE** Integrated Development Environment

**JSON** JavaScript Object Notation

**ODL** Object Definition Language

**OODBMS** objektorientierte Datenbank

**OQL** Object Query Language

**ORDBMS** *objektrelationale Datenbank*

**RDBMS** relationale Datenbank

**REST** Representational State Transfer

**SQL** Structured Query Language

**XML** Extensible Markup Language

# Glossary

**Aggregationsfunktion** Liefern ein einzelnes Resultat einer Spalte, zum Beispiel AVG - Mittelwert oder SUM - Summe.

**Application Programming Interface** Application Programming Interace ist eine Schnittstelle, über welche anderen Applikationen Leistungen beziehen können.

**Atomicity, Consistency, Isolation, Durability** Häufig erwünschte Eigenschaften von Datenbanken, welche bei relationalen Datenbanken durch Transaktionen realisiert werden.

**Aussagenlogik** Ein Teilgebiet der Logik, in welchem es um Aussagen und deren Verknüpfung geht. Für eine Aussage der Aussagenlogik kann genau bestimmt werden, ob sie wahr oder falsch ist.

**Basically Available, Soft state, Eventual consistency** Eine etwas entschärfte Variante von ACID, bei welcher es mehr um die Verfügbarkeit und die Schnelligkeit geht, was mit einer weichen Konsistenz erreicht wird.

**Basisfaktor** Basisfaktoren (unterbewusste Anforderungen) muss das System in jedem Fall vollständig erfüllen, sonst stellt sich beim Stakeholder massive Unzufriedenheit ein. [PR11]

**Binary Large Object** Datentyp von Datenbanken für grosse, nicht weiter strukturierte, binäre Objekte (z.B. Bilder oder PDF-Dateien).

**CAP-Theorem** Besagt, dass ein verteiltes System unmöglich alle drei Werte, Konsistenz (C), Verfügbarkeit (A) und Partitionstoleranz (P), garantieren kann.

**deterministisch** Einen Algorithmus wird deterministisch genannt, wenn zu jedem Zeitpunkt der Folgeschritt eindeutig bestimmt ist. [Wal15]

**Domänensprache** Sprache für Begriffe in einer spezifische Umgebung bzw. einer gewissen Nutzergruppe.

**Eulerkreis** Ein ungerichteter Graph, bei welchen kein Knoten eine ungerade Anzahl anliegender Kanten hat.

**Extensible Markup Language** Markup Sprache zum Austausch strukturierter Daten.

**Generics** Generische Programmierung in Java wird durch sog. Generics seit Java 1.5 ermöglicht. Der Begriff steht synonym für „parametrisierte Typen. [wik15a]

**HTTP-Statuscode** Bei einem HTTP-Request teilt der HTTP-Statuscode Informationen über den Erfolg der Anfrage mit. Die erste Zahl ist die Statusklasse, die bekanntesten Codes sind 200 OK oder 404 Not Found. Die Codes sind in [FR14] definiert.

**Integrated Development Environment** Eine integrierte Entwicklerumgebung (englisch Integrated Development Environment) beinhalten einen Texteditor, Compiler (falls benötigt), Debugger, Formatierungsfunktionen und zum Teil die Möglichkeit zur Erstellung von grafischen Benutzeroberflächen.

**JavaScript Object Notation** Format zum Austausch von strukturierterten Daten, bekannt für seine gute Lesbarkeit und wenig Overhead.

**Object Definition Language** Definitionssprache, welche Ähnlichkeiten mit der Definition von Objekten in objektorientierten Programmiersprachen hat.

**Object Query Language** Object Query Language ist stark an SQL angelehnt, wobei nicht mit den Spalten der Tabelle sondern mit den Attributen des Objekts Abfragen erstellt werden.

**objektorientierte Datenbank** Das Konzept dieser Datenbanken zielt eine eine bessere und nähere Zusammenarbeit mit objektorientierten Programmiersprachen

**objektrelationale Datenbank** Erweitert das Konzept der RDBMS mit Funktionen des objektorientirten Stiles.

**Poll-Prinzip** Beim Poll-Prinzip fragt der Nutzer nach, ob eine Aktion beendet ist. Bei lange Berechnungen kann dies zu sehr vielen unnötigen Requests führen, bei diesem Anwendungsfall wäre das *Push-Prinzip* sinnvoller.

**Polynomialzeit** In Polynomialzeit lösbar heisst, dass die Laufzeitkomplexität in einem Polynom mit der Form $n^k$, wobei $n$ die Eingabelänge und $k$ eine Konstante ist, dargestellt werden kann.

**Polynomialzeit-Verifizierer** Überprüft einer Lösung zu einem Problem in Polynomialzeit.

**Push-Prinzip** Beim Push-Prinzip wird der Nutzer vom System aktiv benachrichtigt, wenn eine Aktion beendet ist.

**relationale Datenbank** Das Grundkonzept dieser Datenbank ist die Relation.

**Representational State Transfer** Programmierparadigma für die Implementation von Webservices. Es bietet eine Kommunikationsschnittstelle für Webanwendung und wird vorallem für Syste-System-Kommunikation verwendet. REST ist Zustandslos und liefert somit bei einem Aufruf einer URL genau jedes Mal den selben Inhalt zurück.

**semistrukturierte Daten** Daten, welche keiner allgemeine Struktur unterliegen. XML ist eine sehr verbreitete Notation dafür.

**Sonar** Sonar ist ein statisches Code-Analyse Tool.

**Stakeholder** Stakeholder sind für den Requirement Engineer wichtige Quellen zur Identifikation möglicher Anforderungen des Systems.[PR11] Ein Stakeholder ist eine Person, die in irgendeiner Weise vom Projekt betroffen ist, jedoch nicht notwendigerweise direkt Einfluss auf den Projektverlauf haben muss.

**Structured Query Language** Abfrage- und Definitionssprache von Datenbanken.

**Turingmaschine** Eine abstrakte Rechenmaschine mit der Leistungsfähigkeit sowohl realer Computer als auch anderer mathematischer Definitionen dessen, was berechnet werden kann.[HMU11]

**vertikaler Durchstich** Beim vertikaler Durchstich, auch vertikaler Prototyp genannt, ist ein Teil der Applikation durch alle Ebenen hindurch implementiert.

**VIM** Ein Texteditor, welcher auf vi basiert und sehr viele Möglichkeiten (z. B. Suchen und Ersetzen) und Shortcuts (z. B. mehrere Zeilen kopieren, verschieben oder löschen) besitzt.

**WebHook** Ein nicht standardisiertes Verfahren, um ständiges Polling zu umgehen. Dabei wird nach einem gewissen Event ein POST-Request an die angegebene URL geschickt.

**YAML** Markup Language mit wenig Overhead und in einer menschenlesbarer Form. Der Name ist ein rekursives Akronym für „YAML Ain't Markup Language“ (früher „Yet Another Markup Language“).

# Literaturverzeichnis


[AA92] Abramson, D. ; Abela, J.: A Parallel Genetic Algorithm for Solving the School Timetabling Problem. In: *Division of Information Technology, C.S.I.R.O*, 1992

[Abd06] Abdullah, Salwani: *Heuristic Approaches for University Timetabling Problems*, The University of Nottingham, Diss., 2006

[Abr91] Abramson, D.: *Constructing School Timetables using Simulated Annealing: Sequential and Parallel Algorithms*. 1991

[Bra15] Braisch, Matthias: *Stundenplangenerierung mit CLP*. `http://www.fh-wedel.de/~si/seminare/ss01/Ausarbeitung/3.stundenplan/splan2.htm#konplex`. Version: Mai 2015

[Bre00] Brewer, Dr. Eric A.: Towards Robust Distributed Systems, 2000

[cit15] *Top 10,000 cited articles in Computer Science [CiteSeer.Continuity; Steve Lawrence, Kurt Bollacker, Lee Giles]*. `http://ww2.ii.uj.edu.pl/~tabor/prII09-10/perl/allarticles.html`. Version: Mai 2015

[Coo71] Cook, Stephen A.: The Complexity of Theorem-Proving Procedures / University of Toronto. Version: 1971. `http://www.cs.toronto.edu/~sacook/homepage/1971.pdf`. 1971. – Forschungsbericht

[Deo10] Deolalikar, Vinay: P != NP / HP Research Labs, Palo Alto. Version: 2010. `http://www.win.tue.nl/~gwoegi/P-versus-NP/Deolalikar.pdf`. 2010. – Forschungsbericht

[Esf15] Esfahbod, Behnam: *P np np-complete np-hard*. `http://commons.wikimedia.org/wiki/File:P_np_np-complete_np-hard.svg#mediaviewer/File:P_np_np-complete_np-hard.svg`. Version: März 2015. – Lizenziert unter CC BY-SA 3.0 über Wikimedia Commons

[FR14] Fielding, Roy ; Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content / RFC Editor. Version: June 2014. `http://www.rfc-editor.org/rfc/rfc7231.txt`. RFC Editor, June 2014 (7231). – RFC. – ISSN 2070–1721

[git15a] *Git*. `http://git-scm.com/`. Version: Mai 2015

[git15b] *knobli/simplatyser*. `https://github.com/knobli/simplatyser`. Version: Mai 2015

[GJ79] Garey, M.R. ; Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979 (A Series of books in the mathematical sciences)

[Gmb15] GmbH, Gruber & P.: *Stundenplan Software für Schulen - UntisExpress*. `http://www.school-timetabling.com/`. Version: März 2015

[GWB03] Gröbner, Matthias ; Wilke, Peter ; Büttcher, Stefan: A Standard Framework for Timetabling Problems. In: Burke, Edmund (Hrsg.) ; De Causmaecker, Patrick (Hrsg.): *Practice and Theory of Automated Timetabling IV*. Springer Berlin Heidelberg, 2003 (Lecture Notes in Computer Science). – ISBN 978–3–540–40699–0

[HBSA11] Hussin, Burairah ; Basari, Abd Samad H. ; Shibghatullah, Abdul S. ; Asmai, Siti A.: IEEE Conference on Open Systems. In: *Exam Timetabling Using Graph Colouring Approach*, 2011

[HHMS09] Hindel, Bernd ; Hörmann, Klaus ; Müller, Markus ; Schmied, Jürgen ; Preisendanz, Christa (Hrsg.): *Basiswissen - Software-Projektmanagement*. 3., überarbeitete und erweiterte Auflage. dpunkt.verlag GmbH, 2009

[HMU11] Hopcroft, J.E. ; Motwani, R. ; Ullman, J.D.: *Einführung in Automatentheorie, Formale Sprachen und Berechenbarkeit*. Pearson Deutschland, 2011 (Pearson Studium - IT). – ISBN 9783868940824

[Jaf14] Jaffer, Muhammad A.: Time Table and Scheduled Meetings Problem - Constraint Optimization through Evolutionary Algorithm / Institute of Business Administration Karachi, Pakistan. Version: 2014. `http://www.academia.edu/6106691/Time_Table_and_Scheduled_Meetings_Problem_Constraint_Optimization_through_Evolutionary_Algorithm`. 2014. – Forschungsbericht

[KN12] Krumke, S.O. ; Noltemeier, H.: *Graphentheoretische Konzepte und Algorithmen*. Vieweg+Teubner Verlag, 2012 (Leitfäden der Informatik). – ISBN 9783834822642

[KPP04] Kellerer, H. ; Pferschy, U. ; Pisinger, D.: *Knapsack Problems*. Springer, 2004. – ISBN 9783540402862

[KS] Koyuncu, Baki ; Seçir, Mahmut: Student Time Table by using Graph Coloring Algorithm / Ankara University Computer Engineering Department. – Forschungsbericht

[Lim10] Limited, I.E.S.: *Introduction to Database Systems*. Pearson Education, 2010. – ISBN 9788131731925

[Lip10] Lipton, Richard J.: *Fatal Flaws in Deolalikar's Proof?* `https://rjlipton.wordpress.com/2010/08/12/fatal-flaws-in-deolalikars-proof/`. Version: August 2010

[LS] Lickel, Roman ; Santschi, Raffael: *R&R Route Planer - Traveling Salesman Problem*. – Semesterarbeit 2013 im Bereich Algorithmen

[McM14] McMillan, M.: *Data Structures and Algorithms with JavaScript*. O'Reilly Media, 2014. – ISBN 9781449373962

[Nag15] Nagl, Manfred: *Algorithmus der Woche - Informatikjahr 2006*. `http://www-i1.informatik.rwth-aachen.de/~algorithmus/algo15.php`. Version: März 2015

[PB04] Pearson, D. ; Bryant, V.: *Decision Maths 1*. Pearson Education, 2004 (Advanced Maths for Aqa). `http://www.suffolkmaths.co.uk/pages/Maths%20Projects/Projects/Topology%20and%20Graph%20Theory/Chinese%20Postman%20Problem.pdf`. – ISBN 9780435513351

[pic11] *Greedy search path example*. `http://commons.wikimedia.org/wiki/File:Greedy-search-path-example.gif`. Version: April 2011

[Pie11] Pierhöfer, Harald: Bäume und Backtracking, 2011

[PR11] Pohl, Klaus ; Rupp, Chris ; Preisendanz, Christa (Hrsg.) ; Schönfeldt, René (Hrsg.) ; Lötsch, Nina (Hrsg.): *Basiswissen - Requirements Engineering*. 3., korrigierte Auflage. dpunkt.verlag GmbH, 2011

[Ric12] Richter, Dr. E.: *Die Klassen P und NP*. `http://apache.cs.uni-potsdam.de/de/profs/ifi/theorie/lehre/ss12/ti2-ss12/folien/Folien11-PundNP.pdf/at_download/file`. Version: Juni 2012

[Sau12] Sautter, Leonie: *Approximationsalgorithmen am Beispiel des Traveling Salesman Problem*, Karlsruher Institut für Technologie, Diss., 2012. `http://parco.iti.kit.edu/henningm/Seminar-AT/seminar-arbeiten/Sautter_final.pdf`

[Sch08] Schrader, Rainer: Graphentheorie Zentrum für Angewandte Informatik Köln, 2008

[Sie03a] Siegfried, Robert: Algorithmische Graphentheorie, 2003

[Sie03b] Siegfried, Robert: Färbungen auf Graphen, 2003

[SL12] Spillner, Andreas ; Linz, Tilo ; Preisendanz, Christa (Hrsg.): *Basiswissen Softwaretest*. 5., überarbeitete und aktualisierte Auflage. dpunkt.verlag GmbH, 2012

[son15] *SonarSource - Continuous Inspection of Code Quality*. http://www.sonarsource.com/. Version: Mai 2015

[spr15a] *Spring Boot*. http://projects.spring.io/spring-boot/. Version: Mai 2015

[spr15b] *Spring Data*. http://projects.spring.io/spring-data/. Version: Mai 2015

[Ste13] Stephens, R.: *Essential Algorithms: A Practical Approach to Computer Algorithms*. Wiley, 2013 (Essentials series). – ISBN 9781118797297

[Vai13] Vaish, G.: *Getting Started with NoSQL*. Packt Publishing, 2013. – ISBN 9781849694995

[Wal15] Walther, Tilman: *Eigenschaften von Algorithmen*. http://www.tilman.de/uni/ws03/alp/eigenschaftenVonAlgorithmen.php. Version: Juni 2015

[wik15a] *Generische Programmierung in Java – Wikipedia*. http://de.wikipedia.org/wiki/Generische_Programmierung_in_Java. Version: Juni 2015

[wik15b] *Representational State Transfer*. http://de.wikipedia.org/wiki/Representational_State_Transfer. Version: Mai 2015

[zha15] *Vorlagen PA/BA*. https://intra.zhaw.ch/departemente/school-of-engineering/bachelorstudium/projekt-und-bachelorarbeiten/vorlagen-paba.html. Version: Mai 2015

# Abbildungsverzeichnis


2.1 Projektplan . . . . . 10

3.1 Übersicht der Komplexitätsklassen ($P! = NP$ und $P = NP$) . . . . . 13
3.2 Suchablauf eines Greedy Algorithmus . . . . . 14

4.1 Hierarchie der Reduktion zum Beweis der NP-Vollständigkeit . . . . . 16
4.2 Kanten Färbung eines Graphen mit 10 Knoten . . . . . 17
4.3 Problem des Handlungsreisenden mit 4 Wegpunkten . . . . . 18
4.4 Beispiel für ein Briefträgerproblem . . . . . 19
4.5 Berechnungszeiten für das Rucksack-Problem mit verschiedenen Eingabeparametern im Vergleich . . . . . 28

5.1 Systemkontext . . . . . 29
5.2 Systemumgebung . . . . . 30
5.3 Use-Case Diagramm . . . . . 32
5.4 Satzschablone . . . . . 36

6.1 System Übersicht . . . . . 44
6.2 Architekturaufbau des Systems . . . . . 45
6.3 Flussdiagramm des Arbeitsablaufs . . . . . 47
6.4 Start eines beliebigen Problemes . . . . . 48
6.5 Abspeichern des Resultates eines beliebigen Problemes . . . . . 49
6.6 CAP-Theorem . . . . . 50
6.7 Klassendiagramm der verschiedenen Problemklassen . . . . . 57
6.8 Klassendiagramm der verschiedenen Resultatklassen . . . . . 58

7.1 Vertikaler Durchstich mit dem Rucksack-Problem . . . . . 59
7.2 Nachrichten der Statusänderungen von Berechnungen . . . . . 61
7.3 Nutzer-Schnittstellenbeschreibung von Swagger . . . . . 62
7.4 Eingabeparameter für eine Rucksack-Berechnung . . . . . 63
7.5 Eingabeparameter für eine Knotenfärbung-Berechnung . . . . . 63
7.6 Eingabeparameter für eine Routen-Berechnung . . . . . 64
7.7 Eingabeparameter für eine Briefträger-Routen-Berechnung . . . . . 64
7.8 Eingabeparameter für eine Stundenplan-Berechnung . . . . . 66
7.9 Eingabeparameter für eine Spielplan-Berechnung . . . . . 67
7.10 Darstellung der Package-Struktur in IntelliJ . . . . . 69
7.11 Github Repository des Simplatyser Projekts . . . . . 70
7.12 Advanced Rest client . . . . . 71

9.1 Konzept des übergeordneten Projekts . . . . . 76

A.1 Risikomatrix . . . . . II

# Listings


6.1 Tabellendefinition in relationalem Datenbanksystem . . . . . 50
6.2 Abfrage in relationalem Datenbanksystem . . . . . 51
6.3 Typendefinition in objektrelationalem Datenbanksystem . . . . . 51
6.4 Verwendung von Typendefinition in objektrelationalem Datenbanksystem . . . . . 51
6.5 Abfrage in objektrelationalem Datenbanksystem . . . . . 51
6.6 Verwendung von Array in objektrelationalem Datenbanksystem . . . . . 51
6.7 Objektdefinition in objektorientierem Datenbanksystem . . . . . 52
6.8 Abfrage in objektorientierem Datenbanksystem . . . . . 52
6.9 Personen Element in JSON Format . . . . . 53
6.10 Abfrage in MongoDB . . . . . 53
6.11 Resultat der Abfrage in MongoDB . . . . . 53
6.12 Serialisierung zeilenorientierte Datenbank . . . . . 53
6.13 Serialisierung spaltenorientierte Datenbank . . . . . 53
6.14 Abfrage in Neo4j . . . . . 54

7.1 Aufbau einer Antwort auf eine Statusabfrage . . . . . 60
7.2 Beispiel einer WebHook Konfiguration für Slack . . . . . 60
7.3 Ausschnitt einer Eingabe für das Stundenplanproblem für die Rahmenbedingungen 65
7.4 Ausschnitt eines Resultats einer Spielplan Erstellung . . . . . 67

A.1 Beispiel einer Eingabe für das Rucksack-Problem . . . . . IV
A.2 Beispiel für Eingabedaten des Rucksack-Problems für den Algorithmus . . . . . IV
A.3 Beispiel eines Resultates für das Rucksack-Problem aus Algorithmus-Sicht . . . . . V
A.4 Beispiel eines Endresultates für das Rucksack-Problem . . . . . V
A.5 Beispiel einer Eingabe für das Knotenfärbungsproblem . . . . . V
A.6 Beispiel für Eingabedaten des Knotenfärbungsproblems für den Algorithmus . . . . VI
A.7 Beispiel eines Resultates für das Knotenfärbungsproblems aus Algorithmus-Sicht . VI
A.8 Beispiel eines Endresultates für das Knotenfärbungsproblems . . . . . VII
A.9 Beispiel einer Eingabe für das Problem des Handlungsreisenden . . . . . VII
A.10 Beispiel für Eingabedaten des Problem des Handlungsreisenden für den Algorithmus VIII
A.11 Beispiel eines Resultates für das Problem des Handlungsreisenden aus Algorithmus-Sicht . . . . . VIII
A.12 Beispiel eines Endresultates für das Problem des Handlungsreisenden . . . . . IX
A.13 Beispiel einer Eingabe für das Briefträgerproblem . . . . . IX
A.14 Beispiel für Eingabedaten des Briefträgerproblems für den Algorithmus . . . . . X
A.15 Beispiel eines Resultates für das Briefträgerproblem aus Algorithmus-Sicht . . . . . X
A.16 Beispiel eines Endresultates für das Briefträgerproblem . . . . . XI
A.17 Beispiel einer Eingabe für das Stundenplanproblem . . . . . XI
A.18 Beispiel für Eingabedaten des Stundenplanproblems für den Algorithmus . . . . . XIII
A.19 Beispiel eines Resultates für das Stundenplanproblem aus Algorithmus-Sicht . . . . XIV
A.20 Beispiel eines Endresultates für das Stundenplanproblem . . . . . XV
A.21 Beispiel einer Eingabe für das Spielplanproblem . . . . . XVI
A.22 Beispiel für Eingabedaten des Spielplanproblems für den Algorithmus . . . . . XVII
A.23 Beispiel eines Resultates für das Spielplanproblem aus Algorithmus-Sicht . . . . . XVIII
A.24 Beispiel eines Endresultates für das Spielplanproblem . . . . . XIX

# Tabellenverzeichnis


2.1 Meilensteine . . . . . 8
2.2 Geplante Abwesenheiten . . . . . 9
2.3 Zeitschätzung auf Arbeitspaketebene . . . . . 11

4.1 Schulfächer . . . . . 20
4.2 Lehrer . . . . . 20
4.3 Klassen . . . . . 20
4.4 Klassenzimmer . . . . . 21
4.5 Möglicher Stundenplan - Variante 1 . . . . . 21
4.6 Möglicher Stundenplan - Variante 2 . . . . . 21
4.7 Knapsack Objekte mit Gewicht und Profit . . . . . 22
4.8 Wahrheitstabelle zur aussagenlogischen Formel . . . . . 24
4.9 Wahrheitstabelle zur 3-SAT Formel . . . . . 25
4.10 Eingabe- und Ausgabedaten der ausgewählten Probleme . . . . . 26
4.11 Berechnungszeiten bei verschiedenen Eingabeparametern für das Rucksack-Problem 27

5.1 Liste der Stakeholder . . . . . 31
5.2 Vorlage für Use Case Spezifikation . . . . . 32
5.3 Use Case UC-1: Lösung beauftragen . . . . . 33
5.4 Use Case UC-2: Berechnung starten . . . . . 33
5.5 Use Case UC-3: Eingabe Parameter abholen . . . . . 34
5.6 Use Case UC-4: Status abfragen . . . . . 34
5.7 Use Case UC-5: Resultat speichern . . . . . 35
5.8 Use Case UC-6: Resultat abholen . . . . . 35
5.9 Vorlage für Anforderungen . . . . . 36
5.10 Anforderung RF-F1 . . . . . 37
5.11 Anforderung RF-F2 . . . . . 37
5.12 Anforderung RF-F3 . . . . . 38
5.13 Anforderung RF-F4 . . . . . 38
5.14 Anforderung RF-F5 . . . . . 39
5.15 Anforderung RF-F6 . . . . . 39
5.16 Anforderung RF-F7 . . . . . 40
5.17 Anforderung RF-F8 . . . . . 40
5.18 Anforderung RF-F9 . . . . . 41
5.19 Qualitätsanforderung RF-NF1 . . . . . 42
5.20 Qualitätsanforderung RF-NF2 . . . . . 42
5.21 Qualitätsanforderung RF-NF3 . . . . . 43
5.22 Priorität der Anforderungen . . . . . 43

6.1 Nutzwertanalyse - Datenbank Varianten . . . . . 56

7.1 Visuelle Darstellung der Zeitfenster-Berechnung . . . . . 65
7.2 Übersicht der angebotenen Schnittstellen . . . . . 68

8.1 Testprotokoll . . . . . 73

A.1 Risikoermittlung . . . . . I
A.2 Risikobewertungsschema . . . . . II
A.3 Risikobewertung . . . . . II

A.4 Risikoanalyse – Massnahmen . . . . . III