

Projektbericht Aladin

Inhaltsverzeichnis

1. Einleitung	1
1.1. Problemerkläuterung	1
1.2. Vorstellung Aladin	1
1.3. Ziel der Projekte	1
2. Steuerrecht	3
2.1. Aufgabenbeschreibung	3
2.2. Technische Umsetzung	3
2.3. Ergebnisse	20
2.4. Ausblick	22
3. Generierung Ereignisgesteuerte Prozessketten	24
3.1. Aufgabenbeschreibung	24
3.2. Planung	24
3.3. Umsetzung	24
3.4. Probleme	32
3.5. Ausblick	32
4. Sortieralgorithmen	33
4.1. Aufgabenbeschreibung	33
4.2. Technische Umsetzung	33
4.3. Ergebnisse	45
4.4. Ausblick	48
5. Generierung von zufallsbasierten Molekülen	50
5.1. Aufgabenbeschreibung	50
5.2. Umsetzung	50
5.3. Probleme	56
5.4. Ausblick	57
6. RSA Verschlüsselung	58
6.1. Aufgabenbeschreibung	58
6.2. Umsetzung	60
6.3. Probleme	66
6.4. Ergebnisse	67
6.5. Ausblick	68
7. Reflexion	69
7.1. Timm Hauschild	69
7.2. Ruben-David Kraus	69
7.3. Alexander Schulz	70
7.4. Dustin Doege	70
7.5. Philip Poplutz	71
8. Schluss	72

8.1. Folgeprojekte	72
--------------------------	----

Abschnitt 1. Einleitung

1.1. Problemerk  uterung

Das Problem der manuellen Erstellung von   bungsaufgaben betrifft sowohl Studierende als auch Lehrende und kann den Lernprozess stark beeintr  chtigen. Wenn es nicht gen  gend   bungsmaterial und Probeklausuren gibt, k  nnen Studierende Schwierigkeiten haben, sich ad  quat auf Pr  fungen vorzubereiten und ihre F  higkeiten in dem Fachbereich zu verbessern. Die Wiederholung von   bungsaufgaben kann dazu f  hren, dass Studierende die L  sungen auswendig lernen, ohne sich wirklich mit dem Material auseinanderzusetzen. Dar  ber hinaus k  nnen die fehlenden Anreize zur Bearbeitung der Aufgaben dazu f  hren, dass das Interesse der Studierenden am Lernprozess nachl  sst. Die fehlende Betreuung beim L  sen von   bungsaufgaben kann zu Unsicherheiten und Verwirrung f  hren, wenn Studierende auf Probleme bei der L  sung der Aufgaben sto  en. Die Orts- und Zeitgebundenheit von Besprechungen zur Bearbeitung von   bungsaufgaben kann zudem dazu f  hren, dass Studierende eingeschr  nkt sind und sich nicht optimal auf Pr  fungen vorbereiten k  nnen. Das manuelle Erstellen und Korrigieren von   bungs- und Klausuraufgaben erfordert auch viel Zeit und kann dazu f  hren, dass Lehrende nicht gen  gend Zeit haben, um sich auf ihre eigentliche Aufgabe zu konzentrieren - n  mlich den Studierenden das n  tige Wissen zu vermitteln und sie bei der Weiterentwicklung ihrer F  higkeiten zu unterst  tzen. Diese Probleme sind besonders gravierend in F  chern mit vielen Studierenden und begrenzten Ressourcen, wie beispielsweise in gro  en Vorlesungen oder Online-Kursen.

1.2. Vorstellung Aladin

Um diese Probleme zu l  sen, wurde das Konzept und Software-Framework ALADIN entwickelt und mehrfach gef  rdert. ALADIN erm  glicht die zufallsbasierte Generierung von beliebig vielen Aufgaben und erlaubt den Studierenden, die Aufgaben online zu l  sen. Falls n  tig, bietet das System auch L  sungshilfen an. Die Studierenden werden durch Gamification motiviert, indem sie beispielsweise Punkte oder Belohnungen f  r das L  sen von Aufgaben erhalten. Das System erm  glicht auch eine asynchrone Zusammenarbeit und Betreuung. Mit ALADIN sollen die genannten Probleme angegangen werden, um die Kompetenz der Studierenden zu verbessern, ihre Klausurergebnisse zu optimieren und ihren Studienerfolg zu erh  hen.

Die Entwicklung von ALADIN ist eine wichtige Ma  nahme, um die Schwierigkeiten von Studierenden und Lehrenden im Zusammenhang mit   bungsaufgaben und Klausuren zu l  sen. Die M  glichkeit, beliebig viele Aufgaben zu generieren und online zu l  sen, wird den Studierenden mehr   bungsm  glichkeiten bieten und dadurch ihr Verst  ndnis von Konzepten verbessern. Die Betreuung durch das System wird sicherstellen, dass die Studierenden Unterst  tzung erhalten, wenn sie sie ben  tigen. Die M  glichkeit, L  sungshilfen zu erhalten, wird auch dazu beitragen, dass sich die Studierenden nicht entmutigt f  hlen und ihre F  higkeiten verbessern k  nnen. Zusammenfassend kann man sagen, dass ALADIN ein wichtiger Beitrag zur Verbesserung des Lernprozesses von Studierenden sein kann.

1.3. Ziel der Projekte

Mithilfe der Projekte sollen verschiedenste Aufgabentypen bez  glich folgender Aspekte betrachtet

werden:

1. Machbarkeit prüfen

Lassen sich die Aufgabentypen ohne Weiteres umsetzen oder sind sie gegebenenfalls zu komplex und müssen in mehrere Projekte aufteilt werden?

Handelt es sich eventuell um NP-Vollständige Probleme?

2. Algorithmen entwickeln

Es sollen Algorithmen entwickelt werden, die zur Umsetzung des Aufgabentyps benötigt werden.

3. Erzeugen und Extrahieren benötigter Daten

Welche Daten benötigt der Algorithmus, um vielfältige Aufgaben zu generieren?

4. Entwurf zur Generierung von Aufgabe, Lösung und Lösungshilfen und der Generierungsparameter

Finden geeigneter Methoden zur Umsetzung der Aufgabentypen, bspw. Templates, Regeln, Machine-Learning-Algorithmen.

Finden einer geeigneten Darstellung der Aufgabenstellung, deren Lösungshilfen und Lösungen zum Beispiel als Text, Grafiken oder mittels interaktiver Elemente.

Finden geeigneter Parameter, welche den Generierungsprozess beeinflussen und anhand welcher der Schwierigkeitsgrad, die Qualität und Vielfalt der Aufgabe festgelegt werden können.

Gesamtziel:

Automatische zufallsbasierte Generierung unendlich vieler Aufgaben, Lösungshilfen und Lösungen.

Abschnitt 2. Steuerrecht

2.1. Aufgabenbeschreibung

2.1.1. Gegebenheiten / Design der Aufgabe

Meine Aufgabe im Projektseminar war es, Aufgaben zum Steuerrecht umzusetzen. Dabei ging es darum, Aufgaben zu generieren bei denen die Studierenden das zu versteuernde Einkommen berechnet. Die Aufgaben sollen mit zufälligen Werten und Kombinationen aus einem Pool und Sachverhaltstypen generiert werden. Um die Lösung des Studenten zu prüfen, muss neben der Aufgabe auch noch die dazugehörige Lösung erstellt werden. Basis dieser Aufgaben bildet ein Text, mit unterschiedlichen steuerlichen Sachverhalten, welche die Studierenden herauslesen müssen. Die Sachverhalte sind dabei miteinander verwoben, können aber müssen nicht aufeinander aufbauen.

Beispielaufgabe mit eingefärbten Sachverhalten

Die ledige Steuerpflichtige K hat ein Sechsfamilienhaus gekauft (Baujahr 1980). Sie nutzt keine Wohnung zu eigenen Wohnzwecken. Übergang von Nutzen und Lasten war am 1.11.2019. Der **Kaufpreis beträgt 1.000.000 EUR**, davon entfallen **200.000 EUR auf den Grund und Boden**. Weitere Aufwendungen: **Notarkosten (10.000 € inkl. USt)**, **GrESt (35.000 €)**, **Grundbuchänderung (590 €)**. Zur Finanzierung des Objektes hat sie 2019 ein **Darlehen in Höhe von 1.000.000 €** aufgenommen, das nach Abzug des Disagios zu 95 % ausbezahlt wurde. Im Jahr 2019 hat sie – neben dem Disagio - **Zinsen an die finanzierende Bank i.H.v. 8.000 €** gezahlt.

Dies soll dazu führen, dass die Studierenden zum aktiven Kombinieren angeregt werden. Die Anzahl an unterschiedlichen Sachverhaltstypen, soll stetig erweiterbar bleiben, im Moment werden Gehalt, Kapitalvermögen, Vermietung und für alle Typen Werbungskosten unterstützt. Um eine natürliche Satzstruktur zu wahren, muss der Text dabei unterschiedliche Satzkonstruktionen enthalten. Dazu sollen mehrere 'Versionen' des Satzes genutzt werden, welche dann zufällig ausgewählt werden. Die Wertebereiche der Aufgaben müssen sich in einem realistischen Bereich bewegen, um nahezu an die Realität zu kommen, sodass sich der Text umso mehr 'real' anfühlt.

2.1.2. Anforderungen

Aus diesen Gegebenheiten ergeben sich folgende Anforderungen an die Software

1. Zufällige Generierung der Summen und Sachverhalte
2. Guter Lesefluss in der Aufgabe durch unterschiedliche Satzstrukturen
3. Generieren der Aufgabe und der Lösung
4. Realistische Wertebereiche
5. Trennung der Generierung und Lösung

2.2. Technische Umsetzung

2.2.1. Allgemeine Architektur

Die Software besteht aus einem Server, welcher die Aufgaben zentral verwaltet, generiert, löst und die Lösungen prüft, sowie einem Client, welcher sich Aufgaben generieren lässt, selbige löst und Lösungen prüfen lässt. Der Server stellt REST-Endpunkte für den Client bereit, welche je nach Funktionalität angesprochen werden. Der Client bietet den Studierenden ein Frontend zur Bedienung und Interaktion mit ALADIN. Ich habe hier eine klare Trennung vorgenommen, einerseits Python, mit seinen zahlreichen Funktionen in Richtung KI, unterzubringen und JavaScript damit die UI responsiv und dynamisch sein kann. Außerdem kann weiterhin klar zwischen den Funktionalitäten abgegrenzt und separat an ihnen gearbeitet werden. Das ganze nur als einzelne Webanwendung zu betreiben, war in den Funktionalitäten so eingegrenzt, dass ich mich dazu entschieden habe, hier diese Trennung vorzunehmen.

2.2.2. Wahl der Frameworks

Der Server wurde in Python programmiert. Einerseits, weil es die Programmiersprache ist, in der ich am fortgeschrittensten bin, andererseits, weil sich in Python viele Frameworks zum Language Processing angesiedelt haben. Als zwei Haupt-Frameworks habe ich PythonFastAPI ([fastapi](https://fastapi.tiangolo.com), <https://fastapi.tiangolo.com>), für die Bereitstellung der REST-Endpunkte sowie HuggingFace (<https://huggingface.co>), für die Language Processing gewählt.

2.2.2.1. REST-API

Meine Wahl für das Framework für die REST-API fiel, wie beschrieben, auf FastAPI. FastAPI bietet eine sehr schnelle Bearbeitung der Requests und hat einen merkbaren Geschwindigkeitsvorteil. Das Framework benötigt lediglich eine Datei für den Start und einen "uvicorn" als ASGI-Webserver um Requests entgegenzunehmen. Zuerst habe ich den Webserver mit Flask betreiben, da hier aber die Abarbeitung der Requests nicht schnell genug war, habe ich auf FastAPI gewechselt. Die Dekoratoren für Funktionen der Endpunkte ist aber sehr ähnlich / nahezu gleich. Wer also Flask kennt, wird mit FastAPI wenig Probleme haben.

2.2.2.2. Frontend

Das Frontend habe ich aufgrund des Studiums mit Vuejs programmiert. Ich konnte hier bereits einige Erfahrungen in diesem Semestern sammeln und da ich die Lernkurve nicht als zu hoch empfand, habe ich mich für dieses Framework entschieden. Zuerst nur in einzelnen Aspekten, da aber die Aufgabenlösung und Anzeige eine dynamische Webseite erfordert, habe ich das komplette Frontend (vorher Flask mit Templates) auf komplett Vuejs umgestellt. Vuejs spricht je nach Funktionalität die REST-Endpunkte an um, im JSON Format, mit dem Server zu interagieren. Um ein einheitliches Styling zu gewährleisten habe ich als Frontend Framework Bootstrap gewählt.

2.2.3. Angewendete Konzepte

Um in folgenden Absätzen Fragen zu klären, erlaube ich zunächst, welche Konzepte und Ideen grundlegende Funktionalitäten in der Software umsetzen. Da wir im Steuerrecht unterschiedliche Sachverhalte pro Aufgabe haben, braucht es eine Datenstruktur für die Sachverhalte. Um diese dann zufällig auszuwählen bzw. aneinander zu hängen ist eine weitere Struktur von Nutzen, die diese Funktion übernimmt.

2.2.3.1. Sachverhalt (Case)

Ein Sachverhalt, nachfolgend auch Case genannt, ist die Repräsentation der notwendigen Informationen zu einem steuerlichen Sachverhalt. Diese Informationen sind in meinen Augen

1. Name des Sachverhalts
2. Art des Sachverhalts (Gehalt, Vermietung, Werbungskosten, Beteiligung,...)
3. Verb das zum Sachverhalt gehört
4. Subjekt / handelnde Person im Sachverhalt
5. Zugehörige Summe

Diese Informationen werden pro Sachverhalt einzeln gespeichert und durch einen Klasse (case.py) abgebildet. Für jeden Sachverhalt werden neuen Klassen instanziiert und mit entsprechenden Werten gefüllt, welche auch nachträglich durch Getter und Setter manipuliert / ausgelesen werden können. Die Klasse hält die notwendigen Informationen wie folgt

Case Klasse

```
class Case:
    def __init__(self, name="", subject="", verb="", object="", number=0) ->
    None:
        self.name = name
        self.subject = subject
        self.verb = verb
        self.object = object
        self.number = number
```

— Python

2.2.3.2. Pool / Auswahl an Sachverhalten (Nodepool)

Um alle instanziierten Sachverhalte in einer Datenstruktur halten zu können, benötigt es eine, dem Case übergeordnete, Datenstruktur. Zuerst habe ich versucht, auch im Hinblick auf die Generierung, die Datenstruktur als Graph aufzubauen, bei der jeder Case ein eigener Unterknoten und die Blätter des Knotens die Informationen zum Case sind. Da dieser Ansatz im Aufbau nicht sehr intuitiv war, habe ich mich im Nachhinein für eine "Pool" entschieden. Dieser Pool hält eine Liste an Sachverhalten und stellt Methoden bereit um gezielt oder zufällig Sachverhalte aus dem Pool auszuwählen, aber auch einzelne Sachverhalte hinzuzufügen oder zu löschen. Diese Funktionalität bildet die Klasse nodepool.py ab, welche ein direktes Zusammenspiel mit der case.py Klasse bildet.

NodePool Klasse

```
class NodePool:
    def __init__(self, name: str) -> None:
        self.name = name
        self.nodes: List[Case] = []
```

— Python

2.2.3.3. Natürliche Sprache (Mask-Filling)

Die Cases stellen für jeden Sachverhalt Informationen bereit, die unbedingt in der Formulierung vorkommen müssen, woraus sich einzelne Satzbausteine ergeben, welche bei der Satzgenerierung angewendet werden müssen. Jedoch bleiben dann Trennwörter und andere "füllende" Satzbausteine übrig, welche nur mit viel Aufwand fest definiert werden können. Um diese "Lücken" zu schließen und unterschiedliche Satzstrukturen ausbilden zu können, habe ich mir das Konzept des Mask-Filling zu nutze gemacht. Mask-Filling lässt sich nutzen, in dem in den Sätzen Platzhalter, sogenannte "MASKS", eingebaut werden, welche nachträglich von einem Sprachmodell gefüllt werden. Das Sprachmodell teilt den Satz in Tokens auf und sucht aufgrund seiner vorhandenen Daten nach einem möglichst passenden Ersatz für die Maske im Satz, um die Maske danach zu ersetzen. Dabei ist zu beachten, dass das Sprachmodell **nicht** auf steuerliche Sachverhalte trainiert wurde und es daher auch öfter dazu kommen kann, dass die Trennwörter nicht in den Use-Case passen.

Ungefüllte Masken im Satz

```
f"J erhält [MASK] Arbeitnehmer 9500€."
f"Er meldet Arbeitsmittel [MASK] 150€ [MASK]."
f"[MASK] Dividende bekommt [MASK] 67000€."
```

Nachträglich gefüllte Masken

Er ist Angestellter und erhält 6700€. Hier kauft Angestellter Fahrtkosten von 1150€ ab. Als Kapitalbeteiligung erhält P 29000€. Hier meldet Angestellter Heizkosten von 1150€ an. Dabei setzt Angestellter Abschreibungen von 1150€ an. Von die Dividende erhält er 64000€.

2.2.3.4. Strategy Pattern

Da bei der Aufgabengenerierung, je nach Parameterkombination, ein unterschiedliches Verfahren genutzt werden muss, habe ich mir das Strategy Pattern zu nutze gemacht. Beim Strategy Pattern wird es einem Objekt bzw. einer Klasse möglich gemacht, ihr Verhalten zur Laufzeit zu ändern. Dafür benötigt man ein Interface, welches die Methoden definiert, die verwendet werden, einen Context, der sein Verhalten ändert, und die konkrete Implementierung der Methoden, welche verwendet werden sollen. Mit diesem Design kann die Aufgabengenerierung ohne tief verzweigte If-Schleifen den richtigen Algorithmus wählen, sowie eine einfache Änderung bzw. Erweiterung gewährleisten. Das Pattern wird in der Datei `generator_strategy.py` umgesetzt. Die Klasse `GeneratorStrategie` stellt das Interface mit der `generate` Methode bereit und verwendet dabei die Python Bibliothek `Abstract Base Classes`, um eine abstrakte Methode zu definieren. Alle Klassen die diese Klasse erben, definieren die konkreten Methoden und die `Context` Klasse nutzt diese Methoden. Folgende Strategien werden aktuell verwendet:

Tabelle 1. Strategy Patterns

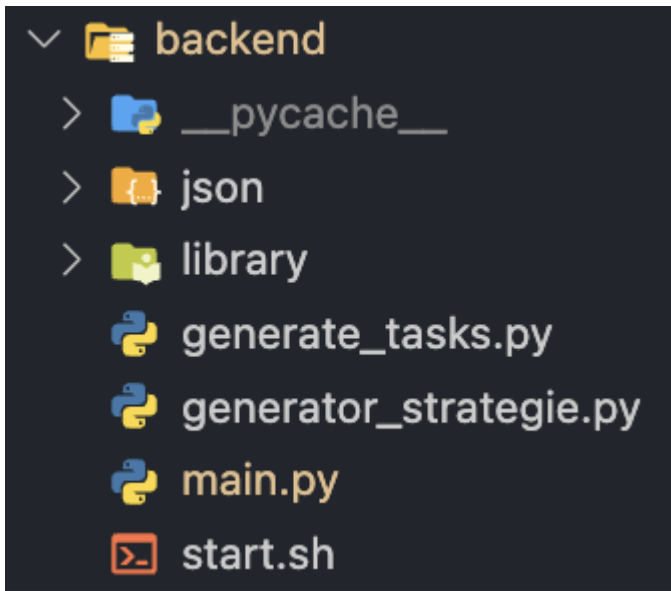
Anzahl Sachverhalte	Anzahl unterschiedliche Sachverhalte	Muss enthalten sein	Strategie
gesetzt	gesetzt	gesetzt	<i>WithDifficultyAndNeededAndAmount</i>
gesetzt	gesetzt	nicht gesetzt	<i>WithDifficultyAndAmount</i>

Da nur der Parameter **Muss enthalten sein** optional ist, sind aktuell nur diese Strategien notwendig.

2.2.4. Backend / REST-API

2.2.4.1. Ordnerstruktur

Die REST-API befindet sich im 'backend' Ordner. Hier gibt es die Subordner und Dateien:



- `generate_tasks.py` → Datei, welche Logik zur Generierung der Aufgaben enthält
- `generator_strategie.py` → Dateien für das Strategy Pattern um dynamisch Algorithmen zur Generierung zu wählen
- `main.py` → FastAPI-Datei, welche gestartet wird und Hauptfunktionen enthält
- `/json` → Ablage JSON-Dateien zum Test
- `/library` → Bibliotheken für andere Module
- `/library/nodepool/case.py` → Klasse, welche einen Sachverhalt repräsentiert
- `/library/nodepool/nodepool.py` → Klasse, welche einen Pool aus Sachverhalten repräsentiert, welche zufällig gewählt werden
- `/library/dependencies.py` → Auflösen und Generieren aller Einnahmen / Ausgaben Fälle
- `/library/laws.py` → Festgelegte, gesetzliche Grundlagen der Fälle
- `/library/numbers.py` → Wertebereiche der Fälle und Rundungsregeln

- `/library/sentenceparts.py` → Satzbausteine für die einzelnen Fälle
- `/library/solution.py` → Klasse, welche eine Lösung für einen Fall darstellt
- `/library/task.py` → Klasse, welche eine Aufgabe mit den einzelnen Fällen darstellt
- `/library/variations.py` → Stellt Funktionen des Sprachmodells bereit und bildet die unterschiedlichen Formulierungen für einen Sachverhalt

2.2.4.2. Kommunikation von Server und Client / Endpunkte

Server und Client kommunizieren über REST-Calls. Der Client sendet je nach gewünschter Funktionalität / Endpunkt eine Anfrage an den Server, welche dann entsprechend vom Server beantwortet wird. Folgende Endpunkte werden vom Server bereitgestellt:

Route	Methode	Beschreibung	Beispiel Request	Beispiel Response
<code>/get-task?{query-string}</code>	GET	Fragt einen neuen Task je nach den festgelegten Parametern in der Query an	<code>curl --request GET \ --url 'http://localhost:8000/get-task?difficulty=1&amount=5'</code>	<code>{ "id": 5, "sentences": ["Die Arbeitnehmerin meldet Heizkosten von 500€ an.", "Dabei setzt Arbeitnehmerin Abschreibungen von 500€ an.", "Arbeitnehmerin [unused_punctuation3] setzt Abschreibungen von 500€ an.", "Pro Unternehmensbeteiligung bekommt G 29000€.", "Derzeit vermietet sie eine Unterkunft für 650€."] }</code>
<code>/generated-tasks</code>	GET	Gibt alle generierten Tasks aus	<code>curl --request GET \ --url http://localhost:8000/generated-tasks</code>	<code>{ "0": false, "1": false, "2": false, "3": false, "4": false, "5": false }</code>
<code>/cases-to-choose</code>	GET	Liste an Cases, die vom Server genutzt werden	<code>curl --request GET \ --url http://localhost:8000/cases-to-choose</code>	<code>["Gehalt-WK", "Vermietung-WK", "Abschreibung", "Gehalt", "Beteiligung", "Dividende", "Vermietung"]</code>

Route	Methode	Beschreibung	Beispiel Request	Beispiel Response
/solve/{task-id}	POST	Löst die Aufgabe mit der festgelegten ID und dem mitgegebenen Payload	curl --request POST \ --url http://localhost:8000/solve/0 \ --header 'Content-Type: application/json' \ --data '[{ "id": 0, "select": "Gehalt-WK", "law": "awdadwa", "num": 1150 }, { "id": 1, "select": "Gehalt", "law": "awdadwa", "num": 1100 }]'	{ "given": { "1": { "name": true, "law": false, "num": false } }, "all_solved": false }
/solution/{task-id}	GET	Komplette Lösung zur Aufgabe mit der ID	curl --request GET \ --url http://localhost:8000/solution/0	[{ "case_name": "Abschreibung", "law": "\$9 Abs. 1", "number": 1900, "type_of_case": "Ausgabe", "hint": "" }, { "case_name": "Dividende", "law": "\$20 Abs. 9", "number": 97000, "type_of_case": "Einnahme", "hint": "" }]
/zve/{task-id}	GET	Zu versteuerndes Einkommen zur Aufgabe mit der ID	curl --request GET \ --url http://localhost:8000/zve/1	7350
/select-options/{task-id}	GET	Optionen für das Select Feld zur Aufgabe mit der ID	curl --request GET \ --url http://localhost:8000/select-options/0	{ "0": { "name": "Vermietung", "value": 650 }, "1": { "name": "Gehalt", "value": 4500 } }

2.2.4.3. Generierung der Aufgaben

Die Aufgaben werden auf Anfrage des Clients je nach Parametern generiert. Folgende Parameter werden im Moment unterstützt:

- Anzahl der Sachverhalte
- Anzahl der unterschiedlichen Sachverhalte

- Sachverhalte welche in der Aufgabe enthalten sein müssen

Diese Parameter können vom Nutzer modifiziert werden und beeinflussen die Aufgabengenerierung. Die Studierenden können diese Parameter über die UI verändern, welche sich dann im Request für die Aufgabengenerierung widerspiegeln. Der Aufgaben- / Task-Endpoint nimmt diese drei Parameter entgegen und wählt mit der `determine_strategy` Methode die Strategie für die Kombination der Parameter für den Context. Darauf werden zuerst die gewünschten Cases aus dem Pool gepickt, danach wird geprüft ob dann bereits die geforderte Anzahl der Cases erreicht ist, wenn nicht werden die restlichen Cases zufällig aufgefüllt, wenn ja werden nur die Cases, die gewünscht sind zurückgegeben. Aus den Cases werden im Anschluss die Lösungen für den Server berechnet sowie die korrekten rechtlichen Grundlagen zugewiesen. Außerdem wird das zu versteuernde Einkommen berechnet und aus den Cases ein Task erstellt, welcher jeder Aufgabe über eine ID identifiziert und eine Liste der Cases, Lösungen und das zVE enthält. Als letztes wird ein JSON vorbereitet, welches die ID des Task sowie alle Cases enthält, bei welchen ein Satz aus jedem Case in der Liste im Task gebaut wird. Für die Generierung der Sätze, werden die unterschiedlichen Satzkonstruktionen in `variations.py` genutzt, welche die `[MASK]` Tokens an unterschiedlichen Stellen als String enthalten. Es wird eine zufällige Variations ausgewählt und anschließend werden alle Masken im Satz durch das Sprachmodell gefüllt, dann wird der Satz zurückgegeben.

```
f"[MASK] {case.object} {case.verb} {case.subject} [MASK] {case.number}€ [MASK].",
f"{case.object} [MASK] {case.verb} {case.subject} [MASK] {case.number}€ [MASK].",
f"[MASK] {case.verb} {case.object} {case.subject} [MASK] {case.number}€ [MASK].",
```

Für eine gesamte Übersicht der Tasks in der Sitzung des Nutzers, hält der Server eine Liste an Tasks, welche bei jeder neuen Generierung um den neuen Task erweitert wird.

Task-Klasse

```
class Task:
    id_generated = itertools.count()
```

```
def __init__(self, cases: list[Case] = [], zve: int = 0, solutions: dict[str,
Solution] = {}) -> None:
    self.id = next(Task.id_generated)
    self.cases: list[Case] = cases
    self.solutions: dict[str, Solution] = solutions
    self.zve = zve
    self.solved = {sol_id: {'name': False, 'law': False, 'num': False} for sol_id
in self.solutions.keys()}
```

```
def to_dict(self):
    return {"id": self.id, "case": [case.to_dict() for case in self.cases],
"solved": self.solved}
```

```
def all_solved(self):
    correct = 0
    for is_correct in self.solved.values():
```

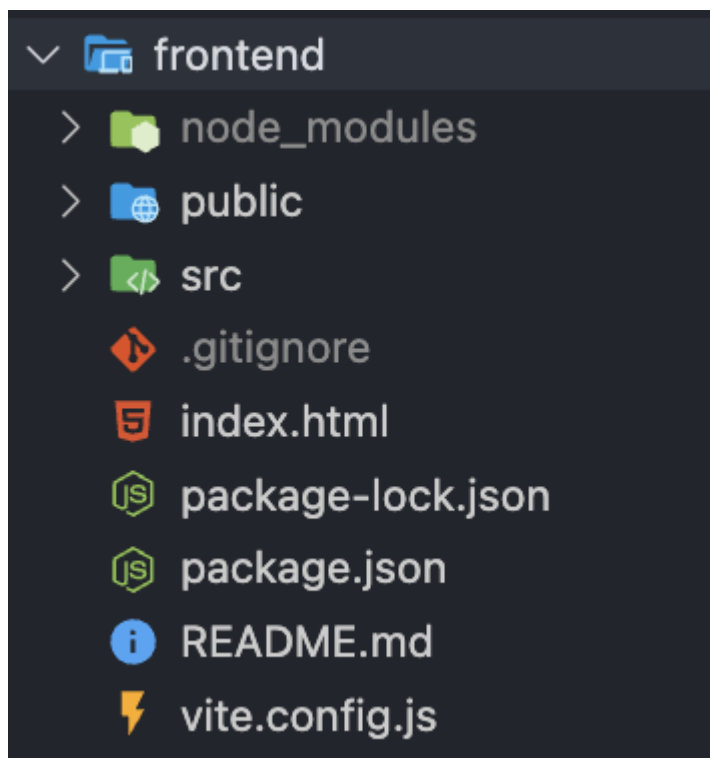
```
if all(is_correct.values()):
    correct +=1
return True if correct == len(self.solutions) else False
```

— Python

2.2.5. Frontend

2.2.5.1. Ordnerstruktur

Das Frontend ist ein Vuejs Projekt und befindet sich im 'frontend' Ordner. Dabei finden sich die Vuejs typischen Unterordner:



- `index.html` → Basis Landing page mit nötigen Script importen und Vuejs App-Container
- `/node_modules` → Module von Nodejs, die benötigt werden
- `/public` → Ordner für statische Assetss
- `/src/App.vue` → Vuejs Haupt App Komponente
- `/src/main.js` → Vuejs Javascript main Datei um App auf Container in index.html zu mounten
- `/src/assets` → Weiterer Ordner für statische Assets
- `/src/components/MainAppComponent.vue` → Vereint alle Komponenten / Spiegel Layout unter Komponenten wieder
- `/src/components/AllTasksComponent.vue` → Anzeigen aller Tasks auf Server
- `/src/components/CustomizeTaskComponent.vue` → Buttons zum einstellen der Aufgabenparameter
- `/src/components/ErrorComponent.vue` → Anzeigen von Errors in der Anwendung
- `/src/components/GivenSolutionComponent.vue` → Anzeige der Lösungstabelle der Aufgabe

- `/src/components/store.js` → Datenstruktur um Daten zwischen Komponenten auszutauschen
- `/src/components/TaskSentenceComponent.vue` → Stellt alle generierten Sätze der Aufgabe da
- `/src/components/UploadFooterComponent` → Upload von Konfigurationen (nicht funktionsfähig)
- `/src/components/UserSolutionComponent.vue` → Lösungsstruktur des Nutzers um Aufgaben zu bearbeiten

2.2.5.2. Einzelne Komponenten

Im Folgenden erkläre ich alle Komponenten die ich im Frontend verwendet habe, dabei setzte ich grundlegendes Wissen in Vuejs / Javascript voraus. Für das Styling wird das CSS-Framework Bootstrap verwendet.

2.2.5.2.1. store.js

Der Store stellt eine *reactive* Komponente in Vuejs dar. Diese Komponente ist dazu da, Daten zwischen Komponenten auszutauschen. In meinem Anwendungsfall ist es am sinnvollsten, die aktuelle Task-ID, den Error sowie die Liste an Sätzen der Sachverhalte im Store zu speichern. Durch die *watch* Lifecycle Methode in Vuejs, können diese Werte einzeln überwacht werden und dynamisch bei Änderung gehandelt werden.

Store.js Datei

```
import { reactive } from 'vue';
```

```
export let store = reactive({
  task_id: null,
  sentences: [],
  error: ""
});
```

— Javascript

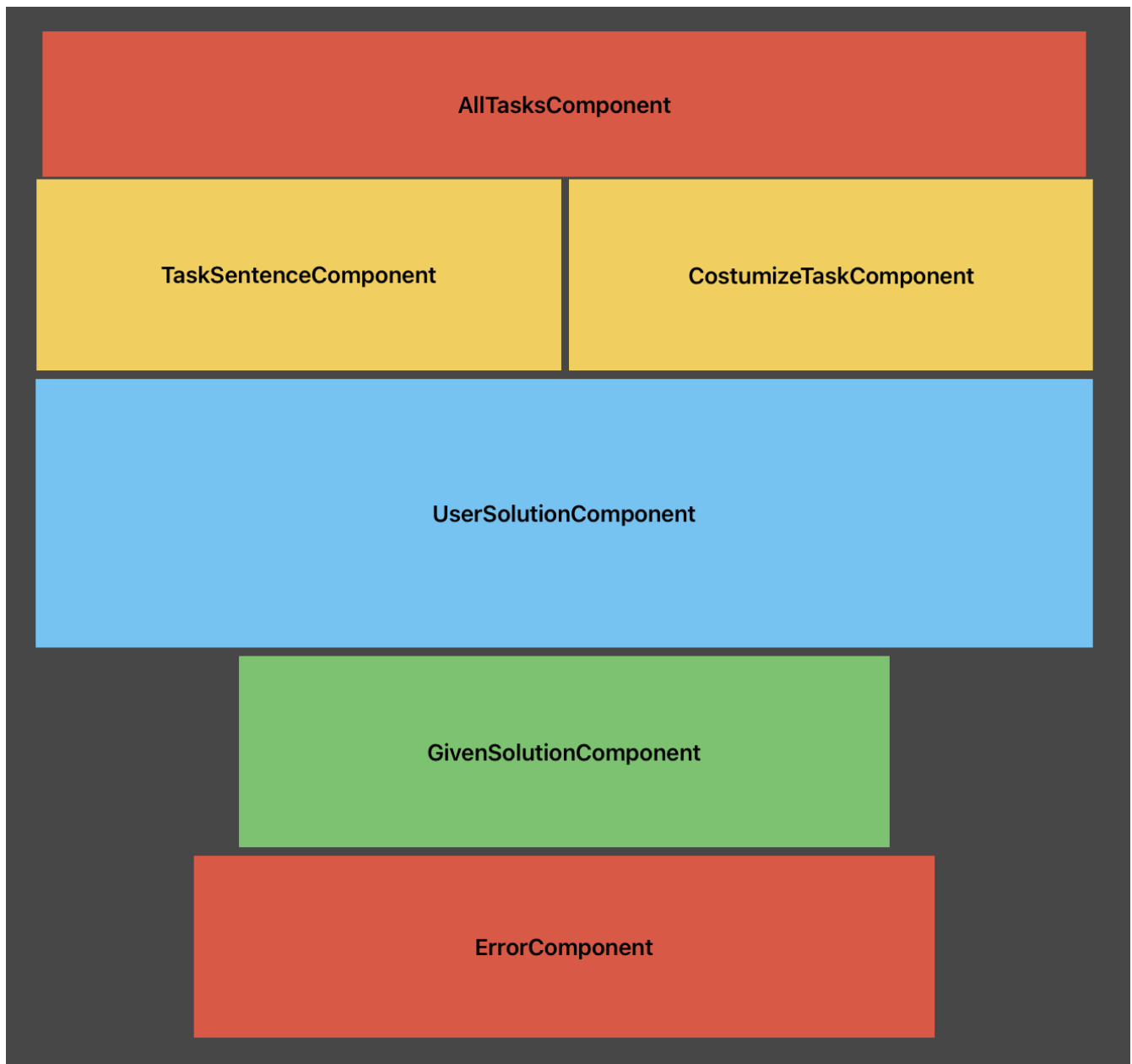
Überwachung eines Store-Wertes

```
computed: {
  task_id() {
    return store.task_id
  }
},
watch: {
  task_id() {
    this.getGeneratedTasks()
  }
}
```

— Javascript

2.2.5.2.2. MainAppComponent

Der MainAppComponent vereint alle Komponenten der Vuejs Anwendung in einem zentralen Bootstrap Layout. Hier gibt es einen zentralen Container, der mehrere Bootstrap Rows beinhaltet. Auf der erstene Row findet sich der **AllTasksComponent**, welcher sich je nach Anzahl der Tasks skaliert und wächst. Darunter befinden sich der **TaskSentenceComponent** mit dem **CostumizeTaskComponent** auf einer Ebene. Beide Komponenten werden dabei zentriert und beanspruchen gleich viel Platz in der Horizontalen. Darunter folgen, jeweils in separaten Reihen, der **UserSolutionComponent**, zur Lösung des Users, der **GivenSolutionComponent** als vorgegebende Lösung, sowie der **ErrorComponent**, für mögliche Fehler in der Anwendung. Die Komponente hat sonst keine Funktionalität.



2.2.5.2.3. AllTasksComponent

Im AllTasksComponent werden alle aktuellen Tasks dargestellt, welche vom Server generiert wurden. Hierbei wird für jeden Task ein button generiert der mit der ID des Tasks versehen wird. Je nachdem ob der Task gelöst wurde, wird entweder ein roter Button (nicht gelöst) oder ein grüner Button (gelöst) generiert. Die Komponente fragt je nach Änderung der im Store gespeicherten Task-ID den **/generated-tasks** Endpunkt ab und erhält ein JSON der aktuell generierten Tasks vom Ser-

ver.

Methode um generierte Tasks abzufragen

```
getGeneratedTasks() {  
  axios.get("http://localhost:8000/generated-tasks")  
    .then((res) => {  
      this.generatedTasks = res.data  
    }).catch((error) => {  
      store.error = error.data  
    });  
}
```

— Javascript

2.2.5.2.4. TaskSentenceComponent

In dieser Komponente werden alle, im Store als Liste gespeicherten Sätze, in der UI angezeigt. Um diese Funktionalität zu erreichen, wird eine *computed* Methode verwendet, deren Rückgabewert in einer Schleife alle Sätze, mit zusätzlichem Leerzeichen, ausgibt.

Computed Methode

```
computed: {  
  sentences() {  
    return store.sentences;  
  }  
}
```

— Javascript

Iterieren über die Liste der Sätze

```
<template v-for="sentence in sentences">  
  {{ sentence }}{{ " " }}  
</template>
```

— HTML

2.2.5.2.5. CostumizeTaskComponent

Um die Parameter der generierten Aufgaben zu beeinflussen, wird der CostumizeTaskComponent verwendet. Basis für die Parameter bilden die Vuejs data values, welche für jeden Parameter vorhanden sind. Diese Daten werden über das *v-model* Binding auf die Input Felder gemapt. Bei den *needed* wird dynamisch eine Gruppe an Checkboxes erstellt, welche dann auf die Needed Liste in den Daten gemapt wird.

Daten der Parameter

```
data() {  
  return {  
    allCases: [],  
    difficultyValue: 3,  
    amountValue: 8,  
    needed: []  
  }  
}
```

— Javascript

Wird die Komponente gemounted, wird der Endpunkt angesprochen, welcher alle verfügbaren Case Typen angibt, um eine Liste bereitzustellen, aus der der Nutzer die gewünscht enthaltenen Cases auswählen kann.

Holen der Verfügbaren Cases

```
getCasesToChoose: function () {  
  const url = "http://localhost:8000/cases-to-choose";  
  axios.get(url).then((res) => {  
    this.allCases = res.data;  
  });  
}
```

— Javascript

Sind die Parameter festgelegt, kann der Nutzer die Aufgabe über einen Button anfragen. Beim Click wird zuerst der Query String aufgebaut, wobei vorhandene Parameter in ein JSON Object übertragen werden. Danach werden die einzelnen Parameter auf ihren Datentyp geprüft und ob sie gesetzt sind, um sie in einen Querystring für den Request zu übertragen.

Übertragen der Daten und Bauen des Query Strings

```
buildURL() {  
  const params = {  
    difficulty: this.difficultyValue,  
    amount: this.amountValue,  
    needed: this.needed  
  }  
  const queryString = Object.keys(params).map(key =>  
(this.isVariableAndNotEmpty(params[key]) ? `${key}=${params[key]}` :  
null)).filter(Boolean).join('&');  
  const url = `http://localhost:8000/get-task?${queryString}`;  
  return url;  
}
```

— Javascript

Mit diesem Querystring wird der Request gegen den Endpunkt geschickt und die, nach den Parametern generierte Aufgabe wird empfangen und ihr Inhalt den korrekten Variablen zugewiesen.

Gesamte Methode zur Anfrage eines Tasks

```
async getTask() {
  const url = this.buildURL();
  console.log(url)
  await axios.get(url)
    .then((res) => {
      store.sentences = res.data.sentences;
      store.task_id = res.data.id;
    })
    .catch((error) => {
      console.log(error);
    });
}
```

— Javascript

2.2.5.3. UserSolutionComponent

Der UserSolutionComponent ist die Komponente mit der meisten Logik, da der Nutzer in dieser die einzelnen Aufgaben löst. Grundlegend soll der Nutzer die einzelnen Sachverhalte aus der Textaufgabe herauslesen und den korrekten Sachverhaltstypen, sowie die richtige Summe und Gesetzesgrundlage zuweisen, aber auch am Ende das korrekte zu steuernde Einkommen berechnen. Um Flexibilität und mehr Komplexität in die Lösung der Aufgabe zu bringen, kann der Nutzer einzelne Reihen der Lösung frei hinzufügen oder löschen (Je Funktion für das Hinzufügen (**addRow**) bzw. Löschen (**deleteRow**)). Um nicht zu viele Reihen zuzulassen, ist die maximale Anzahl an Reihen, die Anzahl der Sachverhalte in der Aufgabe. Überprüft wird dies durch die *checkIfRowNecessary* Methode, bei der abgeglichen wird, ob die Anzahl der Reihen die Anzahl der möglichen Optionen, also die Anzahl der Sachverhalte, überschreiten würde.

Datenstruktur der Reihen

```
rows: [
  {
    'id': 0, 'select': "Sachverhalt auswählen", "_law": '',
    "num": null
  }
]
```

— Javascript

Hierfür besteht die Datenstruktur der Reihen aus einer Liste von JSON Objekten welche mit *v-for* iterativ ausgelesen werden, wobei auf jedes Element ein einzelnes Input Element mit v-model gemapt wird. Die Input Felder erhalten zur eindeutigen Identifikation eine dynamische ID durch das HTML id-Attribut, mit der Syntax: rowID + "_case_name" bzw. "_law" oder "_num". Durch diese dynamische Zuweisung kann jeder Wert jeder Reihe eindeutig identifiziert und zur Reihe zugeord-

net werden.

Mapping mit v-for

```
<div class="row form-row" v-for="row in rows">
  <div class="col col-xs-12">
    <div class="row mb-3">
      <div class="col">
        <select :name="row.id + '_case_name'" :id="row.id + '_case_name'"
class="form-control"
          v-model="row.select">
          <option selected disabled>Sachverhalt auswählen</option>
          <option :value="opt.name" v-for="opt in options">{{ opt.name
        }}
      </option>
    </select>
  </div>
  <div class="col">
    <input type="text" class="form-control" :id="row.id + '_law'"
      placeholder="Gesetzesgrundlage" v-model="row.law"
    :name="row.id + '_law'">
  </div>
  <div class="col">
    <input type="number" class="form-control" :id="row.id + '_num'"
    placeholder="Summe"
      v-model="row.num" :name="row.id + '_num'">
  </div>
  <div class="col">
    <button class="btn btn-danger" :id="row.id + '_del'"
      @click="this.deleteRow(row)">Löschen</button>
  </div>
</div>
</div>
```

— HTML

Ist eine weitere Reihe notwendig ?

```
checkIfRowNecessary: function () {
  let maxRows = this.showMaxRows > 0 ? this.showMaxRows : null;
  return maxRows > this.rows.length ? true : false;
}
```

— Javascript

Weitere Datenattribute sind die korrekt gelösten Reihen (**correct**), welche aus dem Ergebnis des Lösungsrequest ausgelesen werden. Dabei wird angegeben welche Spalte welcher Reihe korrekt oder nicht korrekt ist. Dabei werden je nach Änderung des **correct** Wertes die Korrektheit der Rei-

hen geprüft. Um die Korrektheit zu prüfen, wird die Methode *evaluateCorrectnessOfRow* aufgerufen. Sie prüft für jede Reihe aus dem **correct** Feld ob die einzelnen Spalten richtig sind. Falls sie richtig sind, wird die korrekte Spalte grün umrandet (Bootstrap Klasse) und deaktiviert, sodass sie nicht nachträglich zum falschen geändert werden kann. Außerdem gibt es weiterhin das Attribut **allSolved**, welches bei jeder Lösung aus der JSON Response des Servers ausliest ob alle Aufgaben korrekt gelöst wurden. Wenn ja, wird eine entsprechende Nachricht angezeigt. Für das zu steuernde Einkommen, gibt es ein Attribut für seinen Wert (**zveValue**) und für seine Korrektheit (**zve**). Der **zveValue** wird für das Input Mapping in Vuejs verwendet, der **zve** wird verwendet um zu prüfen ob das **zve** korrekt angegeben wurde. Soll eine Aufgabe gelöst werden, kann dies über einen Button "Aufgabe Lösen" durchgeführt werden. Wird der Button geklickt, werden alle Reihen als JSON Payload eines POST-Request an den Server zur Kontrolle geschickt. Ergebnis wird einerseits dem **correct**, für die einzelnen Reihen und deren korrekte Werte, sowie **allSolved**, für den Check ob alle Aufgaben gelöst wurden, Attribut zugewiesen.

Reihe Lösen

```
solveTask() {
  const url = 'http://localhost:8000/solve/' + store.task_id
  const data = JSON.stringify(this.rows);
  axios.post(url, data, { headers: { 'Content-Type': 'application/json' } })
    .then((res) => {
      this.correct = res.data.given
      this.allSolved = res.data.all_solved
    }).catch((error) => {
      store.error = error.response.data.detail;
    });
}
```

— Javascript

Auch diese Komponente greift auf den Store zu, um zu prüfen ob ein neuer Task vorliegt. Liegt ein neuer Task vor, wird die *reset* Methode ausgeführt, welche alle Daten in der Komponente, sowie alle Klassen / Disabled Attribute der Input Elemente zurücksetzt (falls sie korrekt oder falsch waren).

Reset Methode

```
reset() {
  this.options = [""],
  this.rows = [
    {
      'id': 0, 'select': "Sachverhalt auswählen", "law": '',
      "num": null
    }
  ]
  for(let row of this.rows) {
    document.getElementById(row.id + "_case_name").className = "form-control"
    document.getElementById(row.id + "_law").className = "form-control"
    document.getElementById(row.id + "_num").className = "form-control"
  }
}
```

```
document.getElementById("zve").className = "form-control"
```

```
}  
this.correct = {}  
this.allSolved = false  
this.zve = false  
this.zveValue = null  
}
```

— Javascript

2.2.5.3.1. GivenSolutionComponent

Der GivenSolutionComponent stellt für die aktuelle Aufgabe die komplette Lösung bereit. Diese wird in Form einer Tabelle dargestellt, bei der die Zuordnung der rechtlichen Grundlage, sowie der Summe zum Sachverhalt erfolgt. Die Komponente hält nur zwei Attribute, das zu versteuernde Einkommen sowie die Lösungen.

Daten des GivenSolutionComponent

```
data() {  
  return {  
    solutions: [],  
    zve: null  
  }  
}
```

— Javascript

Um die Lösung beim Server anzufragen, wird beim Klicken auf den "Lösung anzeigen" Button, die Lösung über eine Request angefragt und darauf als Tabelle dargestellt. Die notwendige ID der aktuellen Aufgabe für den Request, wird dabei aus dem Store entnommen. Im gleichen Atemzug fragt die Komponente ebenfalls das zu versteuernde Einkommen der Aufgabe ab.

Methode zum Abfragen der Lösung

```
getSolution() {  
  if (store.task_id !== null) {  
    const url = "http://localhost:8000/";  
    axios.get(url + "solution/" + store.task_id).then((res) => {  
      this.solutions = res.data;  
      console.log(this.solutions)  
    }).catch((error) => {  
      store.error = error  
    });  
    axios.get(url + "zve/" + store.task_id).then((res) => {  
      this.zve = res.data;  
    }).catch((error) => {
```

```

        store.error = error
      })
    }
  }
}

```

— Javascript

Die Lösungen werden, wie im Codeausschnitt zu erkennen, in die Attribute gespeichert. Beide Attribute werden darauffolgend über Templates im HTML ausgegeben.

Ausgabe der Lösung als Tabelle

```

<template v-if="solutions !== []">
  <template v-for="solution in this.solutions">
    <tr>
      <td>
        {{ solution.case_name }}
      </td>
      <td>
        {{ solution.law }}
      </td>
      <td>
        {{ solution.number }}
      </td>
    </tr>
  </template>
</template>
<tr>
  <td>zvE</td>
  <td></td>
  <td>{{ zve }}</td>
</tr>

```

— HTML

2.2.5.3.2. ErrorComponent

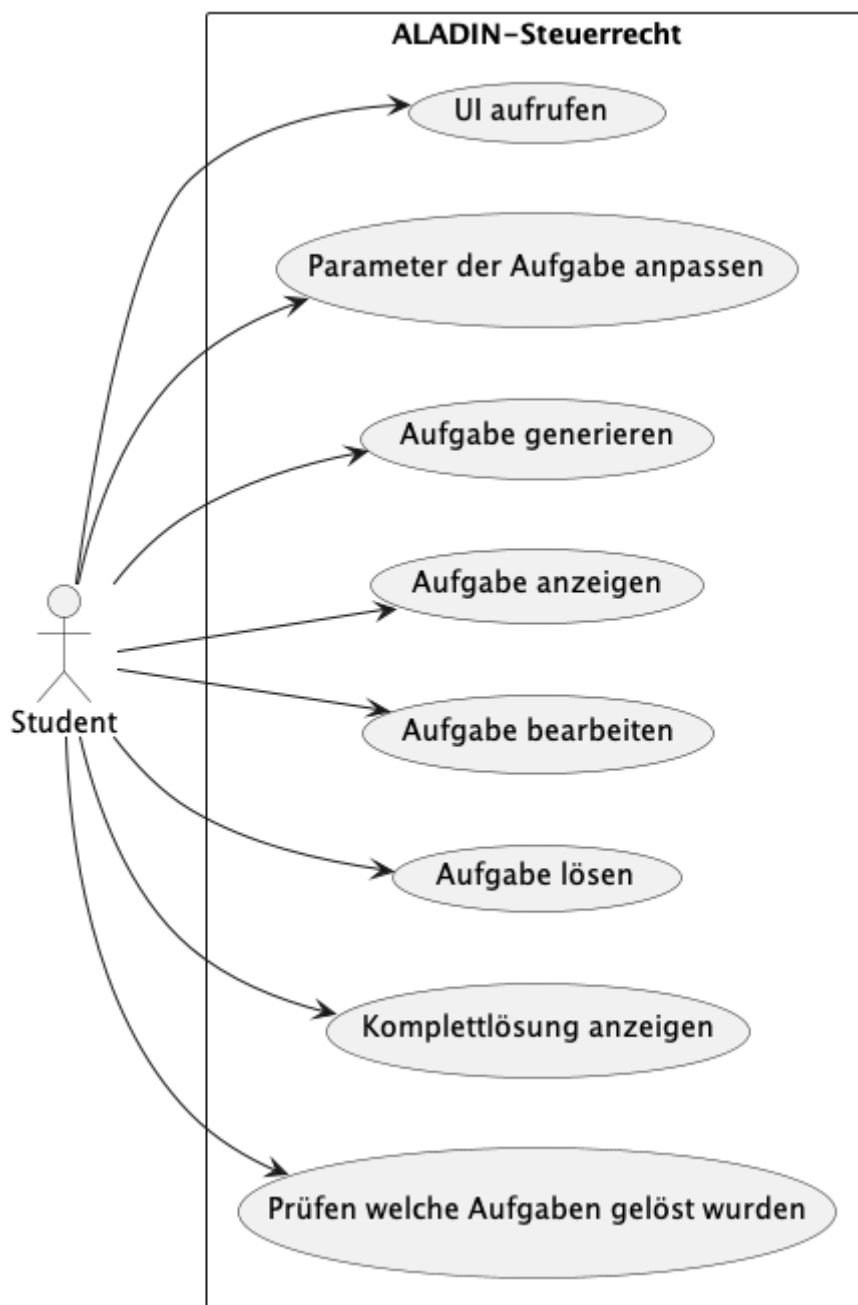
Beim ErrorComponent wird lediglich ein Error im unteren Bereich der Seite angegeben. Dies wird realisiert, indem die Komponente auf den im Store gespeicherten Error bei Änderung zugreift und diese einfach in einer Row ausgibt. Als Attribute besitzt die Komponente hier nur den Store, um nach Änderung des Errors im Store den Error auszugeben.

2.3. Ergebnisse

Um das Erreichte noch einmal zusammenfassend darzustellen, erlautere ich, welche Use-Cases von der Software abgedeckt werden und wie der finale Stand der UI ist.

2.3.1. Use-Cases

Folgende Use-Cases können mit der Software bedient werden:



1. UI aufrufen: Der Student kann die UI aufrufen und die Oberflächenelemente betrachten
2. Parameter der Aufgabe anpassen: Der Student kann, durch die UI Elemente, die Parameter für die gewünschte Aufgabe anpassen
3. Aufgabe generieren: Der Student kann sich nach seinen Wünschen eine Aufgabe generieren lassen
4. Aufgabe anzeigen: Der Student kann sich die, nach seinen Wünschen generierte, Aufgabe anzeigen lassen
5. Aufgabe bearbeiten: Der Student kann die Aufgabe mit den Lösungsreihen abarbeiten
6. Aufgabe lösen: Der Student kann seine Lösung prüfen lassen und eventuell korregieren

7. Komplettlösung anzeigen: Der Student kann sich die vollständige Lösung zur Aufgabe anzeigen lassen
8. Prüfen welche Aufgaben gelöst wurden: Der Student sieht in der UI welche Aufgaben er bereits gelöst hat, diese können noch **nicht** erneut aufgerufen werden

Die grundlegenden Funktionalitäten zur Bearbeitung der Aufgaben sind dadurch abgedeckt und auf ihnen kann aufgebaut werden. Der Student ist also in der Lage, Aufgaben nach seinen Wünschen zu erstellen und zu lösen, sowie seine eigene Lösung zu prüfen.

2.3.2. Stand der UI

Das User Interface beinhaltet am Ende des Projektes alle vorherigen Komponenten. Beim Farbschema habe ich mich an das dunkle Theme von Bootstrap gehalten und auch die Buttons eher subtil eingefärbt. Die einzelnen Komponenten lassen sich, mit dem oben dargestellten Konzept, leicht in der UI erkennen.

finaler Stand der UI

2.4. Ausblick

Durch den zeitlich begrenzten Rahmen des Projektes, gibt es prospektiv noch Verbesserungsmöglichkeiten für die ALADIN Lösung im Steuerrecht. Der jetzige Stand der Software bietet eine gute Grundlage auf der nachfolgend aufgebaut werden kann. Ausgewählte Möglichkeiten möchte ich nachfolgend darstellen.

2.4.1. Hinweise zu den Aufgaben

Es gibt bereits in den Datenstrukturen der Lösungsklasse (Solution.py) ein Attribut, in welchem Hinweise hinzugefügt werden können. Um dem Studenten, auf Wunsch, die Aufgabe zu erleichtern, ist es sinnvoll, einen Button um Hinweise einzublenden hinzuzufügen. Auch ein Einblenden nach beispielweise 3-4 nicht erfolgreichen Lösungsversuchen wäre sinnvoll.

2.4.2. Optimieren des Sprachmodels

Wie zu Anfang beschrieben, wird, um die Sätze grammatikalisch unterschiedlich zu gestalten, ein Sprachmodel von Huggingface verwendet. Dieses Sprachmodel wurde jedoch nicht auf steuerliche Sachverhalte trainiert. Diese erfordern nicht zwangsweise spezielle grammatikalische Konstruktion, jedoch sind die Trennwörter des Sprachmodels nicht immer korrekt.

durch Sprachmodel gefüllte Masken

Er ist Angestellter und erhält 6700€. Hier kauft Angestellter Fahrtkosten von 1150€ ab. Als Kapitalbeteiligung erhält P 29000€. Hier meldet Angestellter Heizkosten von 1150€ an. Dabei setzt Angestellter Abschreibungen von 1150€ an. Von die Dividende erhält er 64000€.

Je nach Aufwand, kann sich ein eigenes trainiertes Model lohnen, um die Qualität der Sätze zu erhöhen.

2.4.3. Neue Sachverhalte

Zusätzlich lassen sich auch neue Sachverhalte hinzufügen und definieren. Das Steuerrecht bietet hier eine enorm große Anzahl an Möglichkeiten, auch beispielweise stark zusammenhängende Sachverhalte, welche aufeinander aufbauen. Auch spezielle Sachverhalte wie das Teileinkünfteverfahren oder Ehegattensplitting sind denkbar.

2.4.4. Neue Aufgabentypen

Um die Übungsmöglichkeiten zu steigern, können auch weitere Aufgabentypen definiert werden. Typische Aufgaben aus dem Steuerrecht umfassen ebenfalls:

- Zurodnung von Einkunftsarten
- Rechnen von Besteuerungsmöglichkeiten
- Vor- / Nachteile von Besteuerungsmöglichkeiten
- Spezielle Textaufgaben zu spezieller Besteuerung
- Bilanzierung

Hierbei ist es möglich, sich von den klassischen Textaufgaben loszulösen.

2.4.5. Upload einer Konfiguration

Grundlegend ist eine Komponente im Frontend enthalten, welche in Zukunft den Upload einer eigenen Konfiguration an Gesetzen sowie Sachverhalten ermöglichen kann. Dazu fehlt noch der Upload der Konfiguration auf den Server / das Backend, sowie das folgende Laden der Aufgaben in die Auswahl.

Abschnitt 3. Generierung Ereignisgesteuerte Prozessketten

3.1. Aufgabenbeschreibung

Diese Aufgabe zielt darauf ab, den Studierenden ein Verständnis für die Modellierung von ereignisgesteuerten Prozessketten (EPK) zu vermitteln. Dabei geht es um die praktische Anwendung von EPKs, um komplexe Geschäftsprozesse grafisch darzustellen, Fehler zu erkennen und einen Überblick über Abhängigkeiten und Beziehungen zwischen den verschiedenen Prozessschritten zu geben. Die Studierenden sollen lernen, wie EPKs erstellt werden, die den Anforderungen eines bestimmten Geschäftsprozesses entspricht. Hierbei sollten die Studierenden auch die verschiedenen Arten von Gates und EPK-Elementen verstehen, die in einer EPK verwendet werden können.

Der Entwickler hatte die Aufgabe EPKs zufällig zu generieren unter Einhaltung von bestimmten Parametern und Einhaltung von syntaktischer Korrektheit. Im weiteren Verlauf des Projektes sollte dann eine Oberfläche erstellt werden, die es den Benutzern ermöglicht, EPKs einfach zu erstellen und zu bearbeiten. Darüber hinaus sollten verschiedene Aufgabentypen entworfen werden, um den Studierenden zu helfen, ihr Wissen und ihre Fähigkeiten bei der Modellierung von EPKs zu verbessern.

3.2. Planung

Um sicherzustellen, dass das Projekt effektiv und erfolgreich umgesetzt wurde, wurde ein regelmäßiges Kommunikations- und Planungssystem mit dem Kunden eingerichtet. Einmal in der Woche fanden Treffen statt, bei denen der Fortschritt des Projekts, erreichte Ziele, Probleme und Unklarheiten besprochen wurden. In diesen Treffen wurden auch die nächsten Aufgabenpakete beschlossen, um sicherzustellen, dass das Projekt im Zeitplan blieb. Darüber hinaus arbeitete der Entwickler zweimal pro Woche gezielt an den in der Planung festgelegten Wochenzielen. Diese Vorgehensweise half dabei, den Entwicklungsprozess auf Kurs zu halten und sicherzustellen, dass das Projekt termingerecht abgeschlossen wurde. Der Entwickler nutzte Java als Programmiersprache, da er hierin am erfahrensten war und Java eine breite Unterstützung durch die Community und eine umfangreiche Sammlung von Bibliotheken und Frameworks bietet. Als Entwicklungsumgebung wurde JetBrains IntelliJ verwendet, die sich durch eine intuitive Oberfläche, intelligente Codevervollständigung und integrierte Versionskontrollsysteme auszeichnet. Die Verwendung von IntelliJ hat dem Entwickler geholfen, schneller und effizienter zu arbeiten und den Entwicklungsprozess zu beschleunigen. Das Versionskontrollsystem Github wurde verwendet, um sicherzustellen, dass das Projekt effektiv und sicher verwaltet wurde. Das Repository für das Projekt ist auf <https://github.com/TinkerTee69/EPC-Generator> zu finden. Das Projekt benötigt lediglich ein aktuelles Java SDK (getestet wurde mit openjdk 19.0.1), was die Einrichtung der Umgebung einfach und unkompliziert macht.

3.3. Umsetzung

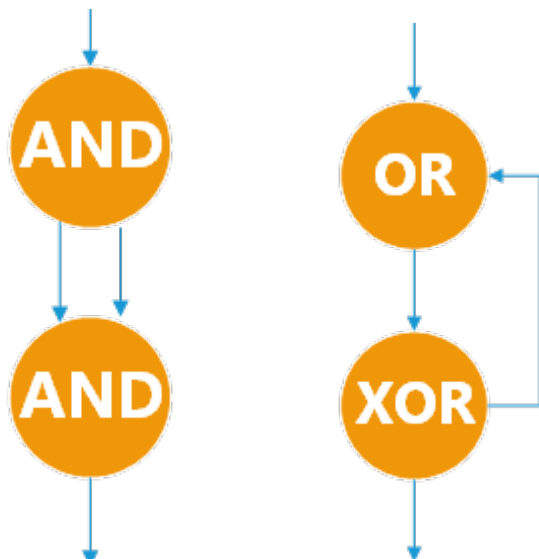
Die Umsetzung des Projekts basierte auf einem sorgfältig entwickelten Programmablaufplan und einem Klassendiagramm, welche es dem Entwickler ermöglichte, den Algorithmus Schritt für

Schritt zu entwickeln und nachzuvollziehen (siehe <https://github.com/TinkerTee69/EPC-Generator/blob/master/PAP.vsdx> und <https://github.com/TinkerTee69/EPC-Generator/blob/master/Klassendiagramm.vsdx>). Hierbei wurden die Regeln von Ereignisgesteuerten Prozessketten (EPKs) berücksichtigt. Die EPKs legen fest, wie Geschäftsprozesse modelliert werden sollten, um eine klare und verständliche Darstellung zu ermöglichen. Falls der Leser nicht mit den Regeln der EPKs vertraut ist, kann er Informationen dazu auf https://www.netzwerk-welt.de/common_files/BWL/EPK.pdf finden.

Im Folgenden wird der Entwickler seinen Quellcode erläutern und die Schritte des Algorithmus detailliert erläutern, um eine klare und verständliche Darstellung der Funktionsweise des Programms zu ermöglichen.

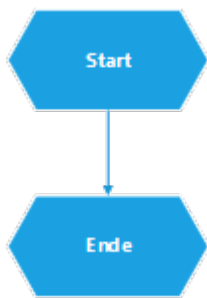
Zu Beginn des Programms werden die Parameter für die Generierung der EPKs gesetzt. Hierbei wird zunächst überprüft, ob der Testmodus aktiviert ist, was in der Parameterklasse Parameter.java definiert wird. Wenn der Testmodus aktiviert ist, können die Eingaben übersprungen und stattdessen direkt mit den gewünschten Parametern gearbeitet werden. Die verfügbaren Parameter für die Generierung der EPKs umfassen die minimale und maximale Anzahl von Elementen zwischen den Gates. Hierbei wird überprüft, ob der Wert für den maximalen Parameter größer ist als der für den minimalen Parameter. Sollte dies nicht der Fall sein, wird der Benutzer aufgefordert, gültige Eingaben zu tätigen, bis verwendbare Werte eingegeben wurden. Darüber hinaus kann die Anzahl von Rhomben und Loops, die in der EPK enthalten sein sollen, bestimmt werden. Rhomben beeinhalteten zwei Gates gleicher Art mit zwei Vorwärtskanten, die das Startgate mit dem Endgate verbinden. Die Gates können dabei als AND, OR oder XOR definiert werden. Loops hingegen haben eine Vorwärts- und eine Rückwärtskante und starten mit einem OR Gate und enden mit einem XOR Gate. Dabei können allein BackwardKanten in Richtung Prozessstart zeigen, was bedeutet, dass sie zu einem vorherigen Prozessschritt zurückgehen.

Links Rhombus, rechts Loop

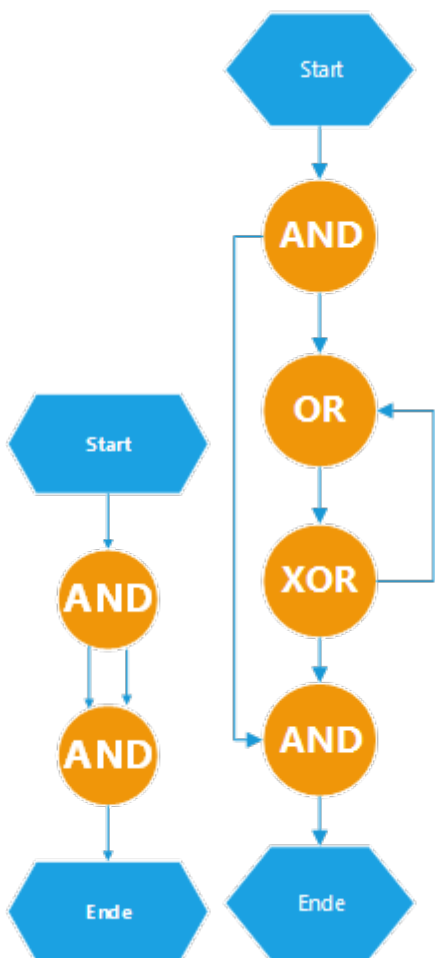


Insgesamt ermöglichen die definierten Parameter eine große Flexibilität bei der Generierung der EPKs und erlauben es dem Anwender, eine Vielzahl von unterschiedlichen Prozessmodellen zu erstellen, um die Übungen individuell anzupassen.

Nachdem die Parameter für die Generierung der EPKs festgelegt wurden, geht es in der Klasse EPK.java mit der Erstellung der ersten Verbindung los:

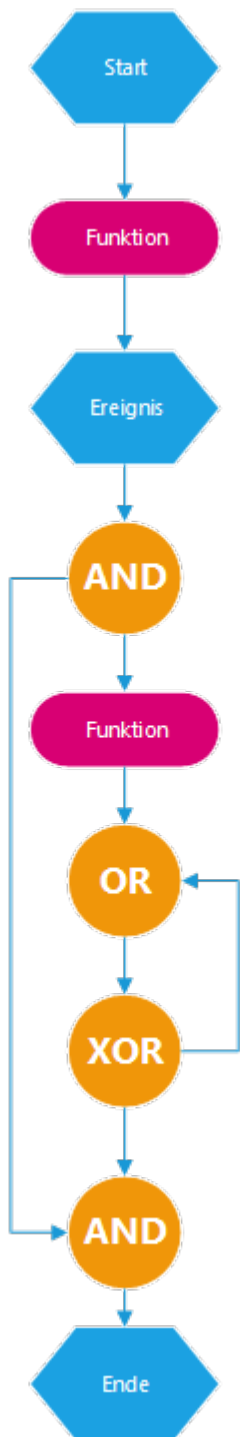


Hierfür werden zwei Ereignisse, das Start- und Endereignis, erstellt und mit einer Vorwärtskante verbunden. Vorwärtskanten können nur in Richtung Prozessende zeigen, daher werden nur diese verwendet, um das EPK zu füllen. Es wird eine Liste, kantenList genannt, erstellt, in der alle Vorwärtskanten aufgelistet sind. Aus dieser Liste wird dann zufällig eine Vorwärtskante ausgewählt, die im nächsten Einfügevorgang mit einem Rhombus oder Loop ergänzt wird.



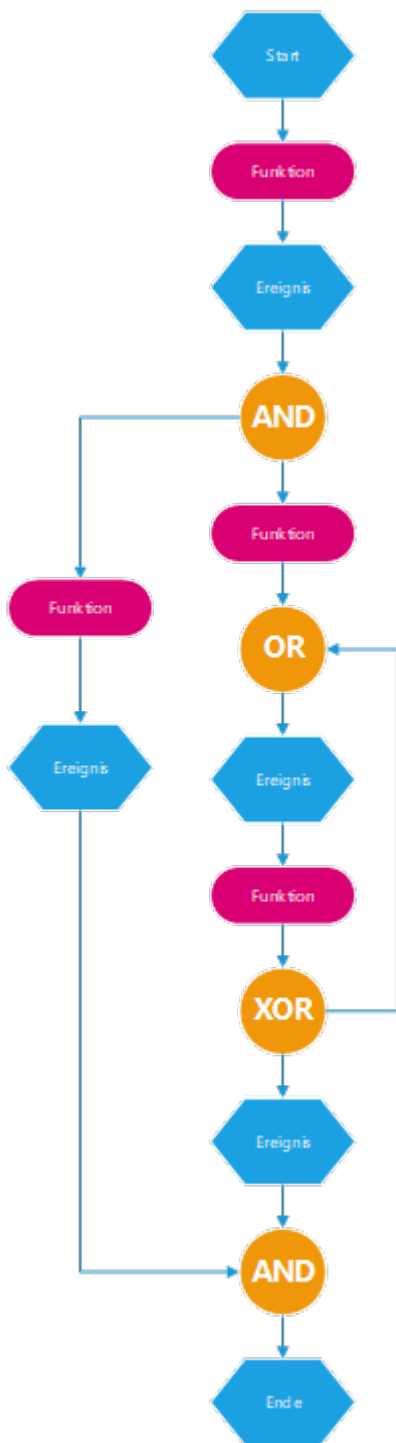
Rhomben und Loops enthalten jeweils ein Start- und Endgate, die mit zwei Kanten verbunden sind. Die ausgewählte Vorwärtskante wird dann der kantenList hinzugefügt, damit sie bei der zufälligen Auswahl der Kante (*verwirklicht durch die Methode rndKante()*) im nächsten Einfügevorgang berücksichtigt wird. Auf diese Weise wird das EPK Stück für Stück aufgebaut, indem nach und nach neue Kanten und Gates hinzugefügt werden. Durch die Verwendung einer zufälligen Auswahl der Kanten und Gates sowie die Berücksichtigung der bereits vorhandenen Vorwärtskanten wird eine hohe Varianz und Abwechslung in den generierten EPKs erreicht. Nachdem alle Rhomben und Loops in das EPK eingefügt wurden, geht es im nächsten Schritt darum, die Elemente zwischen den Loops einzufügen. Hierfür wird die Methode *checkElements()* aufgerufen. Diese Methode ruft dann die entsprechenden Methoden zum Einfügen und Überprüfen der Elemente auf. Da Änderungen an einer Methode Auswirkungen auf das gesamte EPK-Modell haben kann, müssen die Methoden zur Überprüfung der EPK nach jeder Änderung erneut aufgeru-

fen werden, um sicherzustellen, dass die Bedingungen erfüllt werden.



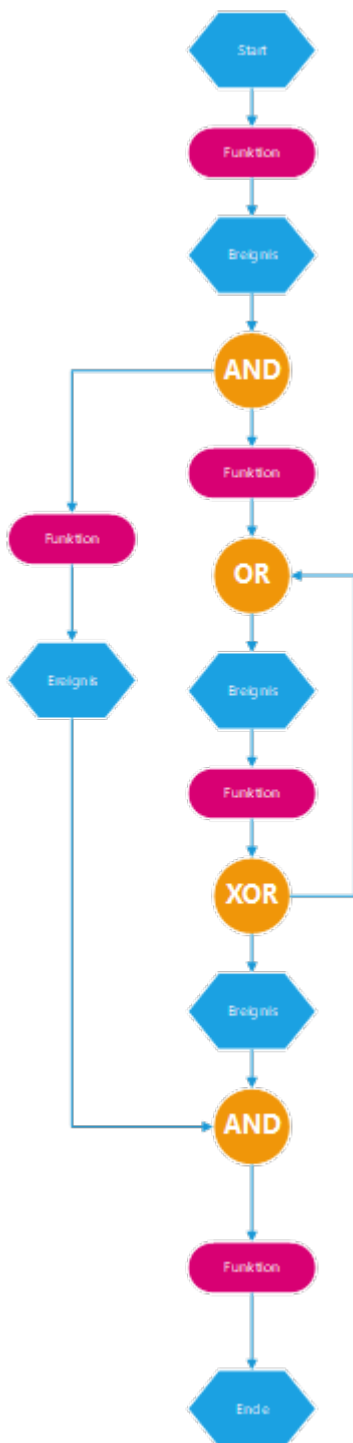
Die check-Methoden, die für das Überprüfen der EPK auf Korrektheit zuständig sind, geben einen boolschen Wert zurück. Falls es zu einer Veränderung kommt, also wenn beispielsweise ein Element eingefügt oder entfernt wurde, wird das `changeFlag` auf "True" gesetzt. Daraufhin werden die Methoden erneut aufgerufen, bis alle Überprüfungen erfolgreich durchlaufen wurden und das EPK den definierten Bedingungen entspricht. Zunächst werden die Elemente vor den öffnenden Gates eingefügt, was in der Methode `checkElementsBeforeStartgate()` durchgeführt wird. Hierbei wird die Liste "list" durchlaufen, in der alle Gates, EPK-Elemente und Kanten aufgeführt sind. In der Schleife wird nach Elementen der Art Rhombus oder Loop gesucht. Wenn eine solche gefunden wird, wird nach der Kante gesucht, die den Rhombus oder Loop als Ende referenziert. Sobald die entsprechende Kante gefunden wurde, wird die Startreferenz untersucht. Wenn diese ein Gate enthält, ist der Fall eingetreten, dass die Kante zwei Gates miteinander verbindet und daher keine Elemente zwischen diesen beiden Gates vorhanden sind. Dies muss nun geändert werden. Um dies zu erreichen, wird eine zufällige Anzahl von Elementen zwischen den beiden

Gates eingefügt, wobei die Anzahl zwischen dem min- und max-Parameter zufällig ausgewählt wird. Die Elemente werden von unten nach oben, also in Richtung Startprozess, eingefügt. Durch diese Methode wird gewährleistet, dass alle notwendigen Elemente im EPK vor den öffnenden Gates (Startgates) vorhanden sind. Im nächsten Schritt geht es darum, die Funktionen und Ereignisse abwechselnd zwischen den Gates zu platzieren. Hierbei wird zufällig mit einer Funktion oder einem Ereignis begonnen, es sei denn, es handelt sich um einen XOORhombus oder LOOP, bei dem das Startgate ein (X)OR beinhaltet und kein Ereignis davor stattfinden darf. In diesem Fall wird rndStartElement auf den Wert 2 gesetzt, was bedeutet, dass die Generierung mit einer Funktion beginnen muss. Diese Vorgehensweise gewährleistet, dass das EPK-Modell den definierten Bedingungen entspricht und dass der Prozessablauf klar und verständlich dargestellt wird. Durch die zufällige Platzierung der Funktionen und Ereignisse wird zudem sichergestellt, dass die Anzahl unterschiedlicher Prozessmodelle, die generiert werden kann, erhöht wird. Durch das Einfügen der Elemente und die zufällige Platzierung von Funktionen und Ereignissen kann es vorkommen, dass Änderungen am EPK-Modell vorgenommen werden müssen. Sobald eine Änderung vorgenommen wird, wird das Flag withoutChange auf "false" gesetzt, um anzuzeigen, dass es Änderungen gab. Solange es zu Änderungen kommt, wird ständig überprüft, ob sich vor einem Startgate Elemente befinden. Hierbei wird geprüft, ob ein Element direkt vor dem Startgate vorhanden ist, oder ob der Abstand zum nächsten Element kleiner ist als die minimale Anzahl von Elementen zwischen den Gates, die zuvor festgelegt wurde. Falls dies der Fall ist, wird ein Element eingefügt, um sicherzustellen, dass das EPK-Modell syntaktisch korrekt ist. Sobald alle notwendigen Elemente eingefügt wurden und es keine weiteren Änderungen mehr gibt, wird das changeFlag zurückgegeben. Dieses ist auf "true" gesetzt, wenn es zu mindestens einer Änderung gekommen ist. Dadurch müssen alle anderen Überprüfungen mindestens einmal durchgeführt werden. Erst wenn alle Überprüfungen erfolgreich durchgeführt wurden, handelt es sich um ein syntaktisch korrektes EPK



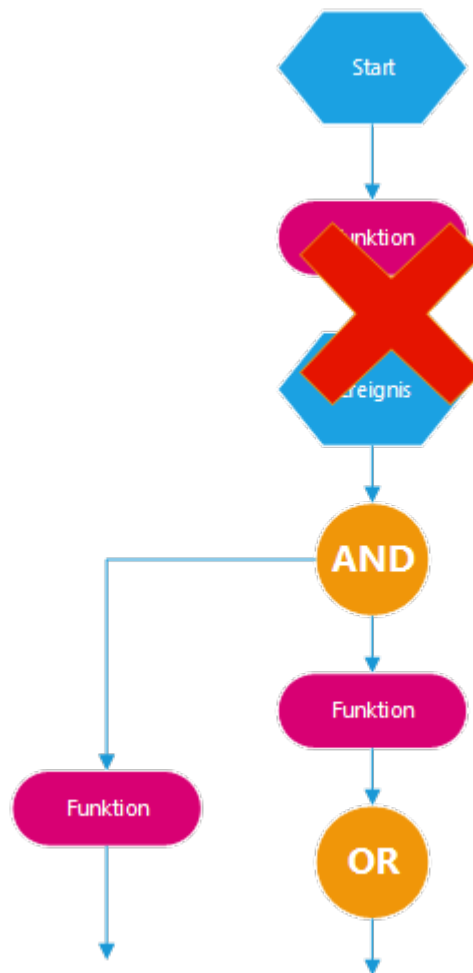
Als nächstes werden die Elemente nach dem Startgate eingefügt / überprüft in *checkElementsBeforeEndGate()*. Hierbei wird ähnlich vorgegangen wie bei der vorherigen Methode, nur das in dieser Methode das Endgate eines Rhombus / Loops gesucht wird. Auch hier werden dann Elemente in der Anzahl innerhalb des Parameterintervalls eingefügt.

Die Generierung von EPK-Modellen erfolgt durch das zufällige Einfügen von Funktionen und Ereignissen. Allerdings kann es vorkommen, dass auch ein Ereignis vor einem (X)OR-Ereignis eingefügt wird. Um sicherzustellen, dass das EPK-Modell syntaktisch korrekt ist, gibt es die *checkXOOR-Methode()*. In dieser Methode wird die Liste "list" durchlaufen und nach einer Vorwärtskante gesucht, die als Ende ein (X)OR-Gate und als Start ein Ereignis referenziert. Wenn dies der Fall ist, wird überprüft, ob eine Funktion zwischen dem Ereignis eingefügt werden kann (d.h. der max-Parameter wurde noch nicht ausgeschöpft), oder ob das Ereignis gelöscht werden muss (max-Parameter würde ansonsten überschritten werden) oder ob aus dem Ereignis eine Funktion gemacht werden muss (min = max-Parameter).



Weitere Methoden wie `checkAND` und `checkBeforeAndAfterGate` dienen der Überprüfung, ob vor und nach einem Gate der gleiche Typ eines EPK-Elements steht und ob hinzugefügt oder gelöscht werden muss.

Die `checkMinMax`-Methode wird verwendet, um sicherzustellen, dass die `min`- und `max`-Parameter eingehalten werden. Wenn dies nicht der Fall ist, werden Elemente hinzugefügt oder gelöscht (in diesem Beispiel gehen wir von `min:1` und `max: 2` Elementen zwischen den Gates aus). Um die Code-Übersichtlichkeit zu verbessern, wurden einige häufig verwendete Routinen in Funktionen ausgelagert. Diese Funktionen tun genau das, was der Name sagt. So gibt beispielsweise die Funktion `"getPreviousKantenIndex"` den Index der vorherigen Kante zurück, während die Funktion `"getElementToDelete"` das zu löschende Element sucht und das vorherige Element mit dem Element nach dem zu Löschenden verbindet. Die Funktion `"deleteElement"` löscht das Element und passt die Referenz an, um auf kein nicht existierendes Element zu zeigen. `"countElementsForward"` zählt die Elemente von dem angegebenen Kantenindex bis zur Kante, die ein Gate als Ende referenziert, und `"add2list"`



fügt die Elemente in die Listen hinzu.

Nachdem ein EPK erstellt wurde, müssen die dazugehörigen Texte eingefügt werden. Hierfür gibt es die Klasse "insertText.java", die drei String-Listen enthält: infinitiv, partizip und substantiv. Zunächst werden die Elemente vom Startereignis bis zum ersten Gate mit der Methode "fillStart()" befüllt. Da das Start- und Endereignis beim Generieren des EPKs bereits erstellt wurden und die IDs 1 und 2 haben, kann das Startereignis leicht gefunden werden. Anschließend werden alle Elemente bis zum ersten Gate mit der Methode "fillText()" befüllt. In der "fillText()" Methode werden die Elemente mit zufälligen Texten ergänzt, je nachdem, ob es sich um ein Ereignis oder eine Funktion handelt. Funktionen erhalten ein Substantiv + Infinitiv, während Ereignisse ein Substantiv + Partizip bekommen. Innerhalb eines Subprozesses wird das Substantiv beibehalten, um eine konsistente Textstruktur zu gewährleisten.

Elemente werden nur befüllt, wenn diese noch keine Texte enthalten (Position == null; Position kann verwendet werden um die Reihenfolge eines Elementes innerhalb eines Subprozesses zu erfahren). Die genutzten Wörter werden anschließend aus der jeweiligen Wörterliste entfernt. Nach dem füllen der Elemente zwischen Start und erstem Gate, werden die Elemente zwischen den Gates befüllt („fillBetweenGates()“). Hierbei wird in der Liste „list“ nach einem Rhombus oder Loop gesucht. Wenn eines gefunden wurde, wird die Kante gesucht, die das Startgate referenziert. Es wird dann wiederum alle Elemente befüllt, bis ein Gate in einer Kante als Endreferenz auftritt. Dann wird der nächste Rhombus oder Loop in der Liste „list“ gesucht. Wenn alle Elemente zwischen den Gates befüllt wurden, werden die Elemente zwischen dem Endgate und dem nächsten Startgate befüllt (Methode „fillAfterEndgate“), oder bis das Endereignis (mit der ID 2) gefunden wurde. Das EPK ist mit Texten versehen, nun soll dieses in der Klasse Edotor.java ausgegeben werden. Als erstes wird die Ausgabe UTF-8 tauglich gemacht. Anschließend wird die Liste „list“ durchlaufen und je nach Objekt wird auf der Konsole in Edotor Schreibweise das jeweilige Objekt ausge-

geben.

3.4. Probleme

Es ist äußerst vorteilhaft, dass der Kunde technische Kenntnisse besitzt, die bei der Umsetzung des Projekts helfen können. Diese können dabei helfen, Fehler zu identifizieren, Schwierigkeiten bei der Umsetzung zu beheben und wertvolle Ratschläge zu geben. Allerdings ist es wichtig, frühzeitig Kontakt mit dem Kunden aufzunehmen, um diese Möglichkeit zu nutzen. Da der Entwickler dies bei der Umsetzung erst spät im Projekt erkannte, kam es zu Frustrationen und Verzögerungen und der bis dahin geschriebene Algorithmus musste von Grund auf neu geschrieben werden. Um dies zu vermeiden, sollten Absprachen mit dem Kunden und eine Präsentation des Quellcodes so früh wie möglich stattfinden, um sicherzustellen, dass das Projekt von Anfang an auf einem guten Weg ist und das Wissen und die Erfahrung des Kunden effektiv genutzt werden.

Während der Testphase wurde speziell im „Großen“ getestet, d.h. es wurden viele Loops, Rhomben und Elemente erstellt, um sicherzustellen, dass das generierte EPK skalierbar ist und auch bei einer größeren Anzahl von Elementen und Gates syntaktisch korrekt ist und auch in komplexeren Szenarien funktioniert. Die Tests verliefen positiv und es wurden keine Fehler festgestellt. Allerdings wurde erst zum Ende der Testphase festgestellt, dass bei einer geringen Anzahl von Elementen und eingeschränkten Parametern andere Probleme auftreten können. Insbesondere kann es in einigen Kombinationen zu Endlos-Loops aufgrund der Überprüfungen kommen. Um diese Probleme zu beheben, wird der Überprüfungsprozess nach 100 Durchläufen abgebrochen und das zuletzt überarbeitete EPK verwendet. Es wurde auch festgestellt, dass es in einigen Fällen vorkommt, dass ein Ereignis vor einem (X)OR-Gate steht, obwohl dies syntaktisch nicht korrekt ist. Dies tritt insbesondere auf, wenn die minimalen und maximalen Element-Parameter sehr nahe beieinander liegen und der Algorithmus keinen Spielraum für Änderungen hat. Wird versucht, die Parameter einzuhalten, können syntaktische Fehler im EPK auftreten. Es muss daher noch ein Weg gefunden werden, um sicherzustellen, dass die Parameter eingehalten werden und gleichzeitig ein syntaktisch korrektes EPK generiert wird.

3.5. Ausblick

Der Entwicklungsprozess der Anwendung zur Generierung von ereignisgesteuerten Prozessketten (EPKs) stieß auf Schwierigkeiten bezüglich der Einhaltung der Parameter und der syntaktischen Korrektheit. Aus diesem Grund wurde die Entwicklung auf die Generierung von EPKs beschränkt und die Webseite Edotor.net wurde als Plattform zur Anzeige der generierten EPKs genutzt. Es bleibt jedoch das Problem, dass die eingefügten Texte keinen zusammenhängenden Geschäftsprozess darstellen. Um dies zu lösen, könnten Sprachmodelle wie ChatGPT in Erwägung gezogen werden. Wenn das Problem mit den Parametern gelöst wird, könnte die Anwendung um die Entwicklung von Aufgabentypen erweitert werden. Da die Entwicklung von EPKs das Hauptziel der Aufgabe war, wurde dieser Bereich priorisiert und andere Aspekte wurden für spätere Entwicklungsphasen aufgeschoben.

Abschnitt 4. Sortieralgorithmen

4.1. Aufgabenbeschreibung

4.1.1. Aufgabenstellung

Die Aufgabe zum Thema Sortieralgorithmen besteht darin, eine Anwendung zu entwickeln, mit welcher es Studierenden möglich ist, den Ablauf von Sortieralgorithmen zu verstehen und zu üben. Es soll ein Generator erschaffen werden, welcher beliebig viele Aufgaben generiert, eine Möglichkeit zur Lösung durch den Studierenden anbietet und auch in der Lage ist, diese Lösung auszuwerten.

Die verwendeten Technologien und Frameworks zur Entwicklung der Anwendung sind frei zu wählen.

Nach ersten Überlegungen und Absprachen in den wöchentlichen Teammeetings ließ sich die Aufgabe spezifizieren: Die Anwendung soll es dem Nutzer durch die Eingabe von verschiedenen Parametern ermöglichen, ein numerisches oder alphanumerisches Feld beliebiger Länge zu erzeugen und dieses dann mit den Sortieralgorithmen Bubblesort, Insertsort, Quicksort und Mergesort zu sortieren.

Es soll möglich sein, sich die einzelnen Schritte anzeigen zu lassen und das Feld mit und ohne Hilfestellung eigenständig zu sortieren.

4.1.2. Anforderungen

Basierend auf der Aufgabenstellung ergeben sich die folgenden Anforderungen an die Anwendung:

1. Die Anwendung muss in der Lage sein, ein zufälliges Feld zu generieren, wobei die Zeichenart (numerisch oder alphanumerisch) und die Länge konfiguriert werden können.
2. Für die Generierung alphanumerischer Felder muss die Anwendung auf ein zuvor erstelltes Dictionary mit deutschen Substantiven zugreifen können.
3. Die Anwendung muss es dem Benutzer ermöglichen, die Sortierreihenfolge des Feldes (aufsteigend oder absteigend) sowie den Sortieralgorithmus aus den verfügbaren Algorithmen (Bubblesort, Insertsort, Mergesort und Quicksort) auszuwählen.
4. Die Anwendung muss die erforderlichen Sortierschritte zum Sortieren des Feldes anzeigen können.
5. Es muss einen Übungsmodus geben, in dem der Benutzer das Feld mit Hilfestellung sortieren kann.
6. Es muss einen Prüfungsmodus geben, in dem der Benutzer das Feld ohne Hilfestellung sortieren muss.

4.2. Technische Umsetzung

Im Rahmen des Aufgabenbereichs der Sortieralgorithmen wurde eine Anwendung entwickelt, die

einen Übungs- und Prüfungsmodus zum Sortieren von Feldern mittels Sortieralgorithmen bereitstellt. Studierende haben die Möglichkeit, beliebige Felder mit unterschiedlichen Längen zu generieren und diese anschließend mit verschiedenen Sortieralgorithmen zu sortieren. Dabei können sowohl numerische als auch alphanumerische Felder generiert werden, wobei letztere aus einem selbst angelegten deutschen Substantiv-Dictionary stammen.

Die generierten Felder können sowohl aufsteigend als auch absteigend sortiert werden und die Sortieralgorithmen Bubblesort, Insertsort, Mergesort und Quicksort stehen zur Verfügung. Die Anwendung ermöglicht es den Studierenden, sich die notwendigen Sortierschritte anzeigen zu lassen. Zusätzlich können die Studierenden die Felder selbst sortieren und eine Auswertung erhalten. Es gibt zwei verschiedene Modi, in denen die selbstständige Sortierung der Felder möglich ist: einen Übungsmodus mit Hilfestellungen und einen Prüfungsmodus ohne Hilfestellung.

Bei der Auswertung erhalten die Studierenden eine Gesamtpunktzahl basierend auf den korrekten Sortierschritten. Jede Aufgabe hat dabei dieselbe Höchstpunktzahl, um Lösungen vergleichbar zu machen. Die Anwendung erlaubt es, ein generiertes Feld mit mehreren Sortieralgorithmen zu sortieren, um einen Vergleich zwischen den verschiedenen Algorithmen zu ermöglichen.

4.2.1. Generelles Vorgehen

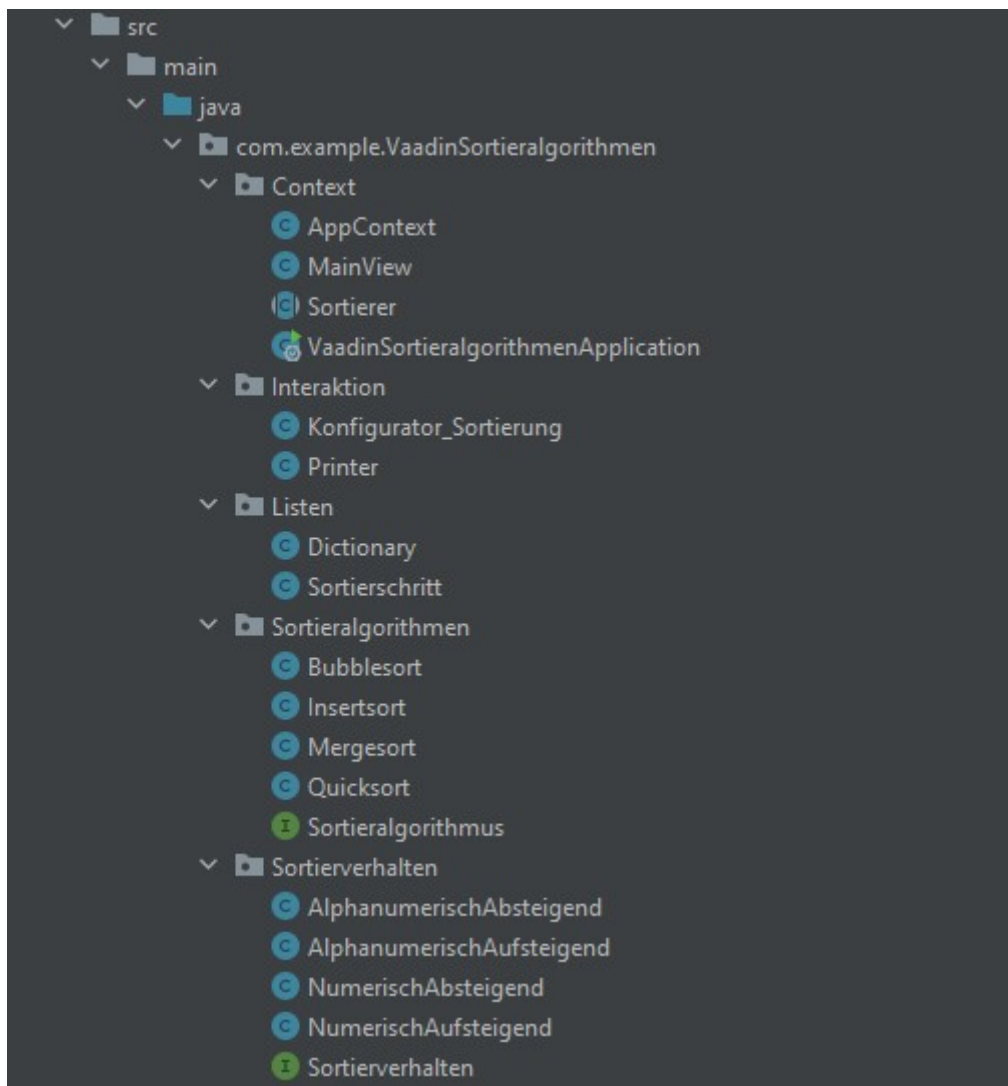
Vor dem Start der tatsächlichen Entwicklung habe ich passende Technologien gesucht, um die Aufgabenstellung umzusetzen. Da ich bereits Programmierkenntnisse in Java hatte, habe ich mich dafür entschieden, Java im Backend zu verwenden und eine Konsolenanwendung zu entwickeln. Zur Umsetzung des Frontends habe ich Vaadin ausgewählt, ein Java-basiertes Webframework mit vorgefertigten UI-Komponenten. Dadurch konnte ich meine Konsolenanwendung in eine Vaadin-Anwendung überführen und die vorgefertigten Komponenten zur Ausgabe nutzen.

Hinweis: Alle folgenden Erläuterungen beziehen sich auf die Vaadin-Anwendung, da diese die Konsolenanwendung beinhaltet. Somit muss die Konsolenanwendung nicht getrennt betrachtet werden.

4.2.2. Ordnerstruktur

Folgende Abbildung soll nur zum Überblick über die Ordnerstruktur dienen und nicht zur Erklärung beitragen. Verwendete Klassen werden in den nächsten Abschnitten erläutert.

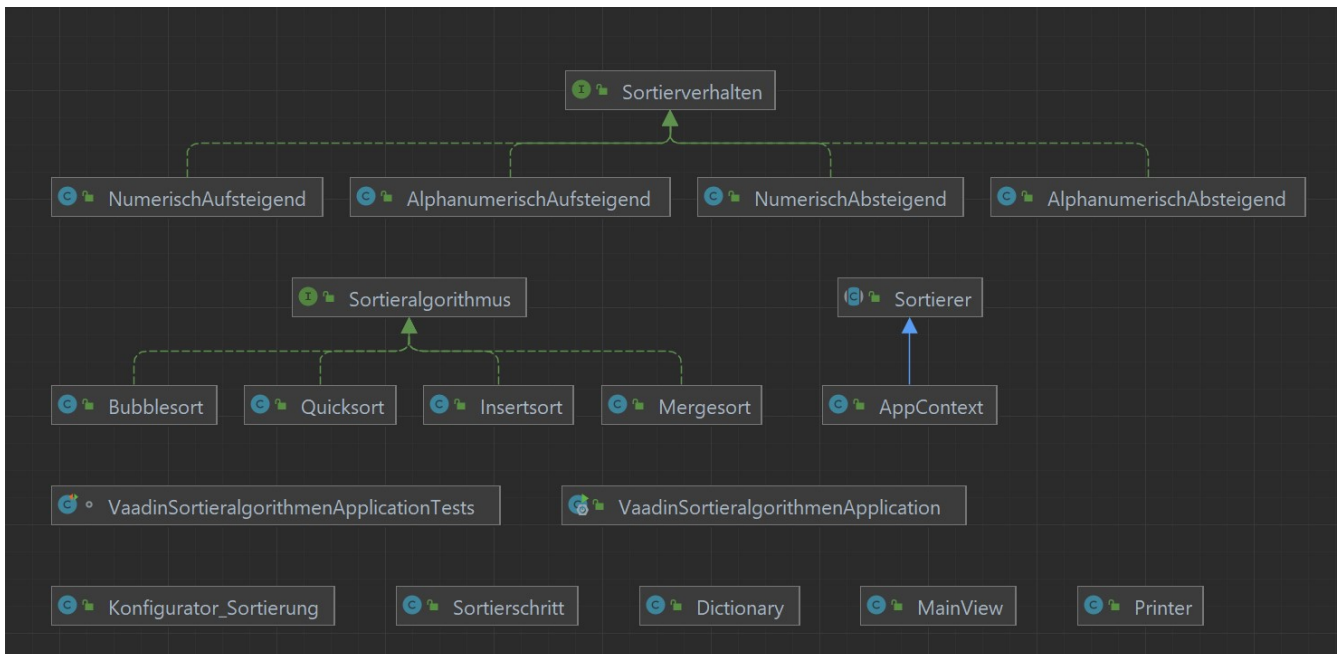
Ordnerstruktur der Vaadin-Anwendung



4.2.3. UML-Diagramm

Das UML-Diagramm soll den Zusammenhang der einzelnen Klassen genauer erläutern und zu einem besseren Verständnis der Anwendung beitragen.

UML-Diagramm zur Vaadin-Anwendung



4.2.4. Frontend

Für die Umsetzung des Frontends wurde das Open-Source-Webframework Vaadin verwendet. Mit Vaadin ist es möglich, eine Webanwendung mit Frontend zu erstellen, ohne dass HTML-, CSS- oder JavaScript-Kenntnisse erforderlich sind. Stattdessen kann eine Java-Anwendung erstellt werden, welche mittels vorgefertigter Komponenten, wie z.B. Buttons, Textfelder und Tabellen, eine Ausgabe erzeugt.

Vaadin verfügt über eine leistungsfähige Layout-Engine, mit der verschiedene Layouts für eine Anwendung erstellt werden können. Typische Arten von Layouts sind horizontale und vertikale Layouts, Grid-Layouts oder Form-Layouts. Verschiedene Komponenten können einem Layout hinzugefügt werden und je nach Typ des Layouts werden sie vertikal oder horizontal angeordnet. Die Layouts können unter verschiedenen Bedingungen ein- und ausgeblendet werden, um eine nutzerfreundliche Benutzeroberfläche zu gewährleisten.

Um ein Vaadin-Projekt zu erstellen, kann man den Spring Initializr (<https://start.spring.io/>) verwenden. Um diese Anwendung zu reproduzieren, müssen folgende Einstellungen getätigt werden:

- Project: Maven
- Language: Java
- Spring Boot: 2.7.9
- Packaging: Jar
- Java: 17
- Dependencies: Vaadin, Spring Boot DevTools

Initialisierung des Projekts mit Spring Initializr

The screenshot shows the Spring Initializr interface with the following configuration:

- Project:**
 - ☒ Gradle - Groovy
 - ☐ Gradle - Kotlin
 - ☐ Maven
- Language:**
 - ☒ Java
 - ☐ Kotlin
 - ☐ Groovy
- Spring Boot:**
 - ☐ 3.1.0 (SNAPSHOT)
 - ☐ 3.1.0 (M1)
 - ☐ 3.0.4 (SNAPSHOT)
 - ☐ 3.0.3
 - ☒ 2.7.10 (SNAPSHOT)
 - ☒ 2.7.9
- Project Metadata:**
 - Group:
 - Artifact:
 - Name:
 - Description:
 - Package name:
 - Packaging: ☒ Jar ☐ War
 - Java: ☐ 19 ☒ 17 ☐ 11 ☐ 8
- Dependencies:**
 - Vaadin** WEB: A web framework that allows you to write UI in pure Java without getting bogged down in JS, HTML, and CSS.
 - Spring Boot Dev Tools** DEVELOPER TOOLS: Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

4.2.4.1. Erläuterung: Begriffe, Konzepte und Datenstrukturen

Im Folgenden werden wichtige Aspekte und Funktionsweisen des Frontends erläutert.

4.2.4.2. Main View (Klasse)

Die Main-View ist eine Java-Klasse, welche das gesamte Frontend beinhaltet und dieses mit dem Backend verbindet. In ihr werden alle Formen und Komponenten definiert, sowie gesteuert, welche ein- und ausgeblendet werden. Die Anwendung ist somit eine Single-Page-Application.

Die Klasse erbt von einem vertikalen Layout, damit hier Komponenten untereinander angezeigt werden können. Außerdem wird der Pfad zur Anwendung festgelegt. Dieser ist bei einer lokalen Installation <http://localhost:8080/>.

Um nicht jede Komponente einzeln ein- und ausblenden zu müssen, ist das Frontend in vier Formen gegliedert, welchen die einzelnen Komponenten zugewiesen werden. Es gibt je eine Form um das Feld und die Sortierung zu konfigurieren, alle Sortierschritte anzuzeigen, für den Übungsmodus und den Prüfungsmodus. Die Formen haben dabei den Dateityp 'Component', welchen die Vaadin-Bibliothek mit sich bringt.

Jede Form erhält wieder ein vertikales Layout, welchem dann beliebig viele Komponenten zugewiesen werden können. Verwendete Komponenten sind zum Beispiel Buttons, Select-fields, Textfelder und Layouts.

Im Folgenden werden die vier Formen genauer erläutert.

4.2.4.3. Steuerung der Ein- und Ausblendung

In Vaadin lassen sich alle Komponenten mit der Funktion `setVisible()` ein- und ausblenden. Somit kann bei Betätigung eines bestimmten Buttons der zugehörige Event-handler die Sichtbarkeit von Komponenten steuern.

In der Anwendung ist immer nur genau eine Form sichtbar. Möchte man zu einer anderen Form

wechseln, wird die aktuelle aus- und die neue Form eingeblendet.

Zusätzlich kommt es vor, dass Komponenten innerhalb einer Form ein- oder ausgeblendet werden. Denn nicht jede Form zeigt beim Einblenden gleich alle Komponenten. Bestimmte Ereignisse können auch hier die Sichtbarkeit verändern. Sobald eine Form geschlossen wird, wird sie auf den ursprünglichen Startzustand zurückgesetzt. Beim Verlassen einer Form wird nicht nur die Sichtbarkeit verändert. Auch müssen bestimmte Komponenten, wie Grids oder ListDataProvider geleert werden, um beim erneuten Aufrufen der Form korrekt zu funktionieren.

4.2.4.4. Konfigurationsform

Die Konfigurationsform dient der Eingabe vom Feld- und Sortierparametern durch den Nutzer. Außerdem kann entschieden werden, in welchen der drei genannten Modis gewechselt wird.

Die Form beinhaltet ein Accordion, welches drei Stufen enthält: Feld konfigurieren, Sortierung konfigurieren und Auswertung wählen. Es ist sichergestellt, dass der Nutzer diese Schritte nacheinander ausführt, aber auch die Möglichkeit hat, vorherige Stufen aufzuklappen.

In der Stufe 'Feld konfigurieren' kann der Nutzer die Zeichenart und Länge des zu generierenden Feldes festlegen. Dabei kann zwischen einem numerischen oder alphanumerischen Feld gewählt werden, welches eine Mindestlänge von zwei Elementen hat und keine obere Begrenzung für die Länge besitzt. Außerdem gibt es einen Button 'Feld generieren', welcher die [Section 4.2.5.3, "Generierung des Feldes"](#) auslöst und die nächste Stufe im Akkordion aufklappt.

Die Stufe 'Sortierung konfigurieren' zeigt das eben generierte Feld an. Der Nutzer kann mit diesem Feld fortfahren oder es durch ein anderes beliebiges Feld ersetzen. Zudem kann die Sortierreihenfolge und der Algorithmus gewählt werden. Das Feld lässt sich somit aufsteigend oder absteigend mit den Algorithmen Bubblesort, Insertsort, Quicksort und Mergesort sortieren. Sind alle Konfigurationen gewählt, kann man den Button 'Sortieren' betätigen, was die Sortierung ([Section 4.2.5.9, "Sortieralgorithmus \(Datenstruktur\)"](#)) auslöst und die Stufe 'Auswertung wählen' aufklappt.

Die letzte Stufe umfasst eine Zusammenfassung der Konfiguration und die Möglichkeit, eine Auswertung zu wählen. Es wird das generierte Feld, der gewählte Algorithmus und die gewählte Sortierung gezeigt. Außerdem gibt es drei Buttons, welche zu den Modis 'Feld anzeigen', 'Übungsmodus' und 'Prüfungsmodus' führt. Betätigt der Nutzer einen der Buttons, wird in die gewünschte Form gewechselt. Dafür wird die Konfigurationsform ausgeblendet und der

4.2.4.5. FeldAnzeigen-Form

Die FeldAnzeigen-Form ermöglicht es sich die erforderlichen Sortierschritte zum Sortieren des Feldes anzeigen zu lassen. Es können entweder jeder Schritt einzeln oder alle Schritte auf einmal angezeigt werden. Zusätzlich gibt es eine Möglichkeit zur Konfigurationsform zurückzugelangen.

Die Form enthält die Buttons 'Nächster Schritt', 'Alle Schritte anzeigen' und 'Zurück zur Konfiguration'. Welche Events diese Buttons auslösen, wird im [Section 4.2.5.11, "Feld anzeigen \(Modus\)"](#) erklärt.

Die einzelnen Schritte werden in einem Grid dargestellt. Dies ist in Vaadin eine Tabelle mit Zeilen und Spalten, wobei der Typ der Spalten genaustens festgelegt werden muss. In diesem Fall wird der aktuelle Stand des Feldes, das erste und zweite getauschte Element, sowie die zugehörigen Indexe

dargestellt.

Wenn alle Schritte angezeigt sind, wird ein Label eingeblendet, was den Nutzer dahingehend informiert. Betätigt der Nutzer weiterhin die Buttons zum Anzeigen der Schritte, wird eine Notification eingeblendet.

4.2.4.6. Übungsmodus-Form

Die Übungsmodus-Form erlaubt es dem Nutzer das generierte Feld selbst zu sortieren. Macht der Nutzer drei falsche Eingaben, wird Hilfe vom System angeboten.

Die Form umfasst ein readonly-Textfeld, welches das generierte Feld anzeigt, zwei Textfelder zur Eingabe der zu tauschenden Indexe, sowie die Buttons 'Tauschen' und 'Zurück zur Konfiguration'.

Was ein Klick auf den Button 'Tauschen' auslöst, wird im Abschnitt [Section 4.2.5.12, “Übungsmodus \(Modus\)”](#) erklärt.

Wenn ein Nutzer die Indexe richtig eingibt, wird der Schritt nach dem Tauschen der beiden Elemente im Grid angezeigt. Ist die Sortierung beendet, wird ein Label mit der benötigten Schrittzahl und der Anzahl der Fehler eingeblendet.

Sind die Nutzereingaben nicht korrekt, wird die Notification 'Schritt ist falsch' gezeigt. Nach drei falschen Eingaben gibt es ein Popup, was über die Anzahl der Fehler in diesem Schritt berichtet und die Möglichkeit mit sich bringt, sich den nächsten Schritt anzeigen zu lassen.

Außerdem gibt es Notifications für den Fall, dass die eingegebenen Indexe keine Zahlen oder nicht in dem Sortierfeld vorhanden sind.

4.2.4.7. Prüfungsmodus-Form

Die Prüfungsmodus-Form hat einen ähnlichen Aufbau wie die Übungsmodus-Form. Es kommt lediglich ein Button 'Auswerten' hinzu. In diesem Modus tauscht der Nutzer die Elemente eigenständig, ohne dass die Eingaben vom System auf Richtigkeit geprüft werden. Es wird nur geprüft, ob die eingegebenen Werte in dem Feld enthaltene Indexe sind und es wird gegebenenfalls eine Notification gezeigt. Die eingegebenen Indexe werden in dem Feld vertauscht und direkt im nächsten Schritt sichtbar. Der Nutzer bekommt dabei keine Hilfe.

Die Schritte werden wieder in einem Grid angezeigt. Sollen diese ausgewertet werden, kann man den Button 'Auswerten' betätigen. Es folgt eine dreischrittige Auswertung.

Zuerst wird ein Grid mit den Sortierschritten des Systems angezeigt. Dieses enthält also alle Schritte, die erforderlich sind, um das Feld mit dem gewählten Algorithmus korrekt zu sortieren.

Anschließend gibt es einen eins zu eins Vergleich der Schritte des Nutzers und der des Systems. Für jeden Schritt wird der zu dem Zeitpunkt aktuelle Zustand des Feldes gezeigt und entschieden, ob der Schritt des Nutzers richtig oder falsch ist. Hierbei gibt es die Möglichkeiten: 'Dieser Schritt ist falsch', 'Dieser Schritt ist richtig', 'Dieser Schritt fehlt' und 'Dieser Schritt ist zusätzlich'. Da es das Grid nicht ermöglicht, Sortierschritte und Text im Wechsel zu zeigen, wird hierbei eine TextArea verwendet.

Zuletzt gibt es eine Punktevergabe. Diese zeigt die Häufigkeit der richtigen und falschen Schritte

und gibt ein Ergebnis in Punkten, sowie in Prozent an. Auch hierfür wird eine TextArea verwendet.

Wie die Schritte ausgewertet werden, wird im Abschnitt [Section 4.2.5.13, “Prüfungsmodus \(Modus\)”](#) erläutert.

4.2.5. Backend

Das Backend der Anwendung wurde mit Java entwickelt. Hierbei wurde die JDK-Version 17.0.2 verwendet.

4.2.5.1. Erläuterung: Begriffe, Konzepte und Datenstrukturen

Im Folgenden erläutere ich Begriffe, angewendete Konzepte, Datenstrukturen und Funktionen. Diese sollen beim Verstehen der Funktionsweise des Backends dienen.

4.2.5.2. AppContext

Die Klasse AppContext bestimmt den Ablauf der Anwendung. Zuerst werden im Frontend die Benutzereingaben abgefragt und dann an diese Klasse übergeben. Je nach Benutzereingabe wird hier dann das Sortierverhalten und der Sortieralgorithmus gesetzt, welcher dann auch das gesetzte Sortierverhalten und das vorher generierte Sortierfeld erhält.

Nachdem das Verhalten gesetzt wurde, muss dann nur noch die *sortieren()*-Methode aufgerufen werden und die Schrittliste mittels *getList()* geholt werden.

4.2.5.3. Generierung des Feldes

Wenn sich der Benutzer im Frontend für ein alphanumerisches oder numerisches Feld entschieden und außerdem eine Länge für dieses Feld festgelegt hat, wird die Methode *feld_generieren()* der Klasse Konfigurator_Sortierung aufgerufen. Je nach Eingabe wird dann ein zufälliges Feld mit der gewünschten Länge generiert. Wenn ein alphanumerisches Feld gewünscht ist, werden sich hierbei zufällige Substantive aus dem zuvor gepflegtem Dictionary geholt.

4.2.5.4. Sortierfeld (Begriff)

Ein Sortierfeld ist eine Reihe von Elementen, welche zufällig angeordnet sind und eine bestimmte Länge besitzen. Dieses soll vom Sortieralgorithmus sortiert werden. Das Sortierfeld, im folgenden Kontext auch als Feld bezeichnet, kann numerische oder alphanumerische Elemente enthalten.

Ein numerisches Feld der Länge 5 könnte folgendermaßen aussehen: 12, 45, 1, 67, 84.

Ein alphanumerisches Feld der Länge 3 könnte folgendermaßen aussehen: Baum, Wurzel, Blatt.

4.2.5.5. Sortierfeld (Datenstruktur)

Um sicherzustellen, dass die Sortieralgorithmen sowohl für numerische als auch für alphanumerische Felder funktionieren, ohne jedes Mal zwischen diesen unterscheiden zu müssen, werden alle Elemente in einem String-Array gespeichert.

4.2.5.6. Sortierschritt (Begriff)

Ein Sortierschritt bezeichnet einen einzelnen Schritt, den der Sortieralgorithmus beim Sortieren eines Feldes ausführt. Dabei sind nicht alle Schritte gemeint, die der Algorithmus während des Durchlaufs des Feldes macht. Ein Sortierschritt tritt nur dann auf, wenn der Zustand des Feldes verändert wird, zum Beispiel durch den Austausch von Elementen wie der 45 mit der 67 im oben gezeigten numerischen Feld.

4.2.5.7. Sortierschritt (Datenstruktur)

Wenn ein Sortieralgorithmus beim Sortieren eines Feldes die Reihenfolge der in dem Feld enthaltenen Elemente ändert, so wird diese Veränderung in einem Sortierschritt gespeichert. In Java habe ich die Datenstruktur eines Sortierschrittes mit einer eigenen, öffentlichen Klasse umgesetzt. Diese enthält:

Bezeichnung	Datentyp	Erläuterung
Sortierfeld	String-Array	Der Zustand des Feldes nach Umsetzung des Schrittes.
Element1	String	Das erste zu tauschende Element im Feld.
Element2	String	Das zweite zu tauschende Element im Feld.
Index1	int	Der Index des ersten Elements.
Index2	int	Der Index des zweiten Elements.

4.2.5.8. Sortieralgorithmus (Begriff)

Sortieralgorithmen, im Folgenden auch als Algorithmen bezeichnet, dienen dazu, Felder zu sortieren.

In der Anwendung wurden folgende Algorithmen implementiert:

- Bubblesort,
- Insertsort,
- Quicksort und
- Mergesort.

Diese Algorithmen unterscheiden sich in Bezug auf ihre Geschwindigkeit, Speicherbedarf, Stabilität und Rechenaufwand, allerdings werden diese Unterschiede in der Anwendung nicht berücksichtigt. Stattdessen steht der Sortiervorgang im Mittelpunkt, also wie die Algorithmen ein Feld sortieren.

4.2.5.9. Sortieralgorithmus (Datenstruktur)

Jeder Algorithmus besitzt eine eigene, öffentliche Klasse. Die Klasse enthält:

Bezeichnung	Datentyp	Erläuterung
Sortierverhalten	Sortierverhalten	Beschreibt, ob ein Feld aufsteigend oder absteigend sortiert werden soll.
Sortierfeld	String-Array	Das zu sortierende Feld.
Sortierliste	ArrayList<Sortierschritt>	Eine Liste, welche alle Sortierschritte, die zum Sortieren des Feldes nötig sind, enthält.

Jede Klasse besitzt eine *sortieren()*-Funktion, welche das übergebene Feld unter Berücksichtigung des Sortierverhaltens sortiert. Außerdem gibt es eine *getList()*-Methode, welche die Schrittliste zurückgibt.

Demonstrativ erkläre ich im Folgenden den Vorgang des Bubblesort-Algorithmus:

Beim Bubblesort werden jeweils benachbarte Elemente des zu sortierenden Feldes miteinander verglichen und gegebenenfalls vertauscht, wenn das linke Element größer ist als das rechte Element. Das größte Element "blubbert" dabei nach oben, wodurch dieser Algorithmus seinen Namen erhalten hat.

Der Algorithmus beginnt beim ersten Element des Feldes und vergleicht diesen mit dem Zweiten. An dieser Stelle spielt das Sortierverhalten eine Rolle - die *verhalten()*-Methode der instanziierten Klasse Sortierverhalten wird aufgerufen und beide Elemente werden übergeben. Je nach gesetztem Verhalten wird geprüft, ob das erste Element größer, kleiner oder gleich dem zweiten Element ist und es wird ein Boolean zurückgegeben. Besitzt dieser den Wert 'true', werden die beiden Elemente vertauscht.

Bei jedem Tausch speichert der Algorithmus einen Sortierschritt in der Schrittliste. Nach Beendigung der Sortierung enthält die Schrittliste alle zur Sortierung notwendigen Sortierschritte.

Der gesamte Vorgang wird wiederholt, bis das erste Element nicht mehr getauscht wird. Anschließend wird zum nächsten Element gewechselt und auch dieses wird von links nach rechts iteriert und getauscht oder eben nicht.

Hinweis: Wie das Verhalten der Klasse gesetzt wird und die Elemente miteinander verglichen werden, ist im Abschnitt [Section 2.2.3.4, "Strategy Pattern"](#) beschrieben.

4.2.5.10. Strategy Pattern

Da bei der Sortierung des Feldes, je nach gewähltem Sortieralgorithmus und Sortierverhalten, ein unterschiedlicher Algorithmus nötig ist, habe ich bei der Implementierung im Backend das Strategy Pattern genutzt. Hierbei ist es möglich, das Verhalten einer Klasse zur Laufzeit zu ändern. Um das Strategy Pattern zu realisieren, gibt es die Interfaces Sortieralgorithmus und Sortierverhalten, welche alle nötigen Funktionen mit ihren Parametern definiert. Zusätzlich gibt es die abstrakte Klasse Sortierer, welche ein Default Verhalten für die Laufzeit festlegt und die Methoden *setSortieralgorithmus()* und *setSortierverhalten()* implementiert, um das Verhalten der Anwendung zu setzen.

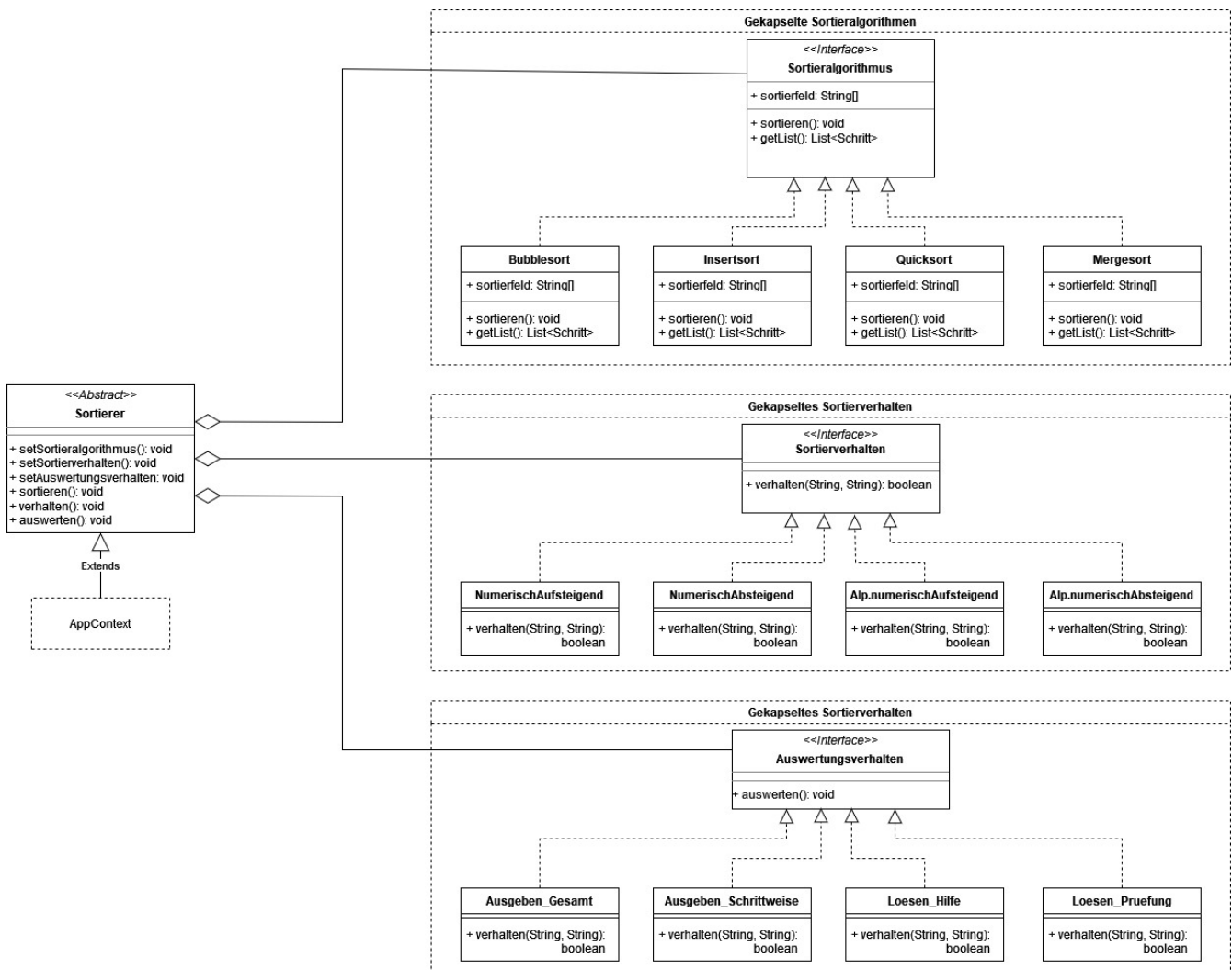
Das Interface Sortieralgorithmus hat hierbei die Methoden *sortieren()*, welche zum Sortieren der

Felder dient und alle nötigen Schritte in der Schrittliste speichert und *getList()*, welche die Liste mit den Sortierschritten zurückgeben kann. Von diesem Interface erben die Klassen Bubblesort, Insertsort, Quicksort und Mergesort, in welchen die genannten Methoden dann implementiert sind.

Im Interface Sortierverhalten befinden sich alle Methoden, welche das Verhalten in den Sortieralgorithmen bestimmen und jeweils einen Boolean zurückgeben. Der Sortieralgorithmus Bubblesort beinhaltet einen Vergleich zweier Elemente des Feldes. An dieser Stelle wird die *verhalten()*-Methode aufgerufen und je nach konfiguriertem Sortierverhalten ein true oder false zurückgegeben. Vom Sortierverhalten erben die Klassen AlphanumerischAbsteigend, AlphanumerischAufsteigend, NumerischAbsteigend und NumerischAufsteigend. In all den Klassen wird genau festgelegt, wie sich der Algorithmus beim Vergleich zweier Elemente verhalten soll.

Um das Verhalten zu verändern, muss der Nutzer im Frontend das gewünschte Verhalten konfigurieren. Hierbei gibt es eine Auswahl zwischen den vier Algorithmen und der Möglichkeit zum aufsteigenden und absteigenden Sortieren. Hat der Nutzer alle Eingaben getätigt, kann nun mit den Methoden *setSortieralgorithmus()* und *setSortierverhalten()* das Verhalten der Klasse [Section 4.2.5.2, “AppContext”](#) gesetzt werden.

UML-Diagramm zum Strategy Pattern



4.2.5.11. Feld anzeigen (Modus)

In der [Section 4.2.4.5, “FeldAnzeigen-Form”](#) kann der Nutzer entscheiden, ob alle Schritte einzeln

oder gleichzeitig ausgegeben werden sollen.

Dies geschieht in dem durch die Liste mit Sortierschritten iteriert wird. Hierfür ist ein `ListDataProvider` nötig, welcher die einzelnen Schritte erhält und nach und nach im Grid anzeigt. Mittels `Schrittcountern` wird sichergestellt, dass es zu keinem Fehler kommt.

Sind alle Schritte angezeigt, erhält der Nutzer einen Hinweis.

4.2.5.12. Übungsmodus (Modus)

In der [Section 4.2.4.6, “Übungsmodus-Form”](#) gibt der Nutzer zu tauschende Indexe ein, welche auf Richtigkeit geprüft werden und anschließend im Grid angezeigt werden.

Hierfür werden die eingegebenen Werte, nachdem der 'Tauschen'-Button betätigt wurde zuerst in einen Integer geparkt. Gelingt dies nicht, wird eine Notification gezeigt, welche den Nutzer über die ungültige Eingabe informiert. Ebenfalls werden die Fälle 'Gleicher Index' und 'Index existiert nicht' mittels `if`-Abfragen geprüft.

Sind die Indexe validiert, werden sie auf Richtigkeit geprüft. Die Schrittliste enthält alle Sortierschritte, die das System benötigt hat, inklusive der getauschten Indexe. Somit wird durch diese iteriert und an der richtigen Stelle die Indexe ausgelesen. Diese werden dann mit den eingegebenen Indexen verglichen. Die Reihenfolge der Eingabe spielt dabei keine Rolle.

Sind die eingegebenen Werte richtig, kann der Schritt angezeigt werden. Ist dies nicht der Fall, wird der Nutzer informiert und kann es erneut versuchen. Ein Counter zählt die falschen Schritte. Sind für einen Schritt drei falsche Eingaben gemacht worden, wird ein Dialog eingeblendet, der Hilfe anbietet. Nimmt der Nutzer die Hilfe an, so wird der nächste Schritt angezeigt, in dem der Index der Schrittliste um eins erhöht wird.

Sobald das Ende der Liste erreicht ist, wird ein Label gezeigt, welches die benötigte Schrittzahl und die Anzahl der gemachten Fehler enthält.

4.2.5.13. Prüfungsmodus (Modus)

Der Prüfungsmodus, welcher in der [Section 4.2.4.7, “Prüfungsmodus-Form”](#) aufgerufen wird, funktioniert vom Prinzip her wie der Übungsmodus.

Stattdessen werden die eingegebenen, validierten Indexe einfach vertauscht, ohne dass ihre Richtigkeit geprüft wird. Nach dem Tausch wird der Sortierschritt im Grid angezeigt. Alle Schritte werden in einer weiteren Liste, der Schrittliste des Nutzers, gespeichert. Diese hilft bei der Auswertung.

Der Vorgang wiederholt sich, bis der Nutzer den Button 'Auswertung' klickt. Von nun an ist die weitere Eingabe der Schritte nicht mehr möglich, denn die Textfelder und der 'Tauschen'-Button werden ausgeblendet.

Es wird ein neues Grid erzeugt, welches die Sortierschritte, die das System (nicht der Nutzer!) getätigt hat anzeigt, indem wieder durch die Schrittliste des Systems iteriert wird.

Anschließend werden die Schritte des Nutzers auf Korrektheit geprüft. Es wird parallel durch beide Listen iteriert und die zugehörigen Felder werden an eine `vergleichen()`-Funktion übergeben. Diese Funktion prüft, ob der Schritt richtig, falsch, zusätzlich oder fehlend ist. Hierfür wird sich an der

`Arrays.equals()`-Methode bedient. Je nachdem welcher Fall eintritt, zeigt die Funktion einen Text in der TextArea des eins zu eins Vergleiches an und erhöht den zugehörigen Counter.

Nachdem alle Schritte miteinander verglichen wurden, werden Punkte für die Lösung des Nutzers vergeben. Eine Sortieraufgabe hat immer genau 10 maximale Punkte. Diese 10 Punkte werden durch die benötigten Schritte vom System geteilt, um eine Wertigkeit für einen Punkt zu erhalten. Die erreichte Punktzahl wird dann errechnet, indem die Anzahl der richtigen Schritte mit der Wertigkeit eines Punktes multipliziert wird. Punktabzug gibt es für falsche und zusätzliche Schritte, fehlende Schritte erhalten keinen Punkt, wirken sich aber auch nicht negativ auf die Gesamtpunktzahl aus. Die Punktzahl wird sowohl absolut, als auch prozentual in einer Textarea angezeigt.

4.3. Ergebnisse

4.3.1. Erfüllte Anforderungen

Im Folgenden werden erneut die Anforderungen, sowie ein Status, ob diese erreicht wurden, aufgeführt.

Anforderung	Status
Die Anwendung muss in der Lage sein, ein zufälliges Feld zu generieren, wobei die Zeichenart (numerisch oder alphanumerisch) und die Länge konfiguriert werden können.	Erreicht
Für die Generierung alphanumerischer Felder muss die Anwendung auf ein zuvor erstelltes Dictionary mit deutschen Substantiven zugreifen können.	Erreicht
Die Anwendung muss es dem Benutzer ermöglichen, die Sortierreihenfolge des Feldes (aufsteigend oder absteigend) sowie den Sortieralgorithmus aus den verfügbaren Algorithmen (Bubble-sort, Insertsort, Mergesort und Quicksort) auszuwählen.	Teilweise erreicht: Der Mergesort-Algorithmus ist zwar implementiert, jedoch ist das Frontend nicht auf diesen Algorithmus abgestimmt, weshalb diese Anforderung nicht vollständig erfüllt wurde.
Die Anwendung muss die erforderlichen Sortierschritte zum Sortieren des Feldes anzeigen können.	Erreicht
Es muss einen Übungsmodus geben, in dem der Benutzer das Feld mit Hilfestellung sortieren kann.	Erreicht
Es muss einen Prüfungsmodus geben, in dem der Benutzer das Feld ohne Hilfestellung sortieren muss.	Erreicht

Die grundlegenden Anforderungen wurden somit erreicht und der Identifikation mit einem Generator ist mit der Generierung des Feldes gegeben. Studenten können sich Aufgaben bzw. Felder

generieren und dann diesen in verschiedenen Modis den Ablauf von Sortieralgorithmen üben.

4.3.2. Stand der Benutzeroberfläche

Die Benutzeroberfläche funktioniert für die erreichten Anforderungen einwandfrei und beim Test konnten keine Bugs gefunden werden.

Konfiguration des Feldes

Konfiguration

▼ Feld konfigurieren

Zeichenart

numerisch ▼

Anzahl

– 5 +

Feld generieren

Konfiguration der Sortierung

▼ Sortierung konfigurieren

Generiertes Feld

[62, 57, 48, 99, 45]

Sortierung

aufsteigend ▼

Algorithmus

Bubblesort ▼

Sortieren

Konfiguration der Auswertung

▼ Auswertung wählen

Generiertes Feld

[62, 57, 48, 99, 45]

Gewählter Algorithmus

Bubblesort

Gewählte Sortierung

aufsteigend

Feld anzeigen

Übungsmodus

Prüfungsmodus

Ausschnitt aus dem Übungsmodus: Nutzer hat wiederholt Fehler gemacht

Übungsmodus

Generiertes Feld

[62, 57, 48, 99, 45]

Index 1

Index 2

Tauschen

Zurück zur Konfiguration

Feld	Getaushtes Elemen...	Getaushtes Elemen...	Getauschter Index 1	Getauschter Index 2
[0] 57; [1] 62; [2] 48; [3] 99; [4] 45;				1
[0] 57; [1] 48; [2] 62; [3] 99; [4] 45;				2

Wiederholter Fehler

Sie haben schon 3 Fehler in diesem Schritt gemacht. Möchten Sie den Schritt anzeigen lassen?

Nein

Ja

Beispiel einer möglichen Notification

Index 1 existiert nicht

Ausschnitt aus dem Prüfungsmodus: Nutzer hat die Sortierung noch nicht beendet

Prüfungsmodus

Generiertes Feld

[62, 57, 48, 99, 45]

Index 1

Index 2

Tauschen

Auswerten

Zurück zur Konfiguration

Feld	Getaushtes Element 1	Getaushtes Element 2	Getauschter Index 1	Getauschter Index 2
[0] 57; [1] 62; [2] 48; [3] 99; [4] 45;	62	57	0	1
[0] 57; [1] 48; [2] 62; [3] 99; [4] 45;	62	48	1	2
[0] 57; [1] 48; [2] 62; [3] 45; [4] 99;	99	45	3	4
[0] 48; [1] 57; [2] 62; [3] 45; [4] 99;	57	48	0	1
[0] 48; [1] 57; [2] 45; [3] 62; [4] 99;	62	45	2	3
[0] 48; [1] 45; [2] 57; [3] 62; [4] 99;	57	45	1	2
[0] 45; [1] 48; [2] 57; [3] 62; [4] 99;	48	45	0	1

Ausschnitt aus dem Prüfungsmodus: Nutzer hat die Sortierung beendet und die Liste des Systems wird angezeigt

Sortierschritte des Systems:

Feld	Getaushtes Element 1	Getaushtes Element 2	Getauschter Index 1	Getauschter Index 2
[0] 36; [1] 25; [2] 95; [3] 29; [4] 44;	95	25	1	2
[0] 36; [1] 25; [2] 29; [3] 95; [4] 44;	95	29	2	3
[0] 36; [1] 25; [2] 29; [3] 44; [4] 95;	95	44	3	4
[0] 25; [1] 36; [2] 29; [3] 44; [4] 95;	36	25	0	1
[0] 25; [1] 29; [2] 36; [3] 44; [4] 95;	36	29	1	2

Ausschnitt aus dem Prüfungsmodus: Vergleich eines richtigen Schrittes

Auswertung einzelner Schritte:

1. Vergleich:

Nutzer - Aktueller Zustand: [0] 36; [1] 25; [2] 95; [3] 29; [4] 44; || Getauschte Felder: [1] 95, [2] 25
System - Aktueller Zustand: [0] 36; [1] 25; [2] 95; [3] 29; [4] 44; || Getauschte Felder: [1] 95, [2] 25

Sie haben diesen Schritt richtig sortiert.

Ausschnitt aus dem Prüfungsmodus: Vergleich eines zusätzlichen Schrittes

Auswertung einzelner Schritte:

1. Vergleich:

Nutzer - Aktueller Zustand: [0] 36; [1] 25; [2] 95; [3] 29; [4] 44; || Getauschte Felder: [1] 95, [2] 25
System - Aktueller Zustand: [0] 36; [1] 25; [2] 95; [3] 29; [4] 44; || Getauschte Felder: [1] 95, [2] 25

Sie haben diesen Schritt richtig sortiert.

4.4. Ausblick

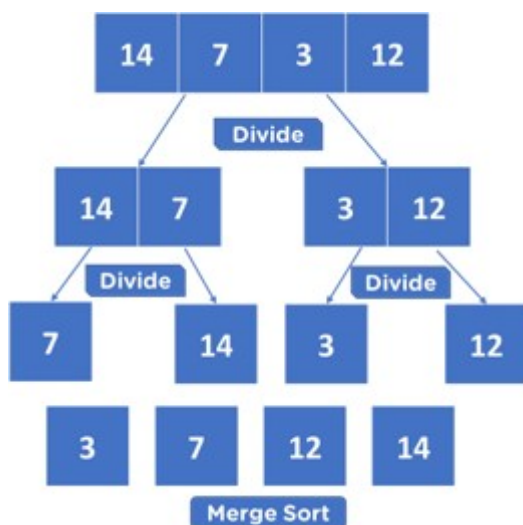
Die Anwendung wurde in einem zeitlich begrenzten Rahmen entwickelt und stellt eine Grundlage dar, auf der in Zukunft aufgebaut werden kann.

Primär gibt es zwei Aspekte, die in Zukunft zur Anwendung hinzugefügt werden sollten:

4.4.1. Mergesort

Der Mergesort-Algorithmus ist bereits im Backend implementiert. Das System ist also in der Lage, ein Feld mit dem Mergesort-Algorithmus zu sortieren. Jedoch ist das Frontend nicht darauf ausgelegt, geteilte Felder, wie es beim Mergesort notwendig ist, darzustellen und auszuwerten.

Funktionsweise des Mergesort-Algorithmus



Es muss eine Möglichkeit gefunden werden, sodass der Nutzer das generierte Feld an beliebigen Stellen teilen und wieder zusammenfügen kann.

4.4.2. Folgefehler in der Auswertung

Die Auswertung funktioniert zurzeit statisch. Macht der Nutzer einen Fehler in den ersten Schritten, sind die folgenden in der Regel auch falsch. Bei der Auswertung muss das System, sobald ein Fehler auftritt, wieder in den Algorithmus springen und an gegebener Stelle neu sortieren.

Hinweis: Hierfür müssen die Sortieralgorithmen neu entwickelt werden, sodass es möglich ist, an jedem beliebigen Stand in den Algorithmus einzusteigen. Erste Algorithmen dafür liegen vor, sind aber nicht in die finale Anwendung integriert. Hierfür hat die Zeit nicht gereicht.

Abschnitt 5. Generierung von zufallsbasierten Molekülen

5.1. Aufgabenbeschreibung

Die Aufgabe war es einen Algorithmus zu entwickeln, der zufällige Atome aus dem Periodensystem auswählt und diese unter Einhaltung der chemischen Regeln miteinander verbindet, so dass am Ende ein Molekül entsteht, das aus zufälligen Atomen besteht. Das endgültige Ziel ist dabei, diese Moleküle in Aufgaben für Studierende zu integrieren. Diese Aufgaben könnten dann zum Beispiel sein, dass ein Teil des Moleküls entfernt wird und die Studierenden den fehlenden Teil ergänzen müssen. Eine weitere Aufgabenvariante besteht aus einem gegebenen Molekül und dazu soll dann der Name angegeben werden.

5.1.1. Einschränkungen der Regeln

Mir wurde relativ zeitig bewusst, dass die vorhandene Zeit für die Entwicklung eines so komplexen Algorithmus nicht ausreichen wird. Deshalb haben wir (Professor & Ich) uns auf ausgewählte Regeln geeinigt. Der Algorithmus soll vorerst mit Kohlenstoff, Wasserstoff und Sauerstoff Atomen arbeiten. Von diesen drei Atomen wird, sobald ein Atom generiert wird, eins zufällig ausgewählt. Die Wahrscheinlichkeiten für die Generierung eines bestimmten Atoms sind dabei unterschiedlich, da sie in der Realität verschieden oft vorkommen. Die Anzahl der vorliegenden Bindungen zwischen den einzelnen Atomen soll zufällig sein. Dies geschieht unter Einhaltung der Valenz, die in dem Algorithmus auch das Hauptkriterium bildet. Sprich, am Ende soll ein Molekül generiert werden, dass keine offenen Bindungen mehr aufweist.

5.2. Umsetzung

Bei der Umsetzung des Algorithmus habe ich auf die Sprache Python gesetzt. Unter Verwendung von Visual Studio Code mit Jupyter Notebook und Google Colab habe ich ihn entwickelt. Als erstes habe ich die Klasse 'Atom' erstellt mit allen Attributen, die gebraucht werden, wie beispielsweise Valenz und Formel. Zusätzlich sind die Nachbaratome eines Atoms in einem Dictionary gespeichert. Dazu gibt es noch die drei Unterklassen der jeweiligen Atome, die eine Instanz der Superklasse (Atom) mit den passenden Werten erzeugen. Zusätzlich habe ich die Funktionen, mit denen man die Valenz 'getValenz()' und auch die Anzahl der freien Bindungen 'getAnzahlfreierBindungen()' eines Atoms ausgegeben bekommt, implementiert. Dazu kommen die beiden Funktionen, mit denen Nachbarn hinzugefügt und entfernt werden können, damit diese in dem Dictionary des jeweiligen Nachbarn gespeichert oder entfernt werden können.

Der Nutzer kann am Anfang eingeben, wie viele Atome ein Molekül enthalten soll (Variable: n) und wie viele Moleküle der Algorithmus berechnen und ausgeben soll. Das Molekül ist am Anfang ein leerer Array (M). Danach wird durch die Funktion 'choose_atom()' zufällig eine Instanz aus den Klassen Kohlenstoff, Sauerstoff und Wasserstoff initiiert (Variable: a). Da die Atome unterschiedlich oft in der Realität vorkommen habe ich die Wahrscheinlichkeiten in der Funktion wie folgt realisiert: Kohlenstoff 50 Prozent, Wasserstoff und Sauerstoff jeweils 25 Prozent. Daraufhin wird a zu M hinzugefügt. Nun besteht das Molekül erstmal aus einem Atom.

Die Bilder stellen eine beispielhafte Vorgehensweise des Algorithmus in der 2D-Ansicht dar, wenn für die Anzahl der Atome 5 eingegeben wurde.



Solange die Anzahl der Atome im Molekül (M) kleiner ist als die gewünschte Anzahl an Atomen (n) ist: $M < n$ wird folgendes ausgeführt:

Das Molekül wird nach Atomen mit freien Bindungen abgesucht und diese Atome werden gespeichert. Von diesen Atomen wird zufällig eines ausgewählt und es wird ein neues Atom durch *'choose_atom()'* erzeugt (Variable: b), das ausgewählte und das neu erzeugte Atom wird über k Bindungen verknüpft. k ergibt sich aus einer Zufallszahl, die zwischen 1 und dem Minimum der freien Valenz von a oder der Valenz von b liegt.

Kurzes Beispiel: a hat 2 freie Bindungen und b eine Valenz von 4. Dann würde k bei 1 oder 2 liegen. Also würden beide Atome mit 1 oder 2 Bindungen verknüpft werden.

Im folgenden Bild wird ein Sauerstoff hinzugefügt, welches $k=2$ Bindungen eingeht.

Bei beiden Atomen wird der jeweilige neue Nachbar im Dictionary mit der Anzahl der Bindungen gespeichert. Danach wird b auch zu M hinzugefügt. Nun befinden sich schon zwei Atome im Molekül.



Diese können aber trotzdem noch freie Bindungen aufweisen. Im nächsten Schritt wird geschaut, ob beide Fälle gleichzeitig eintreffen: Die Anzahl der Atome im Molekül ist kleiner als die gewünschte Anzahl und kein Atom aus M weist freie Bindungen auf. In dem Fall könnte kein neues Atom mehr hinzugefügt werden, da kein Atom mehr weitere Bindungen eingehen kann (Bild stellt dieses Szenario dar). Nun werden vorhandene Bindungen gelöst. Es wird ein Atom ausgewählt, dass nur mit genau einem anderen Atom verbunden ist. Zwischen diesen beiden Atomen werden dann (Zufallszahl zwischen 1 und vorhandenen Bindungen(k)) gelöst. Dabei kann es passieren, dass

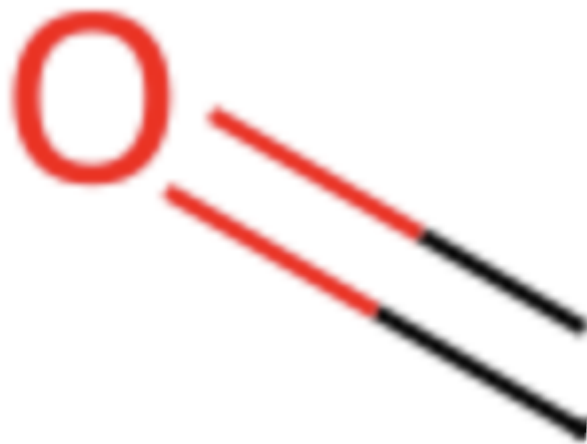
zufällig alle Bindungen zwischen den Atomen entfernt werden.

Beispiel: Eine Bindung wird entfernt



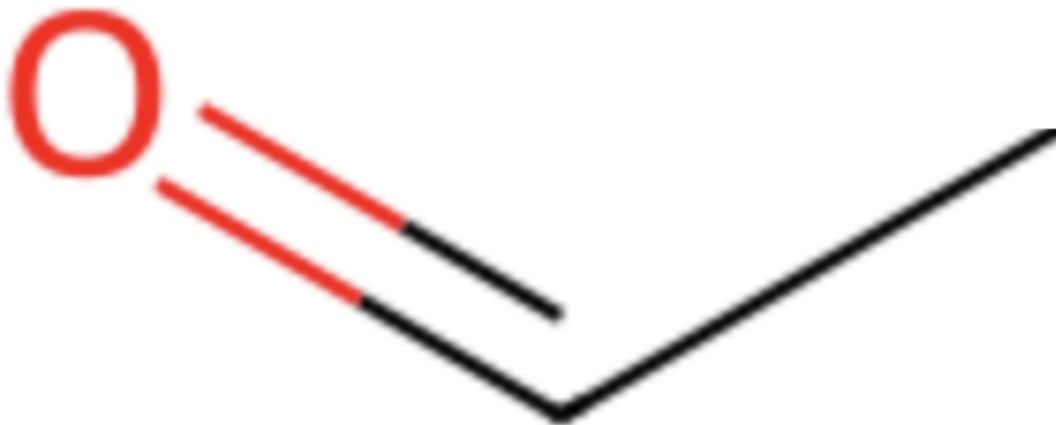
Beispiel: Alle Bindungen wurden entfernt, dann wird das Atom entfernt bzw ersetzt





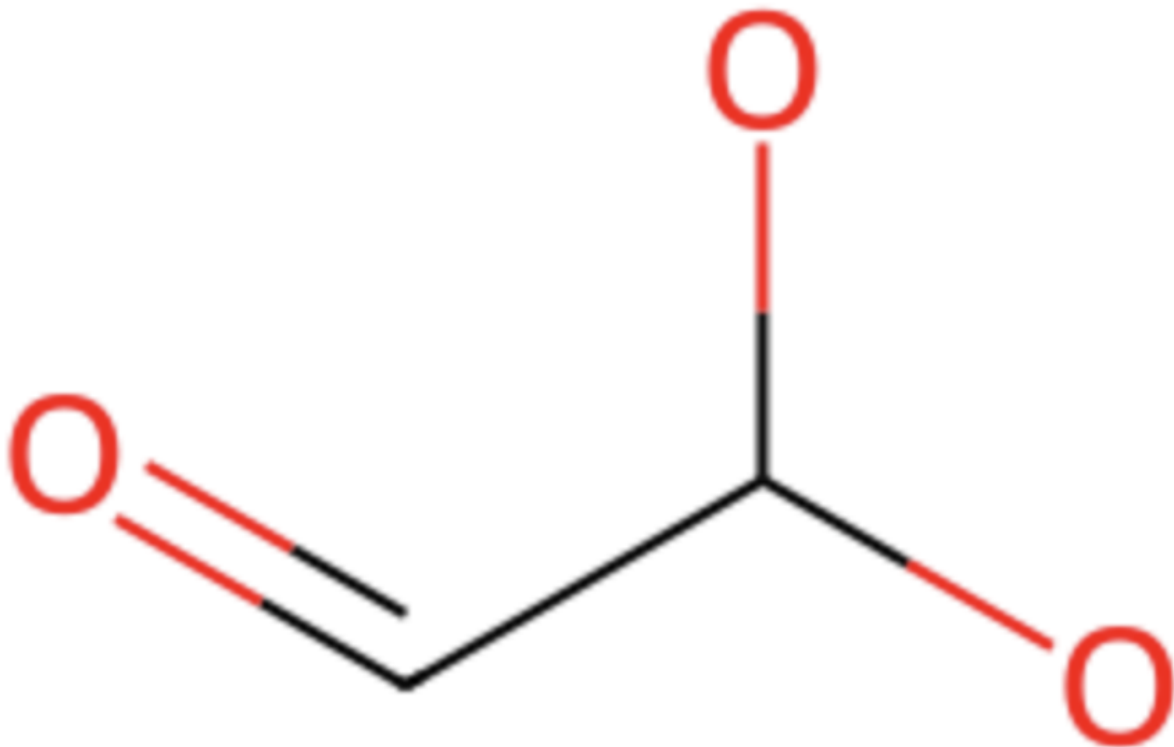
Anmerkung: C Atome werden nicht als C in der 2D-Ansicht geschrieben

Dann würde das Atom, welches ursprünglich nur zu einem Atom verbunden war aus dem Molekül entfernt werden. Hier beginnt die Schleife (Solange $M < n$) von vorn: Es wird wieder eine Instanz der Klasse C,H oder O erzeugt und über k Bindungen mit einem zufällig ausgewählten Atom verbunden..



Sobald die Schleife vollständig durchlaufen ist, haben wir ein Molekül, dass die gewünschte Anzahl an Atomen besitzt ($M = n$), aber eventuell noch freie Bindungen aufweist.

Molekül besteht aus 5 Atomen hat aber noch freie Bindungen



Diese müssen jetzt noch befriedigt werden.

Dabei geht der Algorithmus wie folgt vor: Es werden alle Atome mit freien Bindungen in einem Array gespeichert. Dazu wird zufällig ein Atom aus dem Array ausgewählt und in einem anderen Array alle Nachbarn des ausgewählten Atoms (welches freie Bindungen aufweist) gespeichert. Zwischen dem ausgewählten Atom und einem zufällig ausgewählten Nachbarn werden jetzt die Bindungen (um das Minimum von den freien Bindungen beider Atome) erhöht. Dies geschieht in der Funktion *'bindungen_erhöhen'*.

Wieder ein kurzes Beispiel: Ein ausgewähltes Atom hat 2 freie Bindungen und ein ausgewählter Nachbar hat 3 freie Bindungen → Bindungen zwischen beiden werden um 2 erhöht.

Nun sind schon einige Bindungen mehr befriedigt. Trotzdem kann es sein, dass sich immernoch Atome mit freien Bindungen im Molekül befinden.

Dann wird auf Variante 2 zugegriffen: Es wird ein zufälliges Atom mit freien Bindungen aus dem Molekül ausgewählt (a) und es wird d berechnet.

d wird wie folgt berechnet:

$$d = (\text{Valenz von a}) - (\text{freie Bindungen von a})$$

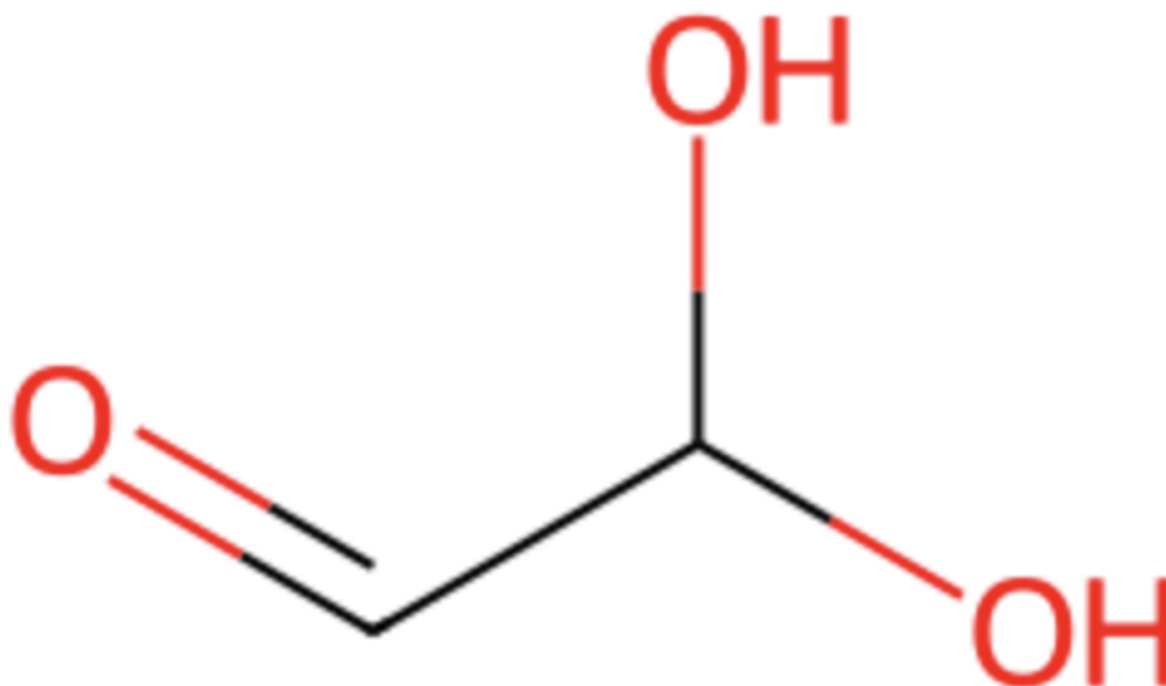
Wenn die Zahl d der Valenz von Kohlenstoff oder Sauerstoff entspricht, dann wird die Funktion *'atom_ersetzen()'* mit dem Namen des passenden Atoms als Parameter aufgerufen.

Dabei wird eine Instanz von dem passenden Atom erzeugt. Von dem zu ersetzendem Atom werden alle Nachbarn mit ihren Bindungen gespeichert und auf die neue Instanz übertragen. Das ausge-

wählte Atom wird dann aus dem Molekül entfernt und das Neue nimmt seinen Platz ein.

Wenn d nicht zu einer Valenz der zwei Atome passt dann gibt es folgende "Notlösung": Es werden d Wasserstoffatome an das ausgewählte Atom angehängen. Dies wird durch die Funktion '*wasserstoff_anhängen()*' realisiert.

Wasserstoffmoleküle anhängen



Nun hat der Algorithmus ein Molekül erzeugt, dass keine freien Bindungen mehr aufweist. Jetzt muss es nur noch dargestellt werden.

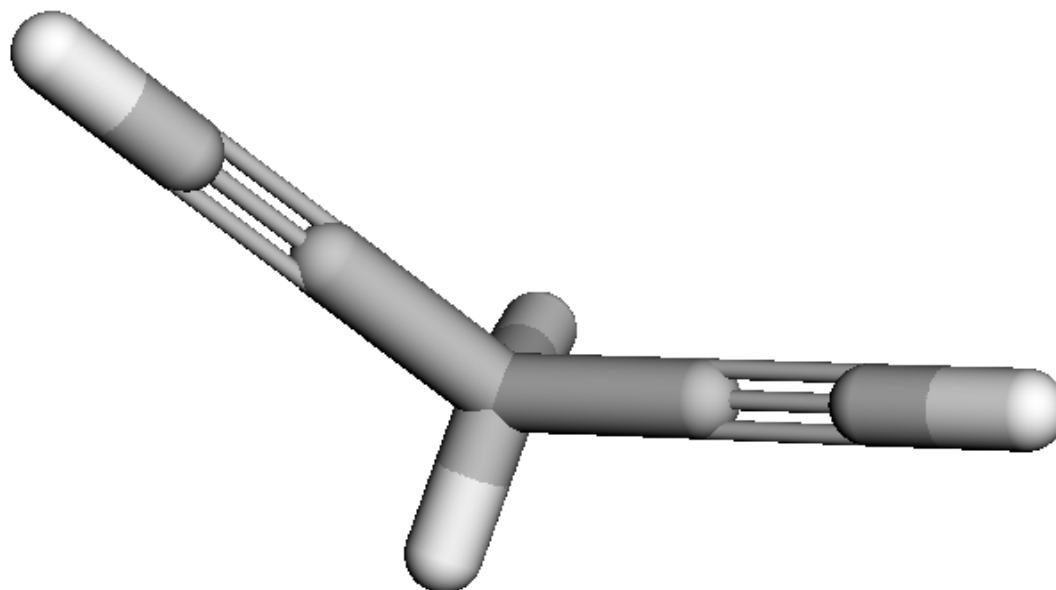
Für die Darstellung habe ich folgende Python-Bibliotheken verwendet: PubChemPy, RDKit und Py3Dmol

Für die Ausgabe wird als erstes der SMILES Code (Darstellung des Moleküls als ASCII Code) zu dem erzeugten Molekül in der Funktion '*SMILES_Code()*' berechnet. Dazu habe ich folgendes als Hilfe genutzt: <https://stackoverflow.com/questions/51195392/smiles-from-graph> Dies ermöglicht es, einen SMILES Code von einem Graphen zu berechnen. Deswegen wird in der Funktion '*create_adjacency_matrix()*' das Molekül als Adjazenzmatrix dargestellt und dann an die Funktion '*MolFromGraphs()*' weitergegeben. Mit dem Ergebnis von '*MolFromGraphs()*' als Parameter für die RDKit-Bibliothek vorgegebene Funktion '*Chem.MolToSmiles()*' wird jetzt der SMILES Code berechnet. Der SMILES Code wird nun in Verbindung mit der PubChemPy Bibliothek genutzt. In der Funktion '*get_from_PubChem()*' wird nun die Datenbank nach dem SMILES Code abgesucht und dadurch kann der Name und die Formel des Moleküls ausgegeben werden. Zusätzlich wird die Anzahl der nicht gültigen Moleküle ausgegeben, dazu aber mehr im Punkt **Probleme**.

Als letztes erfolgt die Ausgabe der 3D-Ansicht des Moleküls durch die Funktion '*show()*'. Der Funktion wird der SMILES Code als Parameter übergeben. py3Dmol stellt hier einige Funktionen bereit, mit denen Einstellungen an der 3D-Ansicht vorgenommen werden können, z.B. '*view()*' mit der ich

die Größe der Ausgabe auf 600x600 beschränkt habe.

Beispiel 3D-Ansicht für penta-1,4-diyne



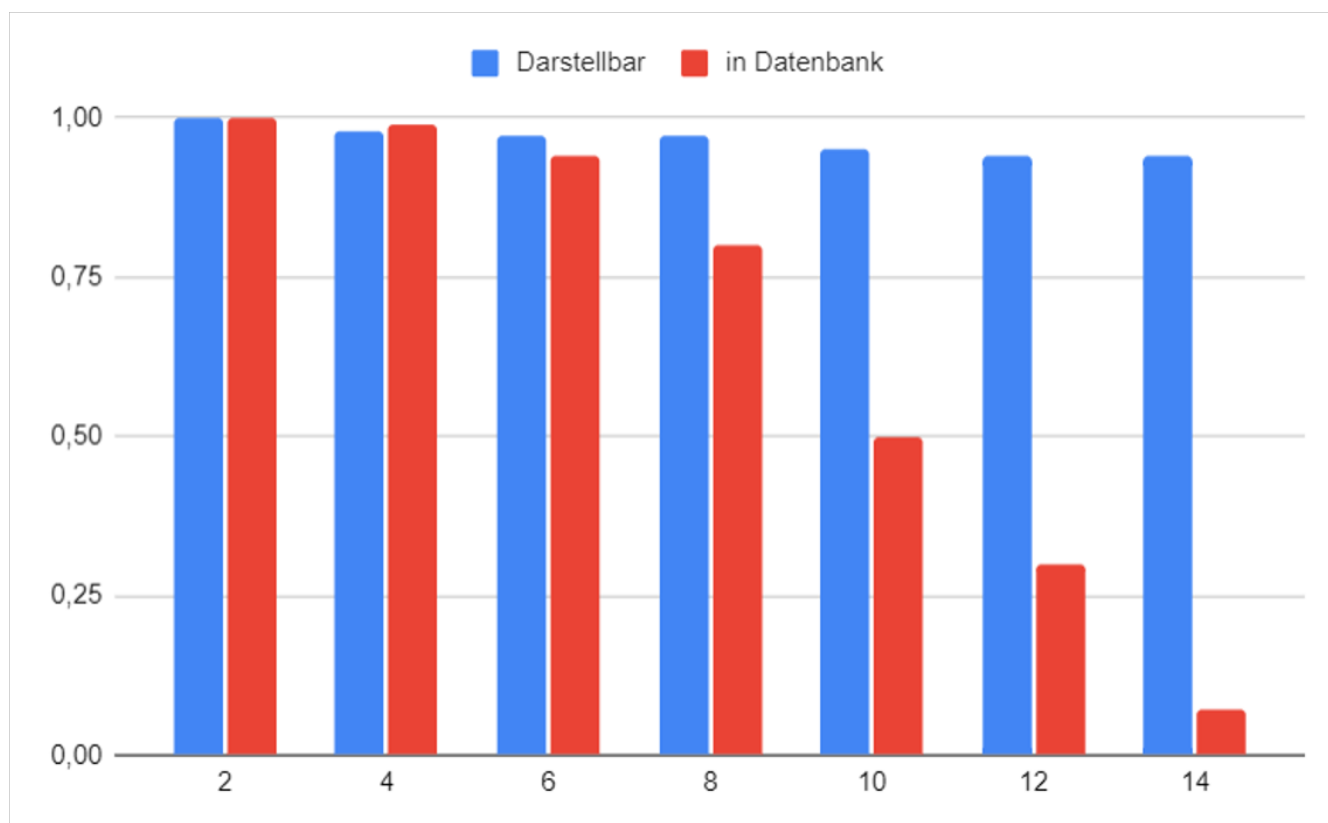
5.3. Probleme

Der Algorithmus kann zufallsbasierte Moleküle berechnen und darstellen. Doch manchmal ergeben sich dabei bestimmte Molekülstrukturen, die es theoretisch geben könnte aber dennoch in der Realität nicht existieren. Diese lassen sich dann nicht darstellen und führen zu einem Error. Deshalb gibt es eine Variable, die diese Errors mitzählt und am Ende die Anzahl der nicht gültig generierten Moleküle ausgibt. Zusätzlich wird der Name und die Formel nur aus der PubChem Datenbank abgerufen und wenn das Molekül dort nicht existiert, dann wird beides nicht ausgegeben. Je größer das Molekül ist desto unwahrscheinlicher ist es, dass der Name und die Formel in der Datenbank vorhanden ist. Genauso ist es bei der Generierung von nicht gültigen Molekülen: je größer desto wahrscheinlicher, dass es nicht gültig ist.

Bei zahlreichen Testläufen haben sich folgende Daten ergeben:

y-Achse: Wahrscheinlichkeit

x-Achse: Anzahl Atome



5.4. Ausblick

Der Algorithmus wurde in seiner Funktion durch die vorher festgelegten Einschränkungen sehr begrenzt, in Zukunft könnten aber noch weitere Regeln implementiert werden, so dass am Ende mit allen Atomen des Periodensystems gearbeitet werden könnte. Auch die Generierung von ungültigen Atomen ließe sich durch implementieren von weiteren Regeln umsetzen, denn man müsste lediglich herausfinden in welcher Struktur gewisse Atome nicht in Verbindung mit anderen Atomen existieren können und diese Möglichkeiten exkludieren. Zusätzlich zieht sich der Algorithmus die Formel und den Namen der Moleküle aus der PubChem Datenbank. Dabei treten einige Probleme auf, die ich vorher schon genannt habe. Dies ließe sich auch mit einer einzigen Funktion beheben, die beides von selber ausrechnet, da Atome oft in verschiedenen Gruppen auftreten und aus diesen Gruppen der Name sowie die Formel abgeleitet werden könnte. Außerdem ist das erst der Grundbaustein für die automatisierte Generierung von Chemie Aufgaben für Studierende.

Abschnitt 6. RSA Verschlüsselung

6.1. Aufgabenbeschreibung

6.1.1. Aufgabenstellung

Mithilfe dieses Aufgabentyps sollen Studierenden das RSA Verschlüsselungsverfahren verstehen und anwenden lernen. Durch unterschiedliche Schwierigkeitsgrade, bei denen die Studierenden stärker an die Hand genommen werden oder mehr Freiheit gegeben wird, können diese ihre Aufgaben auf jeden jeweiligen Lernstand anpassen. Zusätzlich sollen geeignete Lösungshilfen vorhanden sein, mit denen die Studierenden in der Lage sind, Schritt für Schritt die Lösungen der Aufgaben erarbeiten. Wichtig ist dabei, dass die Studierenden nicht vom Umfang der Aufgaben erschlagen werden, sondern diese in kleinteiligen Aufgabenpaketen erledigen können.

6.1.2. Aufgabenteile

1. Erzeugen von teilerfremden Zahlen
2. Bestimmen von Primzahlen
3. Generierung des öffentlichen und des privaten Schlüssels
4. Verschlüsselung von Zahlen und Text
5. Entschlüsselung von Zahlen und Text

6.1.2.1. Beispiel

RSA

gegeben= p, q , Eingabesequenz – gesucht=Chiffrat

• $p=5; q=11$; Eingabesequenz=12

1. $n=55; \phi(n)=(p-1)*(q-1)=40$
2. $e=3 \Rightarrow$ teilerfremd zu 40
3. $3 * d = 1 \bmod 40$ (erweiterter Euklidischer Algorithmus)

Erweiterter Euklidischer Algorithmus

$$e * d \bmod \phi(n) = 1$$

i	e : $\phi(n)$ =	q	r	x	y
1	3 : 40 =	0	3	-13	(nicht notwendig)
2	40 : 3 =	13	1	1	-13
3	3 : 1 =	3	0	0	1

$$x_i = y_{i-1} \quad y_i = x_{i-1} - q_i * y_{i-1}$$

$$40 - 13 = 27$$

$$\Rightarrow d = 27$$

Info:

$$x < 0 \Rightarrow d = \phi(n) + x$$

$$x \geq 0 \Rightarrow d = x$$

4. Geheimtext = Klartext^e mod n
 $12^3 \bmod 55 = 23$
5. Klartext = Geheimtext^d mod n
 $23^{27} \bmod 55 = 12$

Abbildung 1. Beispielaufgabe RSA

6.1.3. Anforderungen

- Zufällige Generierung der vorgegebenen Zahlen
- Schrittweise Anleitung
- Unterschiedliche Schwierigkeitsgrade
- Verständliche Lösungshilfen
- Vollständige Lösung
- Übersichtliche Darstellung
- Intuitive Bedienung

6.2. Umsetzung

Im Nachfolgenden möchte ich kurz auf ein paar wichtige Punkte der Umsetzung eingehen.

6.2.1. Projektverlauf

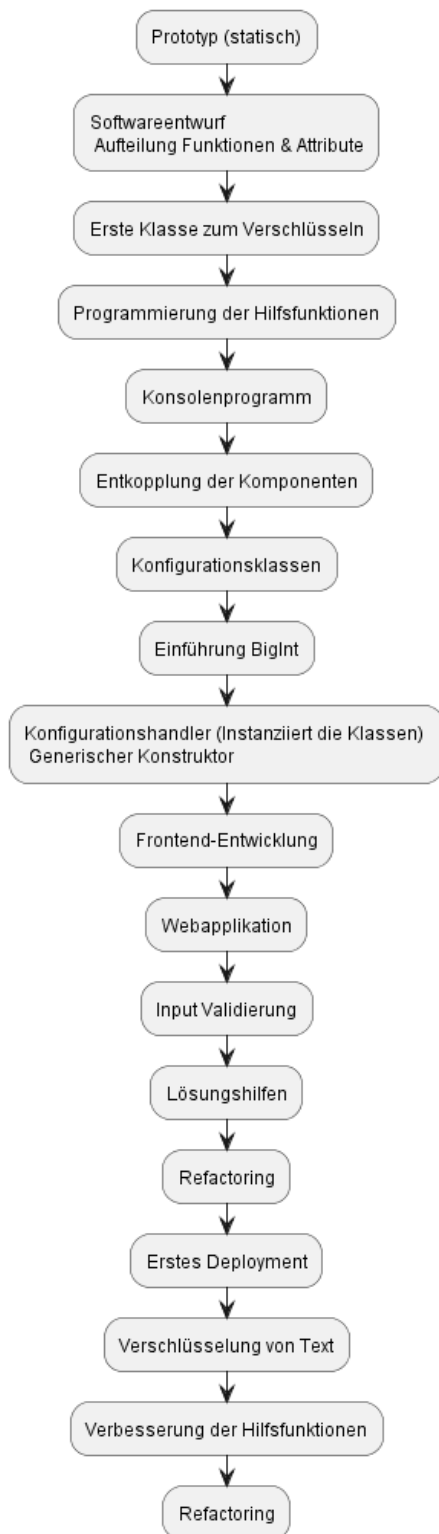
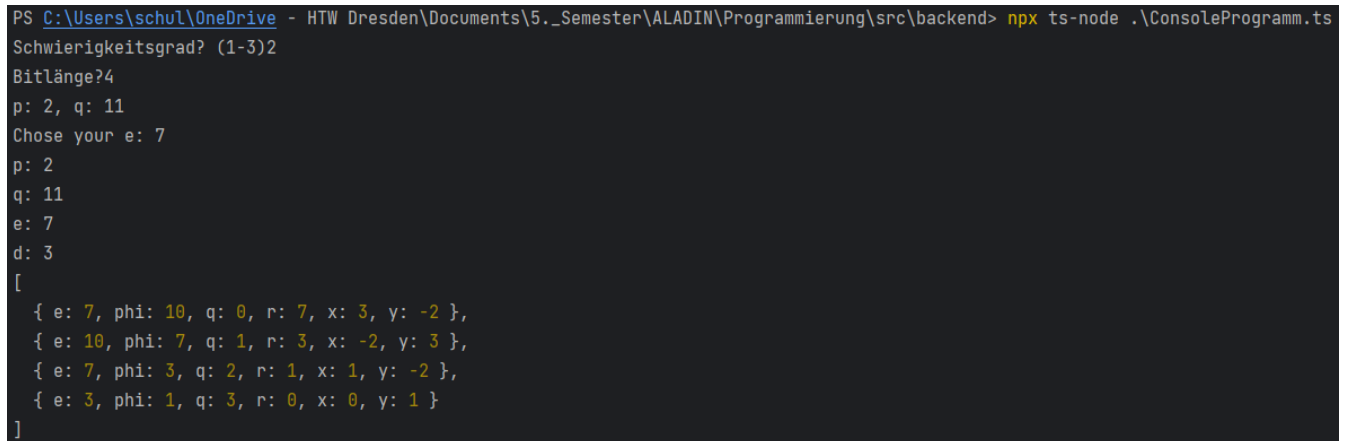


Abbildung 2. Zeitlicher Ablauf - Überblick

6.2.2. Konsolenprogramm

Mithilfe eines einfachen Konsolenprogramms sollten die Basisfunktionen getestet werden können. Die Vorteile hierbei sind zum einen, die schnellere Entwicklung, da der Fokus eher auf die Funktionen als die Darstellung gelegt wird. Außerdem wird die technische Komplexität gering gehalten, weil nur wenige externe Bibliotheken verwendet werden müssen. Im folgenden Screenshot ist die Konfiguration einer Aufgabe zu sehen, anschließend generiert das Programm die erforderlichen Werte für die Aufgabe.



```
PS C:\Users\schul\OneDrive - HTW Dresden\Documents\5._Semester\ALADIN\Programmierung\src\backend> npx ts-node .\ConsoleProgramm.ts
Schwierigkeitsgrad? (1-3)2
Bitlänge?4
p: 2, q: 11
Chose your e: 7
p: 2
q: 11
e: 7
d: 3
[
  { e: 7, phi: 10, q: 0, r: 7, x: 3, y: -2 },
  { e: 10, phi: 7, q: 1, r: 3, x: -2, y: 3 },
  { e: 7, phi: 3, q: 2, r: 1, x: 1, y: -2 },
  { e: 3, phi: 1, q: 3, r: 0, x: 0, y: 1 }
]
```

Abbildung 3. Konsolenprogramm

6.2.3. Webapplikation

Eine Webapplikation bietet mehrere Vorteile, zum einen lässt sie sich relativ schnell in andere Applikationen integrieren, auf der anderen Seite lässt sie sich mit fast allen Geräten anzeigen und bedienen. Im Vergleich zur Konsolenanwendung bietet die Webapplikation eine intuitive Bedienung und wirkt für den Benutzenden ansprechender. Hierbei liegt der Fokus weniger auf der Funktionalität, wie der Generierung der Aufgabe. Es geht eher darum, wie die Inhalte aufgeteilt werden können um ein gutes Benutzererlebnis zu gewährleisten. Die Oberfläche sollte einfach zu bedienen sein und die Studierenden sollen nicht von den Inhalten überfordert werden.

Solution aids

How to basic

How to Euclidic-Algorithm

Solution - Euclidic-Algorithm

How to - Extended Euclidic-Algorithm

Solution - Extended Euclidic-Algorithm

What about D now?

Complete solution

View solution aids

the public and private key.

N

N

i	e	$\phi(\text{phi})$	q	r	x	y
1	13	120	0	13	37	-4
2	120	13	9	3	-4	37
3	13	3	4	1	1	-4
4	3	1	3	0	0	1

Abbildung 4. Webapplikation mit Lösungshilfen

6.2.4. Input-Validierung

Nutzereingaben müssen validiert werden, bevor diese durch das Programm verarbeitet werden. Dafür wurden zwei Ansätze getestet.

Input-Validierung - separat	Input-Validierung - Bubble
<div><div>Home</div><div>Enter your configuration!</div><div>Enter your preferred difficulty</div><div>Select Difficulty</div><div>This field is required</div><div>Enter your bitlength</div><div>3-7</div><div>This field is required</div><div>Submit</div></div>	<div><div>Enter your configuration!</div><div>Enter your preferred difficulty</div><div>Select Difficulty</div><div>Please select an item in the list.</div><div>Submit</div></div>

Die Variante mit den "Bubbles" bietet bessere Barrierefreiheit, da sie in einfacher Form durch fast alle gängigen Browser unterstützt wird, allerdings auch eine komplexere und verteilte Konfiguration. Hierfür wurde ein separates Modul geschrieben, welches eine übersichtliche und gesammelte Konfiguration der Felder ermöglicht.



Mehr zum Softwareentwurf und zur Konzeption ist unter der [Architektur](#) zu finden.

6.2.5. Fazit zur Umsetzung

Bei der Umsetzung galt es einiges zu beachten, um die obigen [Anforderungen](#) zu erfüllen und eine möglichst gute Softwarequalität zu erreichen.

Mir war es wichtig, dass die Software möglichst flexibel bleibt und konfiguriert, statt hartcodiert wird. Dafür mussten geeignete Konzepte, wie dem Konfigurationshandler, entwickelt werden. Auch die Verwendung von generischen Konstruktoren war mir neu und hat zu einer besseren Code-Qua-

lität beigetragen.

Die Verwendung von TypeScript hat mir eine bessere Objektorientierung ermöglicht, jedoch verursacht das strikte Typsystem auch einen Mehraufwand, welcher in eine saubere Lösung investiert werden muss. Allerdings ist die Software dadurch nicht mehr so anfällig für Fehler, da die meisten spätestens beim Kompilieren auffallen und behoben werden müssen.

Generell musste ich die Software mehrfach refactoren, da sich neue Zusammenhänge ergeben haben und auch der Wechsel vom Konsolenprogramm zur Webapplikation größere architekturbedingte Veränderungen mit sich gebracht haben.

Die Konzipierung und Programmierung übersichtlicher und verständlicher Lösungshilfen, ist essenziell, da die Studierenden vor allem durch diese vorangebracht werden und einen Lernerfolg erzielen. Deshalb habe ich probiert, die Lösungshilfen so verständlich und übersichtlich wie möglich zu gestalten. Feingranular und Schritt für Schritt werden Methoden, wie der "Erweiterte Euklidische Algorithmus" erklärt und beispielhaft dargestellt.

6.2.5.1. Schlussfolgerungen für den Quellcode

1. Generische Konstruktoren
2. Verwendung eines Konfigurators (Konfigurationhandler), welcher entscheidet, welche Klassen instanziiert werden
3. Mappings um komplexe `if-else`-Konstrukte oder `switch-case` zu vermeiden

6.2.6. Verwendete Technologien

1. Vite (Development-Server und Build Tool)
2. ReactJS (JavaScript-Programmbibliothek zur Erstellung von webbasierten Benutzeroberflächen)
3. TypeScript (Superset von JavaScript mit Typsystem und besserer Objektorientierung)

6.2.7. Architektur

Im Idealfall sollte die Webapplikation aus einem Server bestehen, auf welchem die Aufgaben generiert werden und welcher die Rechenlast für die Verschlüsselungsaufgaben trägt und einem Client. Der Client würde dann nur die Aufgaben darstellen und mit dem Server kommunizieren, um weitere Inhalte zu laden oder die Lösung abzugleichen. Aktuell ist es so, dass die gesamte Logik im Client ausgeführt wird (FatClient) eine Aufteilung in den klassischen Server-Client ist noch nicht erfolgt. Somit ist die Geschwindigkeit der Applikation wesentlich von den Ressourcen des Clients abhängig.

6.2.7.1. Backend

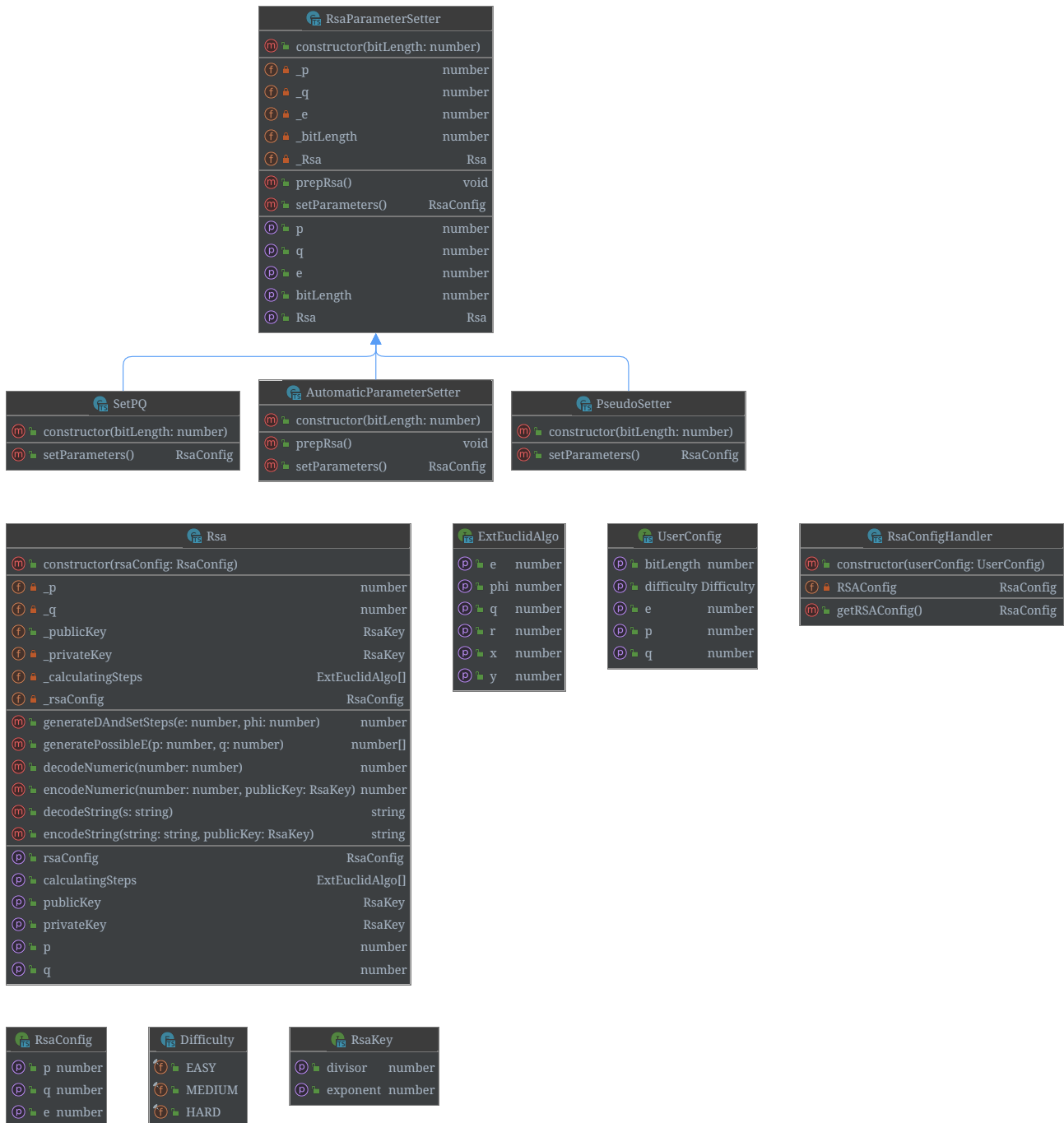


Abbildung 5. Programmaufbau - Backend

6.2.7.2. Frontend

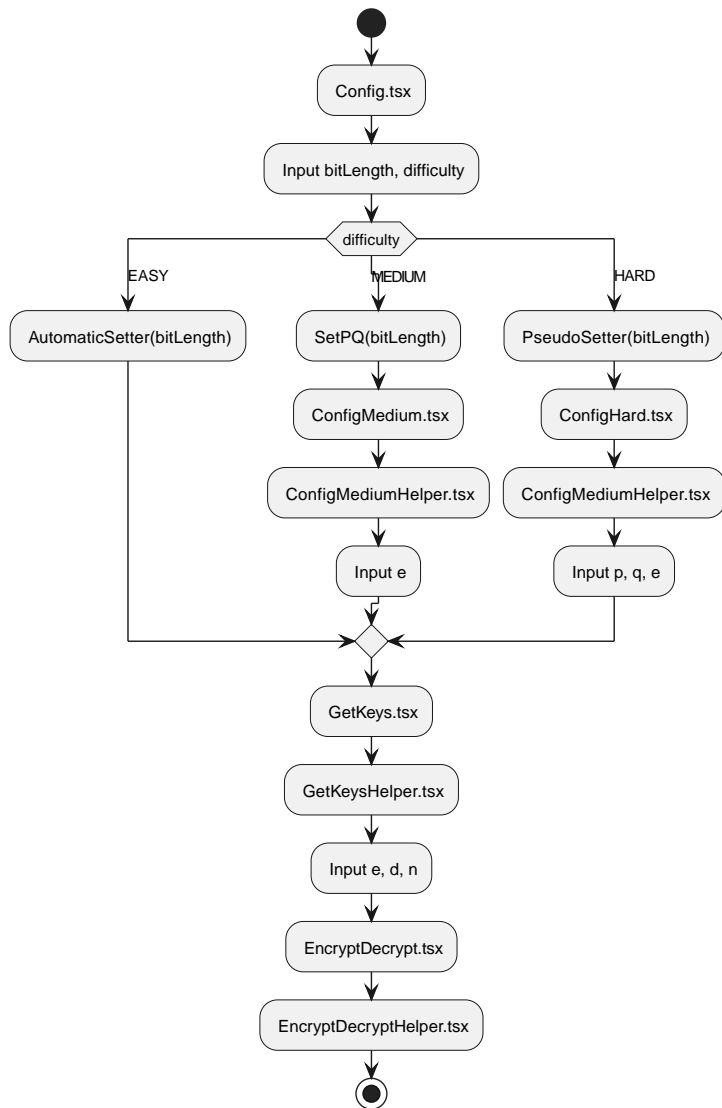


Abbildung 6. Programmablauf - Frontend

6.3. Probleme

- Geschwindigkeit der Applikation ist von der Rechenleistung des Clients abhängig, siehe [Architektur](#)
- Ver-/Entschlüsselung lief mit dem Datentyp `number` nicht immer zuverlässig, da hierfür die Zahlen zu groß und abgeschnitten wurden ⇒ Einführung `BigInt`
- Größe des `BigInt` in TypeScript

Die Verschlüsselung von Text war ursprünglich mittels Byte-Stream geplant. Dadurch entstehen allerdings sehr schnell sehr große Zahlen und es werden immer größere Schlüssel benötigt. Leider waren diese in der gewählten Programmiersprache (TypeScript) nicht darstellbar.

Tabelle 2. Probleme mit der Größe des `BigInt`

<pre>Rsa { _rsaConfig: { p: 3023, q: 3037, e: 5 }, _p: 3023, _q: 3037, _publicKey: { divisor: 9180851, exponent: 5 }, _calculatingSteps: [{ e: 5, phi: 9174792, q: 0, r: 5, x: 3669917, y: -2 }, { e: 9174792, phi: 5, q: 1834958, r: 2, x: -2, y: 3669917 }, { e: 5, phi: 2, q: 2, r: 1, x: 1, y: -2 }, { e: 2, phi: 1, q: 2, r: 0, x: 0, y: 1 }], _privateKey: { divisor: 9180851, exponent: 3669917 } } Ausgangstext: ALA Text als Bytestream: 010000010100110001000001 Text als Dezimalzahl: 4279361 Verschlüsselt dezimal: 8568020 Verschlüsselt binär: 100000101011110011010100 Entschlüsselt binär: 010000010100110001000001 Entschlüsselt Text: ALA</pre>	<pre>Rsa { _rsaConfig: { p: 10000019, q: 10000079, e: 41 }, _p: 10000019, _q: 10000079, _publicKey: { divisor: 100000980001501, exponent: 41 }, _calculatingSteps: [{ e: 41, phi: 100000980001404, q: 0, r: 41, x: 4163812683509, y: -17 }, { _privateKey: { divisor: 100000980001501, exponent: 4163812683509 } }] } Ausgangstext: ALADIN Text als Bytestream: 0100000101001100010000010100010010010011001110 Text als Dezimalzahl: 71795708314190 Verschlüsselt dezimal: 54250818288190 Verschlüsselt binär: 001100010101011101000000110000001011101000111110 F:\HN - Stick\5_Semester\ALADIN\Programmierung\src\backend\Rsa.ts:133 return Number(BigInt(number) ** BigInt(this.privateKey["exponent"]) % BigInt(this.privateKey["divisor"])); ^ RangeError: Maximum BigInt size exceeded</pre>
--	--

Von nun an erfolgt die Verschlüsselung von Text Buchstabe-für-Buchstabe. Vorteil dieser Variante, ist es, dass es für die Studierenden leicht zu verstehen und einfacher in der Anwendung ist.

Schritt für Schritt
Text: ALADIN
Byte: 1000001 1001100 1000001 1000100 1001001 1001110
Dezimal: 65 76 65 68 73 78
Verschlüsselt dezimal: 285 32 285 217 156 199
Verschlüsselt text: ħ ħÇ
Entschlüsselt dezimal: 65 76 65 68 73 78
Entschlüsselt text: ALADIN

Abbildung 7. Verarbeitung des Textes Schritt für Schritt

- Nicht darstellbare ASCII-Zeichen, z.Bsp. Backslash, Linefeed

Einige ASCII-Zeichen lassen sich nicht wirklich darstellen, diese haben in meinem Programm eine eigene Darstellung bekommen = [*]. Für die Verwertung dieser Strukturen mussten komplexere Funktionen geschrieben werden, welche die Zeichenketten aus dem gesamten String extrahieren und umwandeln konnten.

ASCII Table

Decimal	Character	Description	Decimal	Character	Description
0	[NUL]	null character	33	!	exclamation mark
1	[SOH]	start of heading	34	"	double quotation mark
2	[STX]	start of text	35	#	number sign
3	[ETX]	end of text	36	\$	dollar sign
4	[EOT]	end of transmission	37	%	percent sign
5	[ENQ]	enquiry	38	&	ampersand
6	[ACK]	acknowledge	39	'	apostrophe
7	[BEL]	bell	40	(left parenthesis
8	[BS]	backspace	41)	right parenthesis

Abbildung 8. ASCII-Tabelle

- Wertebereich für das Ver-/Entschlüsseln von Text

Es sollen ausschließlich ASCII-Zeichen und keine Unicode Zeichen angezeigt werden. Einige Unicode-Zeichen sind vor allem für deutschsprachige Studierende schwer zu deuten (asiatische Schriftzeichen). Deshalb werden immer dieselben Schlüssel für das Ver-/Entschlüsseln von Text verwendet.

```
const rsaString = new Rsa({p: 3, q: 43, e: 5});
```

6.4. Ergebnisse

Zusammenfassend kann man sagen, dass sich mit dem Programm zufriedenstellend Aufgaben und entsprechende Lösungshilfen generieren lassen. Die Aufgabenstellungen können individuell an den jeweiligen Lernfortschritt angepasst werden und unterstützen damit die Studierenden beim Lernen. Somit wurden die [Anforderungen](#) erfüllt.

Schlussendlich habe ich die aktuelle Version des Projektes [hier](#) veröffentlicht.

ALADIN

Enter your configuration!

Enter your preferred difficulty

Select Difficulty ▼

Enter your bitlength

2 - 10 ↕

Submit

Abbildung 9. Finale Version

Die ausführliche Softwaredokumentation findet man [hier](#). Interessant ist beispielsweise der Aufbau der [RSA Klasse](#) und dessen zugehörige Funktionen. Aber hier findet man auch die sonstigen Klassen und Funktionen, welche für die Applikation benötigt werden.

ALADIN Cryptography

ALADIN Cryptography / backend/rsaCryptography/Rsa / Rsa /

Class Rsa

Hierarchy

- Rsa

Defined in [backend/rsaCryptography/Rsa.ts:17](#)

▼ INDEX

Constructors

- Ⓒ constructor

Properties

Ⓐ _calculatingSteps	Ⓐ _p	Ⓐ _privateKey
Ⓐ _publicKey	Ⓐ _q	Ⓐ _rsaConfig

Accessors

Ⓐ calculatingSteps	Ⓐ p	Ⓐ privateKey
Ⓐ publicKey	Ⓐ q	Ⓐ rsaConfig

Methods

Ⓜ decodeNumeric	Ⓜ decodeString	Ⓜ encodeNumeric
Ⓜ encodeString	Ⓜ generateDAndSetSteps	Ⓜ generatePossibleE

▼ Settings

MEMBER VISIBILITY

- ☒ Protected
- ☒ Private
- ☒ Inherited

THEME Light ▼

▼ Modules

- ALADIN Cryptography
 - backend/rsaCryptography/Difficulty
 - backend/rsaCryptography/ExtEuclidAlgo
 - backend/rsaCryptography/GeneratePrimes
 - backend/rsaCryptography/GetRandomInt
 - backend/rsaCryptography/HasCommonDivider
 - backend/rsaCryptography/IsPrime
 - backend/rsaCryptography/Rsa
 - backend/rsaCryptography/RsaConfig
 - backend/rsaCryptography/RsaConfigHandler
 - backend/rsaCryptography/RsaKey

Abbildung 10. Dokumentation RSA-Klasse

6.5. Ausblick

1. Erweiterung um Textkomprimierung z.Bsp. Huffman-Code

Zusammen mit der Textkomprimierung und dem RSA-Verschlüsselungsverfahren könnte sich eine Komplexaufgabe gestalten lassen.

2. Interaktive Lösungshilfen

Studierende könnten direkt in den Lösungshilfen, z.Bsp. Tabellen ausfüllen und diese auch als Lösung einreichen.

3. BackEnd mit REST-Schnittstellen

Klare Trennung und bessere Ressourcennutzung.

4. Programm um einen Prüfungsmodus erweitern

Denkbar sind eine abschließende Auswertung und Bewertung, wie gut die Aufgaben absolviert wurden und wie viele Fehler gemacht wurden.

5. Zwischenschritte der Studierenden mitspeichern

Dadurch wird es möglich, die gleiche Aufgabe nochmal mit Kommilitonen und Lehrenden durchzugehen, falls Verständnisfragen aufkommen.

Abschnitt 7. Reflexion

Mit der Reflexion, als wichtigen Teil einer Projektarbeit, möchten wir beleuchten, wie die Projektarbeit für jeden Teilnehmer war. Dabei stellt jeder Bearbeiter seine Entwicklung und seine Learnings aus dem gesamten Projekt da, um ein gezieltes Feedback für die Arbeit zu erbringen.

7.1. Timm Hauschild

In dem Projekt Aladin konnte ich wertvolle Erfahrungen sammeln, die mich in meiner persönlichen und beruflichen Weiterentwicklung unterstützt haben. Insbesondere habe ich gelernt, wie man komplexe Algorithmen mit Java erstellt und wie man dabei Klassendiagramme und Programmlaufpläne verwendet. Diese Fähigkeiten werden in der Softwareentwicklung sehr geschätzt und sind ein wichtiger Bestandteil der Programmierung.

Darüber hinaus habe ich meine Fähigkeiten in der Arbeit mit Java-Listen weiter ausgebaut und konnte meine Kenntnisse in der Vererbung von Klassen vertiefen. Dies ist eine wichtige Fähigkeit, die in vielen Projekten benötigt wird und mir in Zukunft sicherlich von großem Nutzen sein wird.

Ein weiterer wichtiger Bestandteil des Projekts war die Arbeit mit ChatGPT, einem Sprachmodell, das in der Lage ist, menschenähnliche Texte zu generieren. Ich war sehr beeindruckt von den Möglichkeiten, die ChatGPT bietet, und sehe großes Potenzial für die Verwendung dieses Modells in verschiedenen Anwendungsbereichen.

Ein weiteres wichtiges Learning war die Bedeutung der Zusammenarbeit mit Kunden und der Berücksichtigung ihrer technischen Expertise. Ich habe gelernt, wie wichtig es ist, den Kunden frühzeitig in den Entwicklungsprozess einzubeziehen und auf seine Bedürfnisse und Anforderungen einzugehen. Durch die Zusammenarbeit mit dem Kunden konnte ich wertvolle Einblicke gewinnen und meine Fähigkeiten in der Softwareentwicklung weiter verbessern.

7.2. Ruben-David Kraus

Retrospektiv betrachtet, empfinde ich die Arbeit am Projekt ALADIN als wertvolle Erfahrung im Studium. Durch das eigenverantwortliche Arbeiten und die freie Auswahl der Themengebiete, hatte ich das Gefühl stets die Aufgabe zu bearbeiten, die mich persönlich begeistert. Durch die wöchentlichen Meetings, fand auch ein reger Austausch zwischen den einzelnen Bearbeitenden statt und es konnten wertvolle Ideen untereinander ausgetauscht werden. Außerdem war dadurch kontinuierliches Feedback durch die Lehrenden möglich, was stets zum erfolgreichen Bearbeiten der Aufgabe beigetragen hat.

Im Hinblick auf mein technisches Know-How konnte ich mich in Richtung aktueller Webentwicklung und Backend weiterentwickeln. Ich habe erfolgreich Vuejs, ein Javascript Framework, angewendet und meine Kenntnisse im Framework, sowie in Javascript entscheidend erweitert. Im Backend konnte ich meine Python Kenntnisse weiter ausbauen und FastAPI als neues Framework erlernen. Architektonisch war es herausfordernd, aber auch spannend, eine eigene API zu entwickeln und diese erfolgreich an das Frontend anzubinden. Dabei habe ich viel über Kommunikation über REST-Schnittstellen gelernt und mein bereits vorhandenes Wissen angewendet.

Das natural language processing, als neues Fachgebiet der Informatik, hat mich in diesem Projekt

am meisten gereizt. In diesem Gebiet erste Erfahrungen zu sammeln, hat mir persönlich viel Erkenntnisse gebracht. Der damit verbundene Einsatz von KI Technologien, auch mit ChatGPT, war überaus spannend. Weiterhin möchte ich mein Wissen in diese Richtung ausbauen.

7.3. Alexander Schulz

Rückblickend habe ich im ALADIN Projekt sehr viel lernen können. Zu einem konnte ich mein Verständnis für objektorientierte Programmierung sehr stark verbessern. Bisher hatten wir im Studium noch keine komplexeren Einzelprojekte in dieser Größenordnung. Ich habe gelernt guten Code mit wenigen Abhängigkeiten zu schreiben und habe diesen permanent für Nachfolger dokumentiert. Außerdem habe ich mit vielen neuen Bibliotheken gearbeitet und so gesehen, wie Technologien integriert werden können und sich Schritt für Schritt über mehrere Stufen eine Software entwickeln kann.

Doch auch fachlich hat sich das Projektseminar ausgezahlt, so konnte ich tief in aktuelle Verschlüsselungsverfahren eintauchen und auch die Unterschiede zwischen diesen erkennen, während ich das RSA-Verfahren implementierte. Viel Aufwand habe ich in die Entwicklung der Benutzeroberfläche gesteckt und teilweise war es nicht so einfach zu entscheiden, wie bestimmte Sachverhalte erklärt und dargestellt werden sollten, damit der Endnutzer das Programm intuitiv bedienen kann und die Aufgaben und Lösungshilfen versteht.

Besonders gut hat mir die wöchentliche Vorstellungs- und Austauschrunde gefallen. Hier konnte man seine Probleme präsentieren und von den Erfahrungen der anderen Teilnehmer profitieren, was mir persönlich sehr bei der Entwicklung geholfen hat. Außerdem war es immer sehr interessant zu sehen, wie die anderen ihre Aufgaben bewerkstelligten und welche Technologien dabei zum Einsatz kamen. Bemerkenswert fand ich die Verwendung des Sprachmodells bei Ruben, mit welchem er seine Aufgabenstellung generieren lässt. Dabei wurde mir klar, wie vielseitig die KI jetzt schon eingesetzt werden kann und wie man selbst diese in eigene Projekte integrieren könnte. Spannend fand ich auch die Entwicklung des Algorithmus für die EPK-Generierung und wie damit nun Geschäftsprozesse generiert werden können.

7.4. Dustin Doege

Zusammenfassend kann ich sagen, dass ich mit dem erreichten Stand der Software zufrieden bin, obwohl nicht alle Anforderungen zu 100% erfüllt wurden. Während des Projekts habe ich viel Zeit investiert, viele neue Methoden kennengelernt und mein Wissen im Bereich der Softwareentwicklung erweitert.

Insbesondere in der Java-Programmierung habe ich große Fortschritte gemacht, auch wenn ich im Rückblick einige Features anders umsetzen würde. Ich bin auch mit dem Vaadin-Framework vertraut geworden, welches mir zu Beginn des Projekts noch fremd war, und werde dessen Verwendung in meiner zukünftigen beruflichen Laufbahn berücksichtigen.

Ich habe die wöchentlichen Meetings genossen, da sie mir geholfen haben, meine Anwendung weiterzuentwickeln und man stetig neue Dinge gelernt hat. Es war interessant zu sehen, was meine Kommilitonen im Verlauf des Projekts erreicht haben. Allerdings hätte ich mir gewünscht, dass die Teamarbeit in diesem Projekt stärker betont worden wäre.

Ich bin dankbar für die Zeit im Projektseminar und gehe mit vielen neuen Kenntnissen und Erfahrungen aus diesem Projekt.

7.5. Philip Poplutz

Bei der Arbeit am ALADIN Projekt habe ich einiges an Kenntnissen dazugewonnen. Am Anfang war ich von der Aufgabe überwältigt, aber durch die wöchentlichen Meetings und die kleinen Ziele, die ich mir jede Woche gesetzt habe, gelang mir das Arbeiten immer strukturierter, schneller und einfacher. Es war mein erstes Code-Projekt in dieser Größenordnung, weswegen ich rückblickend einiges anders gemacht hätte, aber ich bin dennoch zufrieden mit dem Ergebnis. Zusätzlich war es mein erstes Projekt mit Python. Beim Lernprozess habe ich festgestellt, dass Python eine Sprache ist, die ich mir sehr gut selbst beibringen kann. Python zu lernen ist ein wichtiger Schritt für meine Zukunft gewesen und ich werde ab jetzt weiter an Python-Projekten arbeiten.

Außerdem fand ich es sehr gut, dass jeder wöchentliche Updates zu seinem Projekt gegeben hat. Natürlich konnte man dabei nicht immer alles zu 100% verstehen, dennoch haben alle von ihren Fehlern und Erfolgen berichtet, was mir definitiv bei der Weiterentwicklung geholfen hat. Nebenbei habe ich durch das Projekt auch noch meine Chemiekenntnisse aufgefrischt.

Insgesamt war das Projektseminar eine große Bereicherung für mich und auf viele dadurch erlernte Skills werde ich in Zukunft zurückgreifen.

Abschnitt 8. Schluss

Mit dem erfolgreichen Abschluss des Projektes, bleibt immer noch viel Raum für Verbesserungen oder auch neue Projekte innerhalb des ALADIN-Frameworks. Durch unsere Arbeit, um neue Sachgebieten der Informatik in ALADIN einzubringen, haben wir gute Grundsteine für eine Weiterentwicklung gelegt. Mit dem intensiven Auseinandersetzen bieten sich uns aber auch neue Ideen, welche in ALADIN integriert werden können.

8.1. Folgeprojekte

8.1.1. No-Code

Durch die Aktualität von No-Code Plattform und deren Vorteile für schnellen Einsatz von Software, ist auch eine No-Code Lösung für ALADIN denkbar. Hierbei wäre es möglich, ein Tool zu bauen um einheitliche Aufgaben für beliebige Disziplinen anzubieten. Dabei können sich Nutzer eigene Themengebiete erstellen und die Aufgaben einfach über die No-Code Plattform mit einem Editor zusammenbauen. Mit dieser Lösung ließe sich eine einheitliche Aufgabengenerierung schaffen, welche aber in ihrer Anpassbarkeit stärker eingeschränkt ist.

8.1.2. Erweiterung der Fachbereiche

Die Informatik bietet ein großes Spektrum an Fachbereichen, welche noch nicht in ALADIN angeboten werden. Eine Erweiterung ist hier also immer möglich und sinnvoll. Vorstellbar sind weiterhin Disziplinen aus dem Studienablauf der Informatik:

- Vertragsrecht
- Geschäftsprozessmodellierung
- Zahlensystemumrechnung
- Dijkstra Algorithmus
- Aussagenlogik
- Statistik
- Graphentheorie
- Zeichenkodierung / Zeichendarstellung

8.1.3. Integration von Feedback-Mechanismen

In ALADIN wäre es ebenfalls von Nutzen, Feedback-Mechanismen für die Aufgaben einzubauen. Mit diesem Feedback können die Aufgabenerstellung sowie die Lösungen besser evaluiert werden, um die Qualität von ALADIN dauerhaft zu steigern. Außerdem lässt sich aus dem Feedback erkennen wie die Benutzer mit den generierten Aufgaben und Lösungshilfen interagierten.

8.1.4. Lernpfade

Um in einem Fachgebiet fundierte Kenntnisse zu erwerben, gibt es, ähnlich wie in einer Vorlesung, eine Reihenfolge in der man sich Wissen aneignen sollte. Diese Reihenfolge kann in Form eines

Lernpfades umgesetzt werden, welcher eine Folge von Aufgaben und Lösungshilfen darstellt. Dabei wird dem Nutzer eine Reihenfolge vorgegeben und die Frustration, durch einen fehlenden Roten Faden, genommen.

8.1.5. Integration von KI-Elementen

Durch den gezielten Einsatz von KI, kann auch das ALADIN-Framework profitieren. In ALADIN könnte ein KI-Einsatz zum Beispiel bedeuten, dass die gelösten Aufgaben von der KI ausgewertet werden, oder neue Aufgaben durch die KI generiert werden. Außerdem kann das Feedback der Nutzer, wenn bereits integriert, ausgewertet werden um die Qualität der Aufgaben zu erhöhen, aber auch die Lösungshilfen könnten dynamisch mit KI-Einsatz generiert werden.