Verteiltes Genom Browsing

Projektspezifikation

23. Dezember 2015

Inhaltsverzeichnis

1 Integration

1.1 Integrationsprozess

1.1.1 Ablauf der Integration

Die Software wird standardmäßig zwei Quellen integriert haben: die dbSNP und das 1000GenomeProject. Diese werden, über ein Skript gesteuert, heruntergeladen. Wenn dieser Part abgeschlossen ist, wird ein lokaler Parser (später mehr zu den Parsern und ihrer Arbeitsweise) alle relevanten Daten aus den heruntergeladenen Dateien extrahieren und sie in einem, für den globalen Parser akzeptablen Format bereitstellen. Der globale Parser wird diese Dateien dann in Datensätze staffeln und diese dann in das Data Ware House einfügen um sie dann so der Middleware für die Abfragen bereitzustellen.
Der lokale Parser wird benötigt, da es kein allgemeingültiges Standardformat gibt, in dem die Dateien abgespeichert werden. Die Formate sind auch keine, von dem DWH anerkannten, Dateiformate, die integriert werden können. Hinzu kommt, dass Sehr viele weitere Daten, die nicht notwendig sind, in den Dateien vorhanden sind und somit nur die benötigten Daten ausgelesen werden müssen.
Wenn der User weitere Quellen integrieren möchte, benötigt er zwei Dinge dafür: Ein Downloadskript, welches von der gewünschten Quelle die Dateien herunterlädt, und ein lokaler Parser, der die heruntergeladenen Dateien danach für den globalen Parser bereitstellt.

1.1.2 Quellenauswahl

Die Quellen, die von uns integriert sein werden, werden die dbSNP und das 1000GenomeProject sein, da diese alle benötigten Daten frei zugänglich bereitstellen, ohne jegliche Gegenleistung zu verlangen. HGMD hingegen können wir nicht integrieren, da für die Benutzung ihrer Daten eine Lizenz erworben werden muss, die den finanziellen Rahmen des gesamten Projektes sprengt. TCGA konnten wir bei unserer anfänglichen Quellensichtung als "nicht nutzbar"für das Projekt einstufen, da es weder eine Angabe auf ein Referenzgenom gibt, auf das sich die Daten beziehen, noch gibt es Metadaten für die Mutationen. Sollte sich noch eine Möglichkeit ergeben, die Quelle doch integrieren zu können, wird das geschehen, jedoch werden wir uns vorerst auf die frei zugänglichen Quellen konzentrieren, um eine Basis an Daten bereitstellen zu können, weitere Quellen werden im Nachhinein integrierbar sein, weshalb diese Option jederzeit offen steht.

1.1.3 Attributauswahl und -mapping

Beide Basis-Datenbanken werden von dem gleichen Anbieter gehostet, was beiden Quellen ein relativ uniformes Format gibt. So stellen beide Quellen die Daten in BAM-Dateien und vcf-Dateien bereit, wobei wir uns auf die vcf-Dateien berufen, da diese die Daten beinhalten, die wichtig sind für die Abfragen, und auch gleichzeitig einfacher zu entschlüsseln sind, als die BAM-Dateien.
Wir entschieden uns, dass das Referenzgenom in einer Extradatei abgespeichert wird,

um übermäßigen Traffic im DWH zu vermeiden. Des weiteren werden im DWH folgende Daten der Middleware bereitgestellt: Eine einzigartige ID für jeden Eintrag, die Mutationssequenz selber, die Koordinaten, wo im Referenzgenom die Mutation auftritt, der Name der Quelle, um eine schnelle Einordnung nach Quellen zu gewährleisten, der Name des Referenzgenoms, eine Angabe, in welchem Chromosom die Mutation auftritt, und einen Verweis auf den entsprechenden Metadatensatz der Mutation.
Der Metadatensatz besteht aus einer eindeutigen ID, der Quelle, woher die Daten stammen, dem Geschlecht des Testsubjekts, dem Herkunftsland und der Downloadzeit. Wir entschieden uns für diese Metadaten, da sie in unseren Standard-Quellen vorhanden sind, und jede nutzbare weitere Quelle auch zumindest diese Daten bereitstellen wird. Die vorhandenen Dateien beherbergen noch eine Fülle weiterer Metadaten, die aber von nur geringer Bedeutung für den eigentlich Zweck unseres Programms sind.

1.1.4 Mengengerüst

1.1.5 Inputfile-Format vom lokalen Parser erzeugt

Referenzgenomname:„Name des Referenzgenoms Bsp: GRCh38“
Quelle:„hier die Quelle angeben“
$$
SampleID:„SampleID aus der DB“
Genkoordinaten:„Angabe der Koordinaten“
Mutationssequenz:„Sequenz“
$$
SampleID:„SampleID aus der DB“
Gender:„m oder f“
Population:„drei Buchstaben bsp: GBR“
EOF

Der lokale Parser wird für jeden Datensatz einen Eintrag in dieser Art in sein Inputfile schreiben, so dass er, nach erfolgreichem Parsen eine Textdatei mit entsprechend vielen Einträgen dieser Art hat.

1.1.6 Inputfile vom globalen Parser erzeugt

Es werden zwei Textdateien erstellt werden, eine für die Metadaten-Datensätze und eine für die Mutations-Datensätze. Diese werden danach per COPY-Befehl vom DWH eingelesen. Für diesen Befehl benötigen beide Dateien einen bestimmten Aufbau: Jedes Attribut ist voneinander per Leerzeichen getrennt und jede Zeile beherbergt einen Datensatz.

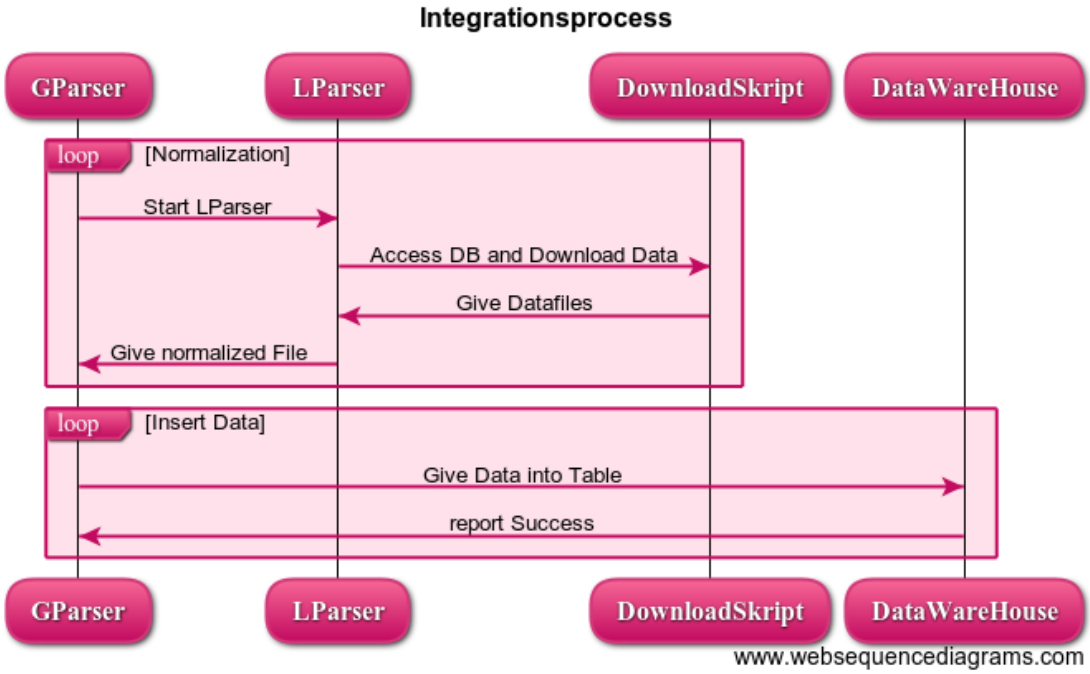


Abbildung 1: Sequenzdiagramm des Integrationsprozesses

1.1.7 Sequenzdiagramm

Der Ablauf besteht aus zwei Schleifen: Zuerst wird der gesamte Prozess von hinten gestartet, um zu gewährleisten, dass vor Beginn des Integrationsprozesses alle Teilprozesse laufen um Probleme während des Prozesses zu vermeiden. Der globale Parser wird gestartet, startet von sich aus die n lokalen Parser, welche dann die Daten ihrer n Quellen herunterladen. Jede Quelle hat einen, eigens für sich geschriebenen, lokalen Parser. Nach dem erfolgreichen Herunterladen parsen die n lokalen Parser ihre Daten in jeweils ein Inputfile. Somit haben wir n Inputfiles. Wenn diese Schleife beendet ist, wird der globale Parser alle n Inputfiles einlesen und aus diesen n Dateien zwei Textdateien erstellen - eine Metadaten-Datei und eine Mutations-Datei. Diese zwei Dateien werden dann vom DWH per COPY-Befehl eingelesen, die Metadaten zuerst, um Probleme mit der Relation zu verhindern, danach die Mutationen.

1.2 Datenbankentwurf

Die, bereits bei dem Attributmapping genannten, Attribute werden in zwei Tabellen gestaffelt, um eine effizientere und weniger fehleranfällige Struktur zu haben. Man kann sich möglicherweise sogar Einträge sparen, da es vorkommen kann, dass mehrere Mutationen von der gleichen Person stammen. Außerdem können die Anfragen der Middleware somit auch besser durchgeführt werden.
Erklärungen der Attribute der "MutationTabelle:

Name	Typ	Format	Wertebereich	Beispiel	Notwe
MutationID	integer	i	0 - 2.147.483.647	1, 2, 3, ...	no
Mutation	text	String	beliebig lang	'ATTCGATTAGCAGT'	no
Mutationsanfang	bigint	i	0 - 9.223.372.036.854.775.807	60000	no
Mutationsende	bigint	i	0 - 9.223.372.036.854.775.807	80000	no
Quelle	text	String	beliebig lang	'dbSNP'	no
Referenzname	text	String	beliebig lang	'GRCh38'	no
Chromosom	char(2)	cc	X, Y oder 0 - 99	'X', 'Y', '64'	nicht no
MetadatenID	int	i	0 - 2.147.483.647	1, 2, 3, ...	no

Erklärungen der Attribute der "MetadatenTabelle:

Name	Typ	Format	Wertebereich	Beispiel
MetadatenID	int	i	0 - 2.147.483.647	1, 2, 3, ...
Quelle	text	String	beliebig lang	'dbSNP', '1000GenomeProject'
Geschlecht	char	c	m oder w	'm', 'w'
Herkunft	text	Länderkürzel	Länderkürzel (meist dreistellig)	'GBR'
Downloadzeit	date	dd/mm/yyyy	Bis zum Jahr 5874897 n.Chr.	'24/12/2015'

Wir entschieden uns für diesen Aufbau, da die Abfragen des Users sich in zwei Bereichen unterteilt: Die Mutation selber und die dazugehörigen Metadaten. Eine Separierung dieser beiden Teilgruppen gibt der Middleware mehr Möglichkeiten schneller und effizienter das gewünschte Ergebnis zu finden. Dabei muss aber beachtet werden, dass es noch eine Relation zwischen den beiden Tabellen geben muss, damit jede Mutation genau einen Metadaten-Datensatz zugewiesen bekommt. Dies war der Grund, weshalb wir uns für das relationale Datenbankschema entschieden. Dieses Datenbankschema gibt uns die Möglichkeit, eine Relation zwischen den Mutationen und den Metadaten zu erstellen.
Nach dieser Entscheidung benötigten wir ein Datenbankmanagementsystem, was den Ansprüchen des Projektes gerecht werden musste, das heißt, dass es effizient mit großen Datenmengen umgehen können muss. Unsere Entscheidung fiel auf PostgreSQL. Dieses DBMS kann problemlos viele Daten und vor allem große Datensätze speichern und verwalten, was ausschlaggebend für das gesamte Projekt ist.

1.3 Entwurf des Parsers

Der lokale Parser wird auf die jeweilige Quelle zugeschnitten sein. Er wird die vorher heruntergeladenen Dateien entpacken, entschlüsseln und danach die relevanten Daten

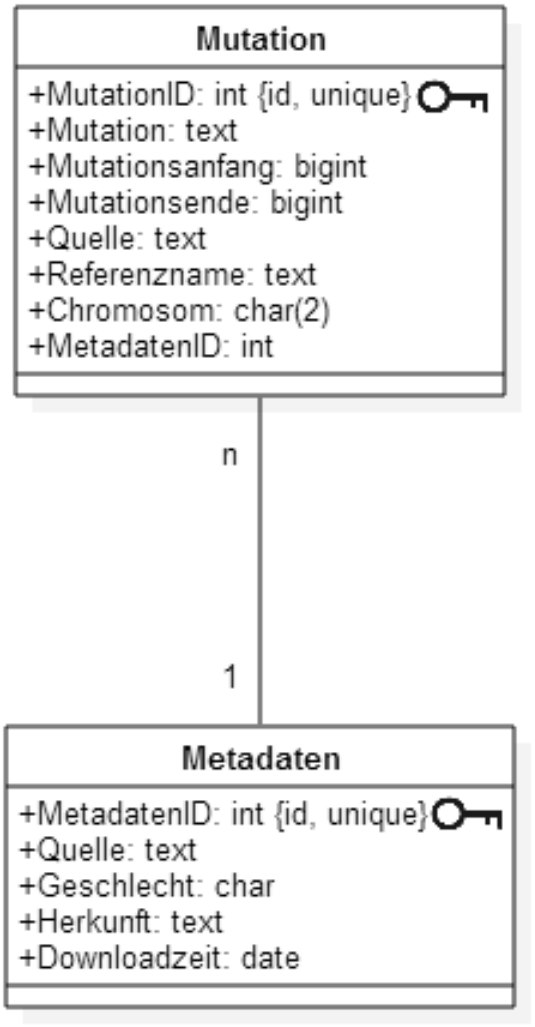


Abbildung 2: Entwurf der Datenbank

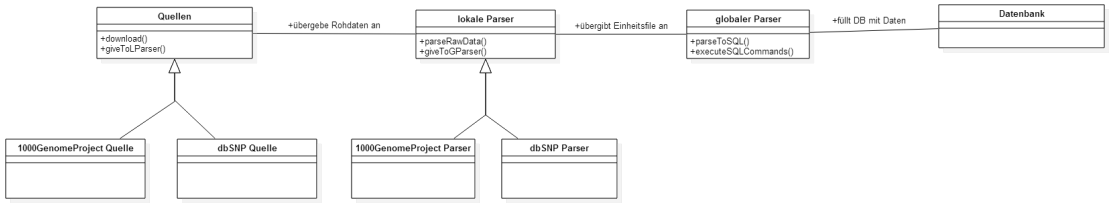


Abbildung 3: TODO

aus den Dateien herauslesen und in einem einheitlichen Format für den globalen Parser abspeichern.
Der globale Parser wird die Einheitsdateien der lokalen Parser nehmen, die beinhaltenden Daten in einzelne Datensätze aufteilen und diese dann in der Datenbank abspeichern und sie somit der Middleware bereitstellen.

1.4 Klassendigramm

1.5 Schnittstellenspezifikation

1.5.1 Schnittstelle: Integration - Middleware

Die Schnittstelle zwischen der Integration und der Middleware ist die, im Modell sichtbare, Datenbank. Sie ist der Ort, an dem die Integration die Daten bereitstellt und von wo die Middleware sich die Daten für die Anfragen abholt.

1.5.2 Schnittstelle: Integration - Benutzer

Die Schnittstelle zwischen der Integration und dem Benutzer ist das Hinzufügen neuer Quellen. Der Nutzer wird angehalten sein, zu wissen, wie seine neue Quelle aufgebaut ist, da er selber ein Downloadskript o.ä. dafür schreiben muss, sowie einen lokalen Parser. Diese werden an den entsprechenden Stellen im Programmcode eingefügt. Die lokalen Parser werden durch ein Interface vereinheitlicht.

1.6 Tests

1.6.1 Unit-Tests

Konkrete Tests konnten wir bisher nicht durchführen, jedoch gibt es einige Dinge zu testen: Es muss getestet werden, ob der lokale Parser arbeitet wie gewünscht, also mindestens 2 Testläufe für ihn: Bei einem, für ihn korrekten Inputfile, muss er ein entsprechend richtiges Outputfile für den globalen Parser erstellen. Sollte er ein Inputfile parsen, was nicht für ihn gedacht ist, soll er das Inputfile verwerfen oder eine Fehlermeldung ausgeben, aber auf alle Fälle das Outputfile nicht mit diesem Input erweitern. Natürlich kann das Inputfile auf verschiedene Weisen korrupt sein, was dort mehrere Testfälle notwendig macht.
Der globale Parser muss auf ähnliche Weisen getestet werden, jedoch kann man bei ihm

als Voraussetzung annehmen, dass die lokalen Parser korrekt arbeiten und auch ein korrektes Outputfile erstellt haben. Somit müsste nur überprüft werden, dass der globale Parser die Daten korrekt ausliest und korrekt in die Datenbank einfügt.

Korrektes Inputfile:

Referenzgenomname:GRCh38
Quelle:1000Genom
$$ SampelID:HG0094
Genkoordinaten:6:19:19
Mutationssequenz: AGTCTAGTA
$$ SampelID:HG0094
Gender:m
Population:GRB
Download:01:01:2001

Fehlerhaftes Inputfile

Referenzgenomname:KeinFehlerMöglich
Quelle:KeinFehlerMöglich
$
SapelID:KeinFehlerMöglich
Genkoordinaten:67:21:15
Mutationsequenz:ATCERROR
$$
SampelID:KeinFehlerMöglich?
Gender:h
Population:XXXX
Download:64:64:2045

2 Middleware

2.1 Indexstruktur

2.1.1 Anforderungen

Der Index soll eine effiziente Suche nach Mutationen für das Frontend ermöglichen. Während des Programmstarts wird der Index aus der vorhandenen Datenbank aufgebaut und steht dann so lange zur Verfügung, bis das Programm beendet wird. Im Index wird mit Intervallgrenzen gesucht und der Index gibt alle Intervalle zurück, die komplett innerhalb des angegebenen Intervalls liegen.
Der Index wird verteilt aufgebaut und liegt auf 4 virtuellen Maschinen verteilt. Die für den Index spezifizierten Funktionen sprechen immer einen Teilindex an. Der Aufruf der Funktionen wird über den IndexController erfolgen, der immer alle 4 Teilindizes ansprechen wird.

2.1.2 Funktionen und Datenstrukturen

Die Funktionen des Indexes variieren in ihrerem Ablauf je nach gewählter Indexstruktur. Momentan existieren 3 Varianten, die getestet werden. Es ist nicht ausgeschlossen, dass weitere Strukturen im Laufe der Entwicklung getestet werden. Da die Dauer des Indexaufbaus für den Endnutzer nicht relevant ist hängt die Auswahl der letztendlich genutzten Struktur lediglich von der Geschwindigkeit der Suchanfragen ab. Im folgenden werden die Such-und Einfüge-Operationen basierend auf den jeweiligen Indexstrukturen beschrieben

IntervallTree
In diesem Fall basiert die Indexstruktur auf einem Intervallbaum.
Hierfür wird die frei zugängliche Bibliothek İntervallST.java"der Universität Princeton genutzt. Beide Funktionen haben hier lediglich die Aufgabe als Interface zu den zugehörigen Bibliotheksfunktionen zu dienen: contains() zur Suche und put() zum Einfügen.
Das Suchergebnis wird nach möglichen Filtern gefiltert. Bei n Intervallen und einer Such-Ergebnisliste der Größe m ergibt sich ein Komplexität von O(log n + m)
Die Bibliothek muss noch angepasst werden, damit das Einfügen von gleichen Intervallen möglich ist ohne, dass das zuerst eingefügte Intervall gelöscht wird.

```
search()
found intervalls = contains(intervall);
forall the elements in found intervalls do
    if element corresponds to specified filters then
        add element to answer list;
    end
    return answer list;
end
```

```
addMutation()
if mutation is already in index then
    return ïndex already contains mutation";
end
put();
return „added mutation“;
```

Suchbaumbasierter Index
Da ein Großteil der Intervalle einstellig bzw sehr kurz sind bietet sich ein einfacher binärer Suchbaum als Datenstruktur an.
Dieser wird um einen Iterator erweitert, damit effizient eine Menge an Knoten ausgewählt werden kann.
Hierfür wird die Java-Klasse TreeMap verwendet, die eine Suche in logarithmischer Zeit ermöglicht. Die Ergebnismenge wird einmal durchlaufen, Mutationen deren Endpunkt außerhalb des gesuchten Intervalls liegen werden dabei entfernt und jeder Knoten wird nach den angegebenen Filtern gefiltert. Bei n Intervallen und einer Such-Ergebnismenge von m Intervallen ergibt sich eine Laufzeit von O(log n +m).
Auch in dieser Implementation dienen die Funktionen als Interface zu den jeweiligen Funktionen der genutzten Klasse: submap() zur Suche und put() zum Einfügen von Objekten).

```
search()
found intervalls = submap(intervall);
forall the elements in found intervalls do
    if element corresponds to specified filters then
        add element to answer list;
    end
    return answer list;
end
```

```
addMutation()
if mutation is already in index then
    return „index already contains mutation“;
end
put();
return „added mutation“";
```

Arraybasierter Index
Sollte die Anzahl der Mutationen groß genug sein, dass sie mit Integer-Variablen darstellbar ist, so bietet sich unter Umständen auch ein arraybasierter Index an.
Dieser speichert alle Mutationen aufsteigend sortiert nach ihrem Anfangspunkt. Wird nun nach einem Intervall gesucht, so iteriert er über das Array beginnend bei der Mutation, deren Startwert noch im gesuchten Intervall liegt. Dabei wird für jede Mutation überprüft, ob ihr Endwert noch im gesuchten Intervall liegt. Ist dem so wird sie zur Ergenismenge hinzugefügt. Hierbei können auch direkt die Filter überprüft werden.
Es so lange iteriert, bis der Startwert aller folgenden Mutationen größer, als der vom Nutzer angegebene Endwert ist.
Falls Mutationen an der gleichen Stelle beginnen, so verschieben sich alle folgenden Mutationen in der Liste, da in aufeinanderfolgenden Zellen gleiche Startintervalle gespeichert werden müssen. Es muss also ermittelt werden, wo sich die erste im Intervall liegende Mutation befindet. Ein Verfahren hierfür wird noch ermittelt
Bei m Mutationen, deren Startwert sich im gesuchten Intervall befinden, liegt die Laufzeit bei O(m + ϵ), wobei ϵ davon abhängt, wie die erste Mutation ermittelt wird. Es kann aber davon ausgegangen werden, dass ϵ einen geringen Anteil an der Laufzeit ausmachen wird.

```
search()
find index x of first mutation that lies in intervall;
while starting point of mutation at index x lies in search intervall do
    if endpoint of mutation at index x lies in search intervall and mutation
    corresponds to specified filters then
        add mutation to answer list;
    end
    x=x+1;
end
return answer list;
```

```
addMutation()
if mutation is already in index then
    return ïndex already contains mutation";
end
insert mutation at corresponding index and adjust array properly;
return „added mutation“;
```

2.2 IndexController

2.2.1 Anforderungen

Der IndexController nimmt Suchanfragen entgegen, leitet diese an die 4 Teilindizes weiter und fügt die Teilergebnisse wieder zusammen.
Falls eine Anfrage intern in mehrere Teilanfragen aufgeteilt werden sollte, weil z.B. ein Gen, nach dem gesucht wird an mehreren Stellen auftreten kann, leutet der IndexController alle Teilanfragen sequentiell an die Indizes weiter, fügt die Teilergebnisse zusammen und schickt die Ergebnismenge an den QueryReceiver zurück.

2.2.2 Funktionen

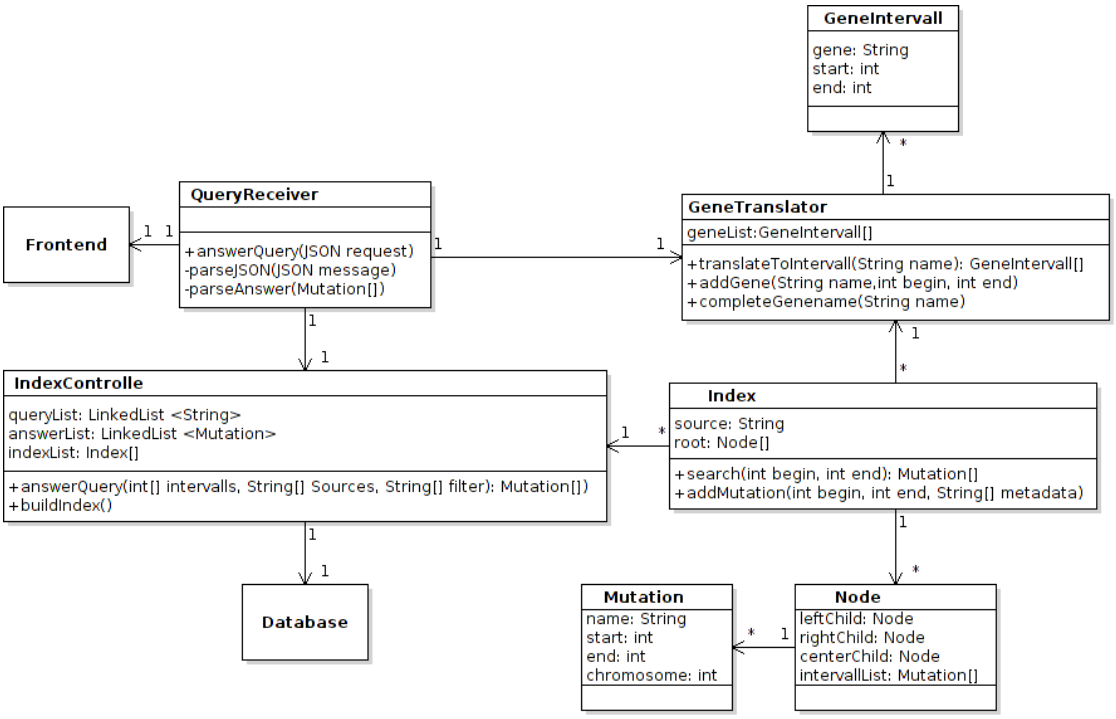
answerQuery(int[] intervals,String[] Sources,int[]filter) Die Funktion erhält mehrere Listen als Parameter, die die nötigen Informationen für die einzelnen Anfragen beinhalten. Jeweils 2 aufeinanderfolgende Einträge in der Intervall-Liste beschreiben den Start-und Endpunkt der gesuchten Intervalle. Die Einträge in den anderen Listen werden für alle Anfragen genutzt
Die Anfragen werden sequentiell an die 4 Teilindizes weitergeleitet und einzelnen Ergebnisse konkateniert und in einer Liste zurückgegeben.

```
forall the queries in parameter list do
    forall the sub indices do
        answer list = index.search();
    end
    concatenate all answer lists;
end
concatenate answer lists of each sub query;
return answer list;
```

buildIndex() Die Funktion wird bei Programmstart ausgeführt und baut auf Basis der Datenbank die 4 Teilinidzes auf.
Für jede Mutation wird zufällig entschieden in welchen Index sie eingefügt wird.

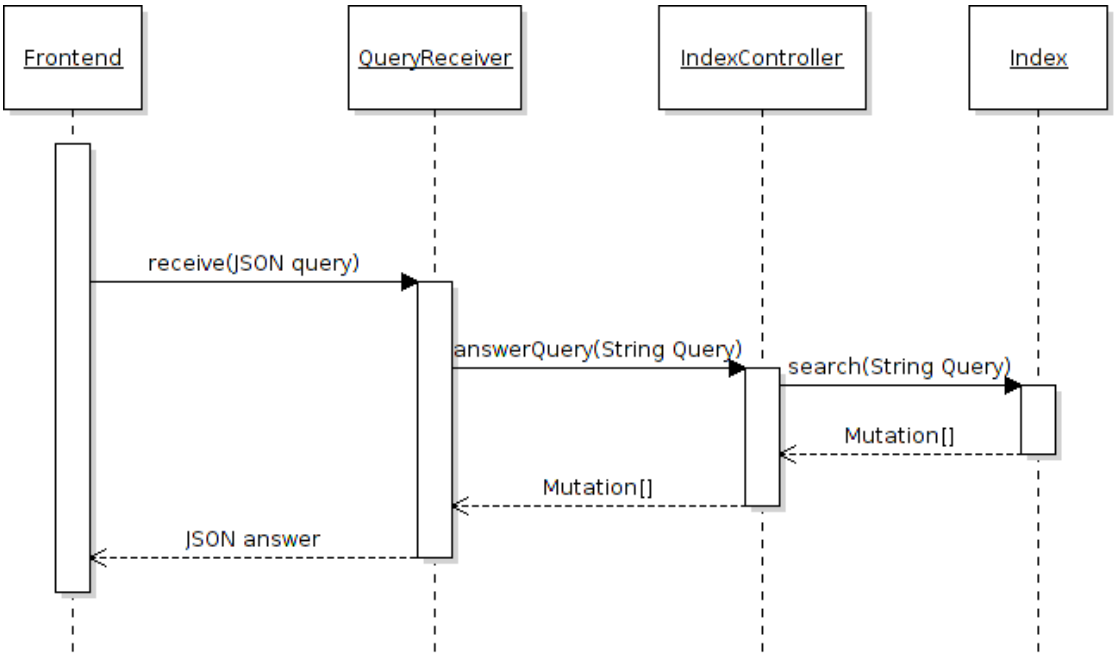
```
forall the elements in database do
    if element is already in index then
        skip element;
    end
    choose which sub index to insert into;
    index.addMutation();
end
return ïndex built";
```

2.3 Klassen-Diagramm

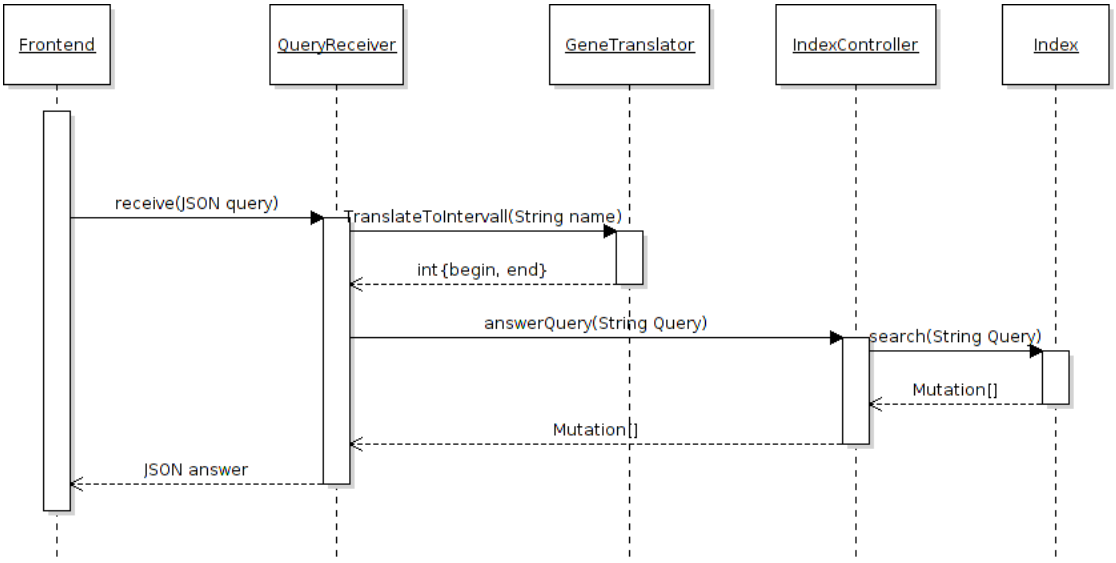


2.4 Sequenzdiagramme

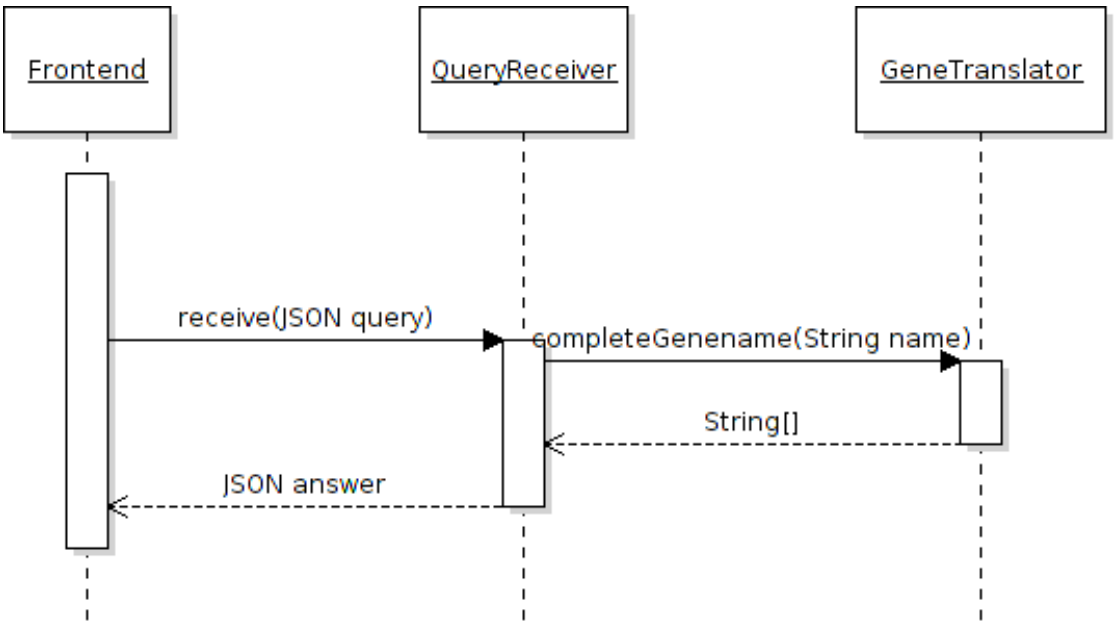
2.4.1 Intervall-Suche



2.4.2 Names-Suche



2.4.3 Suche nach Gennamen



2.5 Unit-Tests

2.5.1 QueryReceiver

Testfall 1 - erfolgreiche Intervallanfrage ohne Metadaten

```
  Eingabe
{
  "source": The Cancer Atlas,\usepackage[utf8]{inputenc}
  "chromosome": 3,
  "position": {"from": 100, "to": 200},
  "zoom": 1,
  "subindex": ,
  "hasDetail": (false)
}

  Ausgabe
{
  {
    "source:" The Cancer Atlas,
    "chromosome": 3,
    "position": {"from": 100, "to": 200},
    "details": { "refseq": ,
                "mutations": [{ "name": ,
                                "position": {"from": , "to": },
                                "metadata":
                              },]},
    "graph": {
              { "subintervall": Anzahl der Subintervalle,
                "counts": Anzahl der Ergebnisse
              }
            }
  }
}
```

Testfall 2 - erfolgreiche Intervallanfrage mit Metadaten

```
  Eingabe
{
  "source": The Cancer Atlas,
  "chromosome": 3,
  "position": {"from": 100, "to": 200},
  "zoom": 1,
  "subindex": ,
  "hasDetail": (true)
}
```

```
  Ausgabe
{
  {
    "source:" The Cancer Atlas,
    "chromosome": 3,
    "position": {"from": 100, "to": 200},
    "details": { "refseq": Referenzsequenz,
                 "mutations": [{Mutation1},{Mutation2},...]},
    "graph": {
              { "subintervall": ,
                "counts":
              }
            }
  }
}
```

Testfall 3 - erfolglose Intervallanfrage

```
  Eingabe
{
  "source": The Cancer Atlas,
  "chromosome": 3,
  "position": {"from": 100, "to": 200},
  "zoom": 1,
  "subindex": ,
  "hasDetail": (true)
}

  Ausgabe
{
  {
    "source:" The Cancer Atlas,
    "chromosome": 3,
    "position": {"from": 100, "to": 200},
    "details": { "refseq": ,
                 "mutations": ...},
    "graph": {
              { "subintervall": ,
                "counts":
              }
            }
  }
}
```

Testfall 4 - erfolgreiche Namensanfrage

Eingabe

```
{
  "source": The Cancer Atlas,
  "chromosome": 3,
  "search": "Gen im Cancer Atlas"
}
```

Ausgabe

```
{
  "source": The Cancer Atlas,
  "chromosome": 3,
  "search": "Gen im Cancer Atlas",
  "position": {"from": Startposition, "to": Endposition}
}
```

Testfall 5 - erfolglose Namensanfrage

Eingabe

```
{
  "source": The Cancer Atlas,
  "chromosome": 3,
  "search": "Gen, dass nicht im Cancer Atlas ist"
}
```

Ausgabe

```
{
  "source": The Cancer Atlas,
  "chromosome": 3,
  "search": "Gen im Cancer Atlas",
  "position": " "
}
```

Testfall 6 - fehlerhafte Anfrage

Eingabe

```
{
  "source": ""
}
```

Ausgabe

```
{
  "answer": "unknown format"
}
```

2.5.2 GeneTranslator

Der GeneTranslator hat 2 Aufgaben. Zum einen soll er während der Indexerstellung mit Inhalt (also Gennamen und zugeörigen Intervallen) befüllt werden, zum anderen soll eine Suche nach Gennamen in ihm möglich sein.

Testfall 1 - addGene - Einfügen in Datenstruktur
Eingabe: Testgen, 350, 500
Ausgabe: Dass Gen sollte in den Baum eingefügt sein und per searchForGene() findbar sein

Testfall 2 - addGene - Doppeltes Einfügen in Datenstruktur
Eingabe: Testgen, 350, 500
Testgen, 350, 500
Ausgabe: Dass Gen sollte nur einmal in die Datenstruktur eingefügt werden. Ausgabe, dass das Gen bereits in der Struktur vorhanden ist.

Testfall 3 - addGene - Aufruf ohne Parameter
Eingabe: [...]
Ausgabe: Fehlermeldung über parameterlosen Aufruf

Testfall 4 - tranlateToIntervall - erfolgreiche Suche
Eingabe: Testgen (befindet sich bereits in Datenstruktur)
Ausgabe: 350, 500

Testfall 5 - tranlateToIntervall - erfolglose Suche
Eingabe: Testgen2 (befindet sich nich in Datenstruktur)
Ausgabe: Fehlermeldung über erfolglose Suche

Testfall 6 - tranlateToIntervall - Aufruf ohne Parameter
Eingabe: [...]
Ausgabe: Fehlermeldung über Parameterlosen Aufruf

Testfall 7 - completeGeneName - erfolgreiche Suche mit einem Ergebnis
Eingabe: Test (Testgen1 befindet sich bereits in Datenstruktur)
Ausgabe: Testgen1

Testfall 8 - completeGeneName - erfolgreiche Suche mit mehreren Ergebnissen
Eingabe: Test (Testgen1 und Testgen2 befinden sich bereits in Datenstruktur)
Ausgabe: Testgen1, Testgen2

Testfall 9 - completeGeneName - erfolglose Suche
Eingabe: Test (Es befindet sich kein Gen mit Präfix Test in der Datenstruktur)
Ausgabe: Fehlermeldung über erfolglose Suche

2.5.3 IndexController

Testfall 1 - answerQuery - einfache Intervallanfrage
Eingabe: 1,3,100,200
Ausgabe: zum Intervall gehörende Mutationsobjekte

Testfall 2 - answerQuery - komplexere Intervallanfrage
Eingabe: 1,3,100,200 ; 1,3,150,350
Ausgabe: eine Mutationsliste mit den Ergebnissen beider Anfragen

Testfall 3 - answerQuery - unvollständige Intervallanfrage
Eingabe: 1,100,200
Ausgabe: Fehlermeldung über unvollständige Anfrage

Testfall 4 - answerQuery - leere Anfrage
Eingabe: [...]
Ausgabe: Fehlermeldung über leere Anfrage

Testfall 5 - answerQuery - überspezifizierte Intervallanfrage
Eingabe: 1,3,100,200,300,400
Ausgabe: Fehlermeldung über überspezifizierte Anfrage

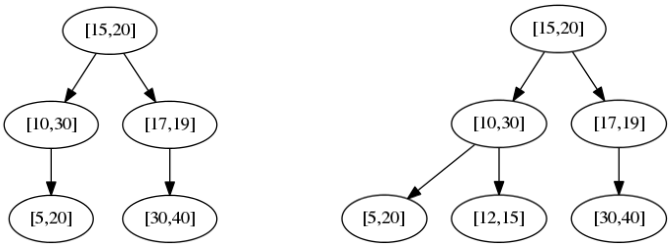
Testfall 6 - buildIndex - erfolgreicher Indexaufbau
Eingabe: [...] (Datenbank ist erreichbar)
Ausgabe: Erfolgreich gebauter Index

Testfall 7 - buildIndex - erfolgloser Indexaufbau
Eingabe: [...] (Datenbank ist nicht erreichbar)
Ausgabe: Fehlermeldung über erfolglosen Indexaufbau

2.5.4 Intervallbaum

1. Intervalle einfügen:
Das Intervall muss im Baum an der richtigen Stelle eingefügt werden und der Baum muss gegebenenfalls neu balanciert werden (z.B. wie ein AVL-Baum).

Bsp.: Einfügen des Intervalls [12,15]



2. Intervall mit Startpunkt > Endpunkt einfügen:
Wenn ein Intervall (S,E) mit S>E eingefügt wird, dann sollte unser Programm eine Fehlermeldung ausgeben und darauf hinweisen, dass die Grenzen für das Intervall nicht korrekt sind.

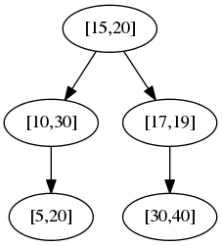
Bsp.: Einfügen des Intervalls [20,10] in einen beliebigen Baum.

3. Intervall mit Start- bzw. Endpunkt außerhalb des betrachteten Zahlenbereichs:
Wenn ein Intervall in dem Baum eingefügt werden soll, das teilweise oder vollständig außerhalb unseres Zahlenbereichs liegt (Länge des Genoms), dann muss es eine Fehlermeldung geben, die dem Nutzer mitteilt, dass der gültige Zahlenbereich überschritten wurde.

Bsp.: Einfügen des Intervalls [-5,7] in einen beliebigen Baum.

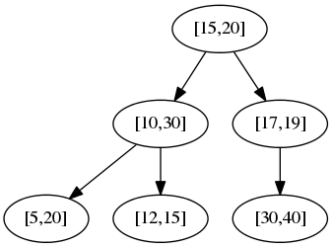
4. Schon vorhandenes Intervall einfügen:
Duplikate sollen von unserem Baum nicht gespeichert werden, d.h. es wird kein neuer Knoten hinzugefügt, sondern die Informationen (bei uns also Pointer auf Dateien) des neuen Knotens müssen im bereits vorhandenen Knoten mitgespeichert werden.

Bsp.: Einfügen des Intervalls [15,20] in den folgenden Baum



5. Suche nach vorhandenem Intervall:
Bei der Suche sollen alle Intervalle ausgegeben werden, die das gesuchte Intervall in irgendeinem Punkt überlappen.

Bsp.: Suche im folgenden Baum



Suche [4,5] ⇒ gib [5,20] aus
Suche [25,35] ⇒ gib [10,30] und [30,40] aus
Suche [20,20] ⇒ gib [15,20] und [5,20] aus

2.6 Stresstests

Der Stresstest hat zum Ziel herauszufinden, wie lange die durchschnittliche Query-Laufzeit ist.
Ziel des Systems ist es eine Laufzeit von unter einer Sekunde zu erreichen.
Um dies zu überprüfen werden mehrere Anfragen der gleichen Art (Intervallsuche, Gennamenssuche, Präfixsuche) sequentiell gestellt und die Antwortzeit gemessen. Alle Anfragen sollen in unter einer Sekunde eine Antwort erzielen.

3 Schnittstellenspezifikation: Middleware - Frontend

Bei der Kommunikation zwischen Middleware und Forntend ist ein Austausch von JSON über die HTTP-Methode GET vorgesehen. An die Middleware werden folgende Informationen geschickt:
Quellen; Chromosom; Position; Suche; Präfix-Flag; Zoomstufe; Anzahl der Subintervalle; HasDetail-Flag.
Von der Middleware werden bei bestimmten Flags, folgende Daten erwartet:
Wenn *IsPräfix-Flag* = `true`: Array aller Präfixe
Wenn *IsPräfix-Flag* = `false`: Name und Position der Suche

Wenn *HasDetail-Flag* = `true`: RefSeq + Mutationen mit MetaDaten
Wenn *HasDetail-Flag* = `false`: aggregierte Mutationen (durch Zoomstufe+Anzahl der *Subintervalle*)

3.0.1 Suchanfrage JSON

GUI ⇒ Middleware

```
{
  "source": ["src1", "src2", ...],
  "chromosome": int x,
  "search": String a
  "isPrefix": (true|false)
}
```

GUI ⇐ Middleware

```
{
  "source": ["src1", "src2", ...],
  "chromosome": int x,
  "search": String a,
  "prefix": ["String a", "String b", ...],
  "position": {"from": int x, "to": int y}
}
```

Bemerkung Wenn das *isPrefix* den Wert `'true'` besitzt, wird `"prefix"` befüllt mit einem Array aller möglichen Präfixe.
Wenn das *isPrefix* den Wert `'false'` besitzt, wird `"prefix"` nicht befüllt, aber `"search"` und `"position"` werden dann befüllt.

3.0.2 Intervallanfrage JSON

GUI ⇒ Middleware

```
{
  "source": String s,
  "chromosome": int x,
  "position": {"from": int x, "to": int y},
  "zoom": int y,
  "subindex": int z,
  "hasDetail": (true|false)
}
```

GUI ⇐ Middleware (pro Quelle)

```
{
  {
    "source:" String s,
    "chromosome": int x,
    "position": {"from": int x, "to": int y},
    "details": { "refseq": String b,
                 "mutations": [{ "name": String s,
                                 "position": {"from": int x, "to": int y},
                                 "metadata": {...}
                               },{...},...]},
    "graph": {
               { "subintervall": int x,
                 "counts": int y
               },
               {...}
             }
  }
}
```

Bemerkung Wenn das *HasDetail-Flag* `'true'` ist, dann wird `"detail"` befüllt und `"graph"` bleibt leer.
Wenn das *HasDetail-Flag* `'false'` ist, dann wird `"graph"` befüllt und `"detail"` bleibt leer.
Mit `"subindex"` ist die Anzahl der *Subintervalle* gemeint. Als *Subintervalle* werden kleine Bereiche des Intervalls bezeichnet. Diese werden dazu verwendet um einen Graphen detalierter darzustellen.

3.1 Beispiel der Kommunikation mit JSON

Suchanfrage Es wird eine Suchanfrage des Users getätigt. Die *Suche* wird mit dem Wert *"FOXP2"* befüllt.

GUI ⇒ Middleware

```
{
  "source": ,
  "chromosome": ,
  "search": "FOXP2"
  "isPrefix": false
}
```

GUI ⇐ Middleware

```
{
  "chromosome": 7,
  "search": "FOXP2",
  "prefix": [],
  "position": {"from": 114086310, "to": 114693772}
}
```

Darauf folgt eine Intervallanfrage mit der *Zoomstufe* von `'1,000,000bp'` und der Anzahl der Subintervalle von `'200'`.

GUI ⇒ Middleware

```
{
  "source": "Source 1",
  "chromosome": 7,
  "position": {"from": 114086310, "to": 114693772},
  "zoom": 1,000,000,
  "subindex": 200,
  "hasDetail": false
}
```

GUI ⇐ Middleware

```
{
  {
    "source:" "Source 1",
    "chromosome": 7,
    "position": {"from": 114086310, "to": 114693772},
    "details": {},
    "graph": {
              { "subintervall": 114091310,
                "counts": 233
              },
              {
                "subintervall": 114096310,
                "counts": 122
```

```
            },
            {
              "subintervall": 114101310,
              "counts": 89
            },
            {
              "subintervall": 114106310,
              "counts": 234
            },
            {...},
            ...
          }
  }
}
```

Intervallsuche Der User gibt ein *Intervall* an und möchte sich in der niedrigsten *Zoomstufe* die Details anschauen.

GUI ⇒ Middleware

```
{
  "source": "Source␣1",
  "chromosome": 14,
  "position": {"from": 5465, "to": 5665},
  "zoom": 200,
  "subindex": 0,
  "hasDetail": true
}
```

GUI ⇐ Middleware

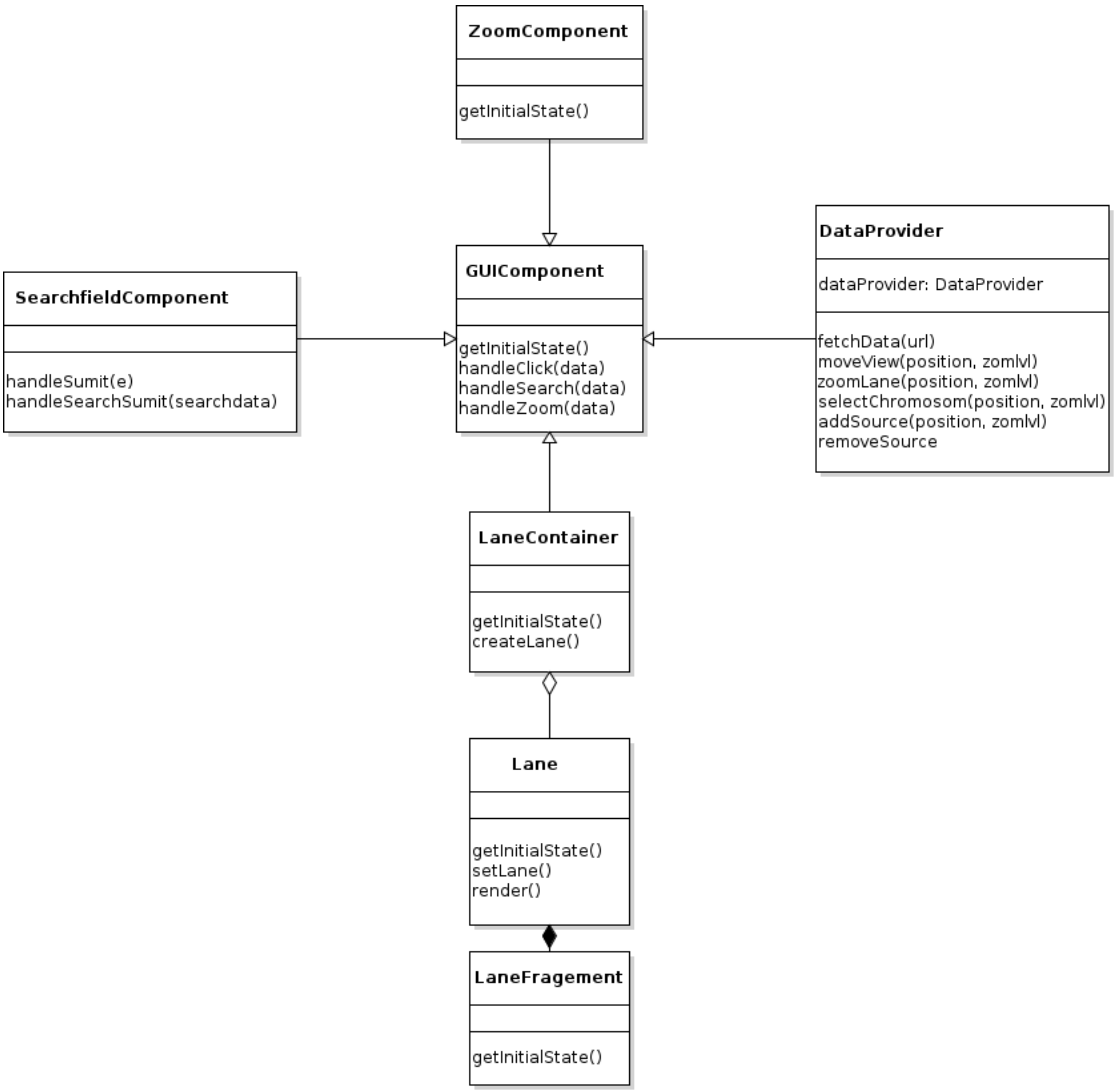
```
{
  {
    "source:" Source 1,
    "chromosome": 14,
    "position": {"from": 5465, "to": 5665},
    "details": { "refseq": "CCCTGAGAAAGACGCCCTGAGCGGGTAACGTCAACTGTCG
                            TGGGATTGGAAGAACTTGGTGGGGTTCCACATATCATGAT
                            CACGTAGAACACAATAAAGAAAATCTCCGGAGCGTGAATT
                            AAACTGAACTGACTCTTGTCAGTGTCTCGCTTACCACGTC
                            AAGGAAGTCAATGATGCCCTAATCTGGATAGCTTTGTCAT",
                 "mutations":
                 [{ "name": "RS12334",
                              "position": {"from": 5480, "to": 5498},
                              "metadata": {"sequence": "TCGGTATTCTACGTCTGC"}
                            },
                  { "name": "RBDDFG23",
                              "position": {"from": 5560, "to": 5567},
                              "metadata": {"sequence": "GGGCACT"}
                            }
```

```
                { "name": "RS12334",
                              "position": {"from": 5599, "to": 5610},
                              "metadata": {"sequence": "TGACGAGATCG"}
                            }
                { "name": "RS12334",
                              "position": {"from": 5650, "to": 5655},
                              "metadata": {"sequence": "CGACC"}
                            }]
            },
    "graph": {}
  }
}
```

4 Frontend

4.1 Mock-Ups der Benutzerschnittstelle

4.2 Klassen-Diagramm



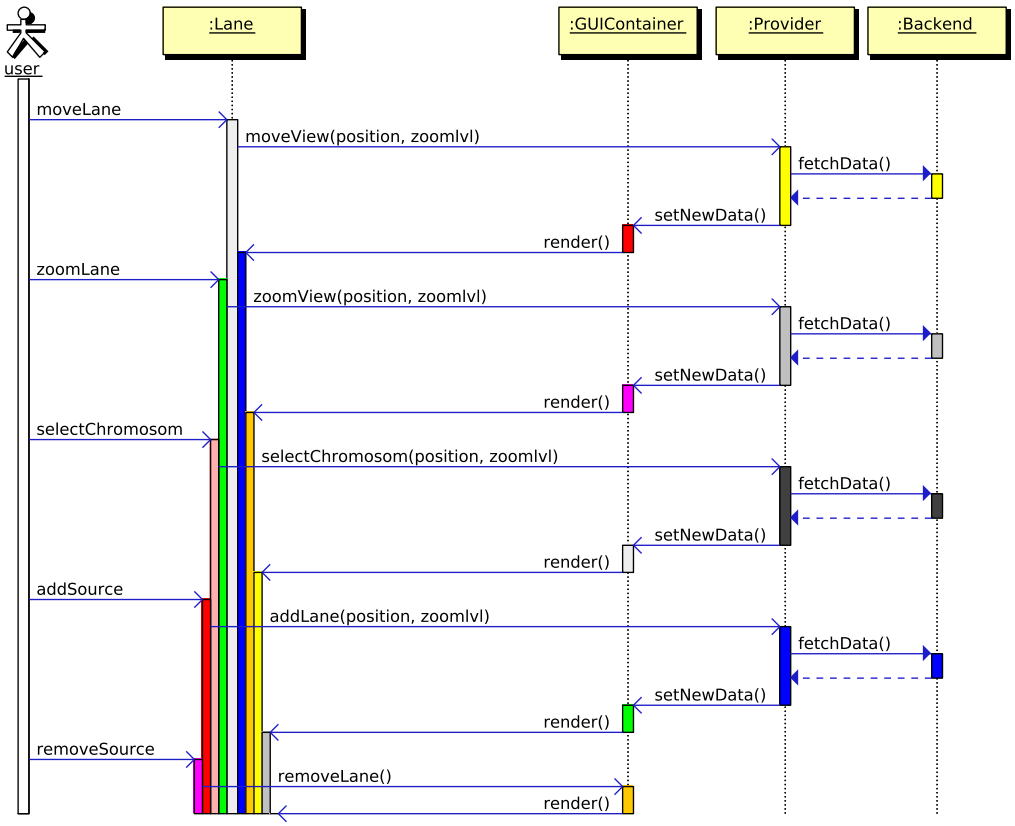
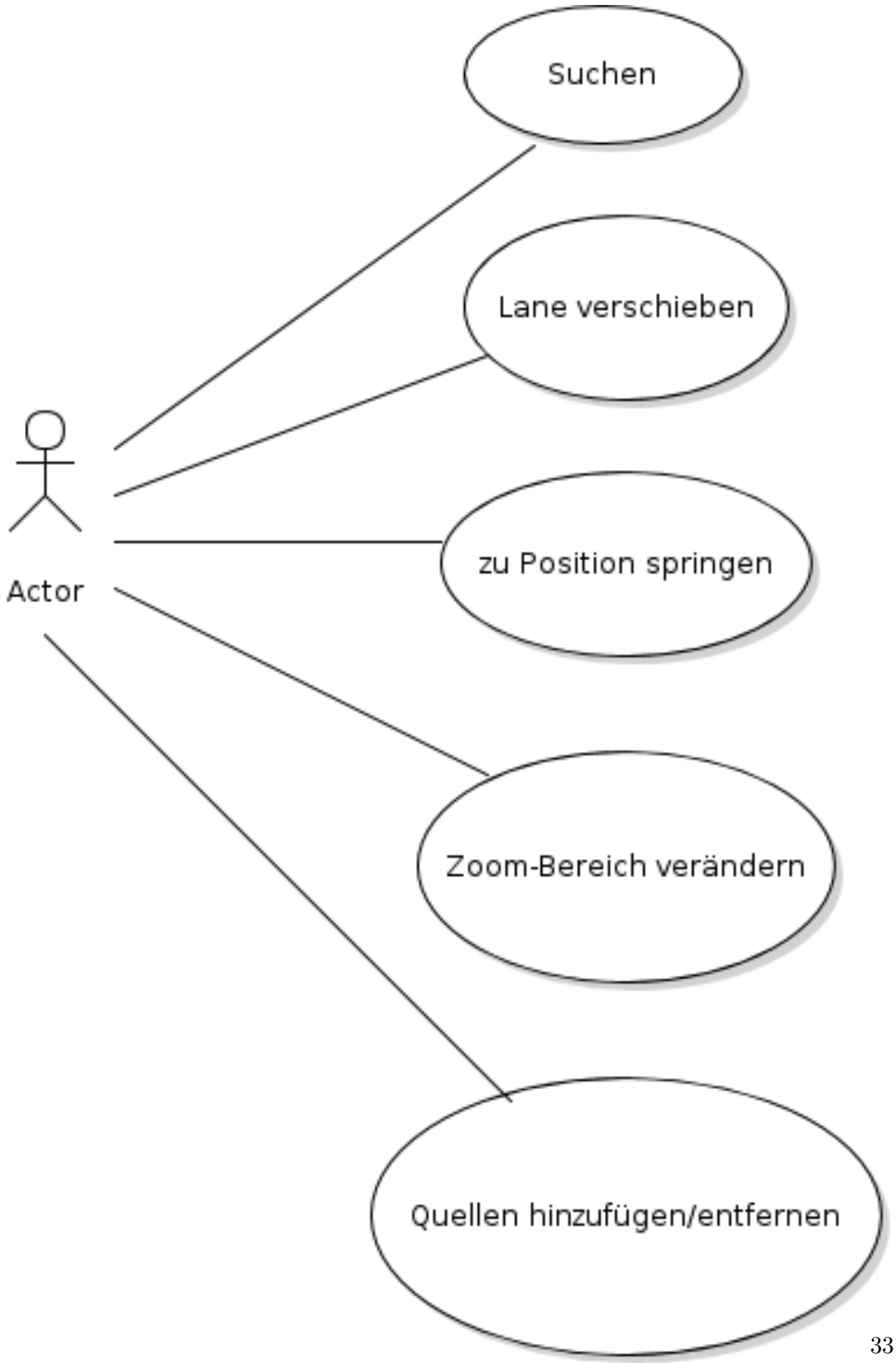


Abbildung 4: Ich bin eine Unterschrift

4.3 Sequenzdiagramm

4.4 Use Cases



4.5 Unit-Tests

4.5.1 Suchfunktion

1. Wenn ich als Nutzer eine leere Suche starte, dann möchte ich eine entsprechende Fehlermeldung angezeigt bekommen.
2. Wenn ich eine Suche mit falscher Eingabe starte, dann möchte ich eine entsprechende Fehlermeldung angezeigt bekommen.
3. Wenn ich als Nutzer nach einem gültigen Intervall suche, dann wird automatisch die Zoomstufe auf dieses Intervall angepasst.
4. Wenn ich als Nutzer nach einem vorhandenem Gene suche, dann wird automatisch die Zoomstufe auf dieses Intervall angepasst.

4.5.2 Quellen-Button

1. Wenn ich als Nutzer auf einen "QuellenButton drücke, dann wird mir die entsprechende Quelle zusätzlich zu den bereits dargestellten Quellen, angezeigt.
2. Wenn ich als Nutzer auf den "QuellenButton einer bereits angezeigten Quelle drücke, wird die entsprechende Quelle nicht mehr angezeigt.

4.5.3 Quellen-Scroller

1. Als Nutzer kann ich mich über horizontales Scrolling synchron durch die Quellen bewegen.

4.5.4 Zoom-slider

1. Wenn ich die Zoomstufe über den Slider ändere, dann werden die Quellen entsprechend der eingestellten Stufe dargestellt.
2. Wenn ich als Nutzer die feinste Zoomstufe einstelle, dann werden mir die Basenpaare angezeigt.
3. Wenn ich als Nutzer eine andere Zoomstufe einstelle, dann werden mir aggregierten Daten angezeigt.

4.5.5 Chromosom-Auswahl

1. Als Nutzer kann ich über ein Dropdown aus einer Vorauswahl von Chromosomen auswählen.
2. Wenn ich als Nutzer ein Chromosom auswähle, dann wird die Quellen-Anzeige automatisch entsprechend des ausgewählten Chromosoms aktualisiert.
3. Wenn ich als Nutzer das bereits ausgewählte Chromosom erneut auswählen, dann passiert nichts.

4.5.6 Allgemein

1. Wenn ich als Nutzer auf eine Anfrage warten muss, wird mir dies durch einen Loading-Spinner signalisiert.