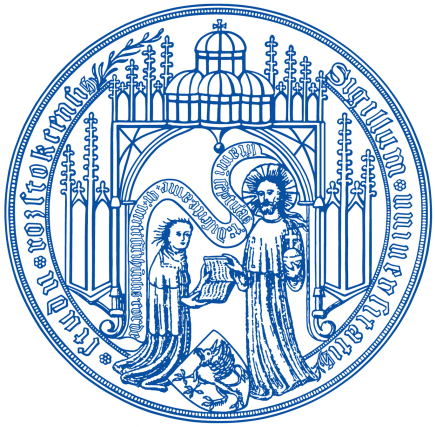
Constraint-getriebene Generierung von medizinischen Testdaten

Bachelorarbeit

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik



vorgelegt von: Najaf Abdiyev
Matrikelnummer: 219204774
geboren am: 22 März 2000 in Baku
Erstgutachter: Dr.-Ing. Hannes Grunert
Zweitegutachter: Dr. rer. nat. Marco Stubbe
Abgabedatum: 09.09.2025

Inhaltsverzeichnis

1 Einleitung

> „Das beste Rezept für eine moderne Gesundheitsversorgung ist eine Dosis Daten und eine Prise Systemdenken.“ [Jun25]

Diese Erklärung von Sven Jungmann hebt die Bedeutung hochwertiger medizinischer Daten und deren systematische Nutzung hervor. Um das Wachstum von Krankheiten zu beeinflussen und zu senken, Epidemien entgegenzuwirken und letztendlich das Leben der Patienten zu verbessern, führen Forscher verschiedene medizinische Studien durch. Eines der Beispiele ist die Arbeit pharmazeutischer Unternehmen. Mit Hilfe medizinischer Daten können sie die Wirksamkeit von Medikamenten untersuchen und auch neue entwickeln [VFA24]. Nicht zu übersehen ist das Forschungsprojekt onkoFDZ [Bun25c], das künstliche Intelligenz zur Analyse medizinischer Daten einsetzt. Onkologen können damit neue Einsichten über die Wirksamkeit der Behandlung erhalten und ihre Behandlungsempfehlungen für Patienten aktualisieren.

Aus diesem Grund werden in Deutschland aus den Krankenhäusern verschiedene medizinische Patientendaten gesammelt, um sie den Forschern für verschiedene Studien zur Verfügung zu stellen. Seit 2018 werden im Rahmen von Medizininformatikinitiative an allen Universitätskliniken Deutschlands die sogenannten Datenintegrationszentren eingerichtet, in die medizinische Daten aus verschiedenen Quellen gesammelt werden. Die gesammelten Daten sind im Kerndatensatz MII vereinheitlicht und basiert auf dem Standard HL7 FHIR, welcher für die nationale sowie internationale Verwendung der Daten notwendige Bedingung ist. Diese Daten werden in FHIR–Servern gespeichert, die einen zentralen Bestandteil der Datenintegrationszentren für die Bereitstellung von medizinische Daten bilden. Um Daten für die Forschung zu erhalten, muss ein Antrag über das Forschungsdatenportal für Gesundheit (FDPG) gestellt werden. Nach der Genehmigung des Antrags werden die benötigten Daten in pseudonymisierter Form aus den Datenbanken und FHIR–Servern extrahiert und den Forschern zur Verfügung gestellt [ATW+24]. Eine andere Methode zum Beschaffen von Daten ist die Extraktion mit Hilfe von Skripten. Skripte sind in der Regel in der Programmiersprache R verfasst und enthalten Code, der bestimmte Operationen oder Berechnungen an den Daten durchführt. Die Ergebnisse dieser Datenmanipulation stellen die eigentlichen Analyseergebnisse dar, die in anonymisierter Form an die Forschenden übermittelt werden [For25].

1.1 Problemstellung

Gesundheitsdaten von Patienten erfordern besondere Aufmerksamkeit und Schutz. Laut Datenschutz-Grundverordnung (DSGVO), soll die Verarbeitung dieser Daten unter Einhaltung spezifischer Regeln und Bedingungen erfolgen. Der Patient muss seine schriftliche Einwilligung zur Nutzung seiner medizinischen Daten geben. Ein Verstoß gegen diese Nutzungsbedingungen für die medizinischen Daten eines Patienten kann schwerwiegende rechtliche Konsequenzen haben [Eur16]. Aus diesem Grund muss vor der Verwendung dieser Skripte an echten medizinischen Patientendaten deren Funktionalität überprüft und sichergestellt werden, dass sie die Persönlichkeitsrechte nicht verletzen. Wegen solcher Einschränkungen steigt der Bedarf an künstlich generierten Daten, mit denen man diese Skripte testen kann.

1.2 Zielsetzung

Im Rahmen dieser Arbeit wurde das Ziel gesetzt, die Anforderungen zu formalisieren, die an die Testdaten gestellt werden, und FHIR–konforme Testdaten zu generieren. Unter Anforderungen versteht man syntaktische und semantische Anforderungen, die in der FHIR–Spezifikation und im Kontext des

MII–Kerndatensatzes vorab festgelegt sind. Das Arbeitskonzept wird als prototypische Anwendung präsentiert, in der die Generierung aufgerufen werden kann und die generierten, FHIR–konformen Testdaten sichtbar sind.

1.3 Forschungsfrage

Die Forschungsfrage, auf die in dieser Arbeit eine Antwort und Lösung gesucht werden soll:

- Wie können FHIR–konforme Testdaten generiert werden?
- Welche strukturellen und semantischen Anforderungen müssen erfüllt werden, damit die generierten Daten den FHIR–Profilen entsprechen, die im Rahmen des MII–Kerndatensatz definiert sind?

1.4 Aufbau der Arbeit

Das zweite Kapitel erörtert die theoretischen Grundlagen und erklärt wesentliche medizinische Terminologien wie ICD, SNOMED CT und LOINC sowie die Evolution der HL7–Standards bis hin zu FHIR. Das dritte Kapitel behandelt den Stand der Technik, in dem bestehende Generatoren und FHIR–Server analysiert und ihre Eignung im Kontext der Arbeit diskutiert wird. Im vierten Kapitel wird das Konzept der prototypischen Anwendung erläutert, einschließlich der Benutzeroberfläche, der Ressourcenvorlagen sowie der Interaktion mit dem FHIR-Server und dem Validator. Das fünfte Kapitel erläutert die Implementierung im Detail. Hierbei werden die verwendeten Bibliotheken und Werkzeuge, die Parametrisierung der Oberfläche, die Erstellung der Ressourcen sowie der Validierungsprozess erläutert. Das sechste Kapitel konsolidiert die Ergebnisse der Arbeit und bietet einen Ausblick auf potenzielle Erweiterungen.

2 Theoretische Grundlagen

In der zweiten Hälfte des 20. Jahrhunderts wurden im Gesundheitswesen immer mehr digitale Systeme eingesetzt. Krankenhäuser, Labore und Arztpraxen begannen damit, Patientendaten elektronisch zu speichern. Dabei trat ein großes Problem auf: Die IT–Systeme verschiedener Hersteller konnten nicht miteinander kommunizieren. Daten wurden in eigenen, sogenannten proprietären Formaten gespeichert. Das machte den Austausch von Informationen zwischen Einrichtungen schwierig oder sogar unmöglich [PJGBF$^+$23]. Da die Systeme nicht zusammenarbeiteten, mussten die Daten oft mehrmals eingegeben werden. Es kam zu Fehlern, Informationsverlusten und einem höheren Aufwand im Arbeitsalltag. Deshalb wurde deutlich, dass einheitliche Standards gebraucht wurden, damit verschiedene Systeme unabhängig vom Hersteller miteinander kommunizieren können [PJGBF$^+$23, BG21].

Um dieses Ziel zu erreichen, begannen viele Organisationen bereits damals damit, solche Standards zu entwickeln. Bis heute werden diese Standards kontinuierlich weiterentwickelt, verbessert und bringen dem Datenaustausch im Gesundheitswesen immer mehr Neuheiten und Möglichkeiten ein. In diesem Kapitel betrachten wir die Entwicklung dieser Standards und Terminologien, die schließlich in der heutigen Zeit in FHIR zusammengeführt und genutzt werden.

2.1 Terminologie

Medizinische Informationen müssen klar definiert und standardisiert sein, bevor sie strukturiert verarbeitet und zwischen verschiedenen Systemen interoperabel ausgetauscht werden können. Hier kommen medizinische Fachausdrücke und Klassifikationssysteme ins Spiel. Sie sind die Grundlage für eine konsistente Dokumentation und genaue Kommunikation im Gesundheitswesen. Im Weiteren werden drei der bedeutendsten international anerkannten Systeme vorgestellt, die wir in FHIR verwenden: ICD, SNOMED CT und LOINC.

2.1.1 ICD

Die ICD (internationale Klassifikation der Krankheiten) ist ein global anerkanntes System, das zur systematischen Erfassung, Dokumentation und Analyse von Krankheiten und Gesundheitsproblemen dient. Sie entstand erstmals bei der Notwendigkeit, die Nomenklatur zur Erfassung der Todesursachen bei Menschen zu standardisieren. Diese Idee wurde von internationalen Statistikern und medizinischen Fachleuten vorgeschlagen und führte 1893 zur Schaffung der ersten Version dieser Klassifikation unter der Leitung von Jacques Bertillon [Wor25a]. Seitdem wurde das ICD ständig unter der Koordinierung internationaler Konferenzen aktualisiert, an denen auch die Weltgesundheitsorganisation (WHO) teilnahm. Die weit verbreitete Version der ICD–10 wurde 1990 von der WHO übernommen und wurde allmählich in vielen Ländern eingeführt, einschließlich Deutschland. In Deutschland wurde diese Klassifizierung vom Bundesinstitut für Arzneimittel und Medizinprodukte (BfArM) speziell für den nationalen Gebrauch angepasst und heißt ICD–10–GM [Bun25b].

Der ICD–11 ist seit dem 1. Januar 2022 offiziell gültig. Es bietet eine digitale Struktur, eine verbesserte Unterstützung für diverse Krankheiten und zusätzliche Kodierungsoptionen. Die 2019 umgesetzte aktualisierte Version stellt einen wichtigen Fortschritt in der medizinischen Klassifikation dar. [Wor25b]. In Deutschland ist der gesamte Übergang zur ICD–11 derzeit noch nicht vollzogen. Es braucht eine gewisse Zeit, um diese Version vollständig in die bestehenden Systeme zu integrieren. Deshalb kommt in Deutschland zurzeit nur die ICD–10 zum Einsatz [AvdHW$^+$22]. Das ICD–System liefert weltweit vergleichbare Gesundheitsstatistiken, unterstützt die Abrechnung im Gesundheitssystem und dient als Grundlage für

die medizinische Forschung und die Entscheidungsfindung im Gesundheitswesen. Jedes ICD–Code stellt eine Krankheit und besteht aus einer Reihe von Buchstaben und Zahlen, die Ärzten und anderen Angehörigen der Gesundheitsberufe helfen, Krankheiten und Gesundheitsprobleme systematisch zu organisieren. Im Code steht zuerst ein Buchstabe, gefolgt von zwei Zahlen. Sie können auch einen Punkt und eine oder zwei weitere Zahlen hinzufügen. Diese Struktur ermöglicht es, Diagnosen in Ebenen der Allgemeinheit und Spezifität zu organisieren. Zum Beispiel steht der ICD–10–Code **E11.9** für *„Nicht näher bezeichnete Typ–2–Diabetes mellitus“* und lässt sich wie folgt interpretieren:

- **E:** Der erste Buchstabe zeigt, zu welchem Kapitel die Krankheit gehört. Hier zum Kapitel **E00–E90**, das sich mit *Hormon–, Ernährungs– und Stoffwechselkrankheiten* beschäftigt.
- **11:** Die beiden Ziffern danach geben die genaue Krankheitsgruppe an. In diesem Fall ist es *„Typ–2–Diabetes mellitus“*.
- **.9:** Die Zahl nach dem Punkt zeigt eine weitere Unterteilung. Hier bedeutet sie *„nicht näher bezeichnet“*, also dass keine genaueren Angaben oder Komplikationen vorliegen.

2.1.2 SNOMED CT

SNOMED CT ist eines der größten medizinischen Begriffssysteme weltweit. Die Abkürzung steht für „Systematized Nomenclature of Medicine – Clinical Terms“ und heute wird von der International Health Terminology Standards Development Organization (IHTSDO) gepflegt und weiterentwickelt. Beim Erscheinen von SNOMED CT wurde darauf abgezielt, eine gemeinsame Sprache für medizinische Terminologie zu schaffen, die weltweit verwendet werden kann. Heute enthält SNOMED CT über 311.000 medizinische Begriffe, die durch mehr als 1,36 Millionen Beziehungen miteinander verbunden sind und kann man mit seiner Hilfe viele Bereiche der medizinischen Versorgung wie Symptome, Diagnosen, Körperteile oder Medikamente viel genauer beschreiben [KF16].
Während ICD–10 eine monohierarchische Struktur aufweist, ist SNOMED CT polyhierarchisch aufgebaut. Dies bedeutet, dass ein Konzept mehreren übergeordneten Kategorien gleichzeitig angehören kann, was ein großer Vorteil für die realitätsnahe Abbildung komplexer medizinischer Sachverhalte ist. SNOMED CT verwendet Attribute zur Bildung von Beziehungen zwischen Konzepten und erlaubt damit auch tiefere semantische Strukturierung. Auf diese Weise wird die Dokumentation viel genauer zusammengestellt, die Computeranalyse und Entscheidungsunterstützung werden erheblich verbessert [Act12].

Zum Beispiel gibt es einen Patienten mit „Asthma“. In SNOMED CT ist „Asthma“ mit der eindeutigen Konzept–ID codiert. Gleichzeitig steht dieses Konzept auch in Beziehung zu anderen übergeordneten Kategorien, die ihrerseits auch eigenständige Konzepte mit einer eindeutigen Konzept–ID darstellen, wie:

- **Respiratorische Erkrankungen**
- **Chronische Erkrankungen**
- **Allergische Erkrankungen**

Zudem kann über Attribute wie `associated morphology` oder `finding site` weiter präzisiert werden z.B. „Entzündung der Bronchien“. Asthma kann dank dieser Polyhierarchie und attributbasierten Klassifikationen viel genauer in einem semantischen Kontext beschrieben werden als mit traditionellen Klassifikationen wie der ICD–10.
Seit 1. Januar 2021 ist Deutschland offiziell Mitglied von SNOMED International. Das Bundesinstitut für Arzneimittel und Medizinprodukte (BfArM) ist seitdem für die nationale Nutzung von SNOMED CT zuständig. Diese Maßnahme soll die semantische Interoperabilität im deutschen Gesundheitswesen verbessern und eine einheitliche Dokumentation in Forschung und Versorgung fördern. Die Medizininformatik-Initiative (MII) führte im Jahr 2020 die Pilotierung von SNOMED CT durch. Seitdem ist das System für alle Gesundheitseinrichtungen in Deutschland kostenlos, einschließlich Übersetzungen und lokalen Anpassungen [SNO21].

2.1.3 LOINC

Ärztliche Einrichtungen und Systeme zur Darstellung verschiedener Ergebnisse von Labor– und klinischer Untersuchungen verwendeten ihre eigenen Codes und Namen, was wiederum den Austausch und die Integration erschwerte. Aus diesem Grund begannen im Februar 1994 Fachleute vom LOINC–Komitee am Regenstrief-Institut mit der Entwicklung eines internationalen Standards namens LOINC (Logical Observation Identifiers Names and Codes). Bei der Erstellung von LOINC war das Hauptziel, dass verschiedene medizinische Einrichtungen dieselben Codes für Labor– und klinische Daten verwenden, um so den Austausch zu erleichtern und das Verständnis zwischen verschiedenen Systemen zu verbessern [Reg25, Aye18].

Das Institut Regenstrief hat auch eine Software namens Regenstrief Loinc Mapping Assistant (RELMA) entwickelt, die Werkzeuge enthält, mit denen Datenbanken angezeigt und lokale Begriffe mit LOINC–Begriffen abgeglichen werden können. Um LOINC in verschiedenen Ländern nutzen zu können, übersetzen Freiwillige die LOINC–Begriffe in ihre Muttersprache. In Deutschland übernimmt BfArM die Übersetzung und allgemeine Verarbeitung der LOINC–Termen. Momentan sind etwa 16.000 Termen ins Deutsche übersetzt [Bun25a].

2339-0 – *Glucose [Mass/volume] in Blood*

Dieser LOINC–Code beschreibt die Bestimmung der Glukosekonzentration im Blut und enthält und es ist nicht schwer zu bemerken, dass bei dieser kompakten Darstellung fehlen konzeptionelle Angaben und weitere Informationen über die Messung, die auch wichtig sind. Aus diesem Grund ist jeder LOINC–Eintrag intern in sechs strukturierte Komponenten untergliedert:

- **Component** – Was wird gemessen?
- **Property** – Welche Eigenschaft wird gemessen?
- **Time** – Zu welchem Zeitpunkt oder Zeitraum wurde gemessen?
- **System** – In welchem Körpermaterial oder Organ wurde gemessen?
- **Scale** – Welche Art von Ergebnis liegt vor?
- **Method** – Welche Messmethode wurde verwendet? (optionale Auswahl)

Der oben als Beispiel angegebene LOINC-Code 2339-0 wird vollständig zusammen mit den Komponenten wie folgt aussehen:

$$\mathit{Glucose} : \mathit{MCnc} : \mathit{Pt} : \mathit{Bld} : \mathit{Qn} : — \tag{2.1}$$

Aus einer solchen Form der detaillierten Beschreibung, in der alle Komponenten angegeben sind, ist der Name der gemessenen Substanz (*Glucose*) ersichtlich. Danach folgt die Abkürzung *MCnc* (Mass Concentration), die angibt, dass die Menge der gemessenen Substanz als Masse pro Volumen Flüssigkeit angegeben ist. *Pt* (Point in time) verdeutlicht, dass die Messung zu einem spezifischen Zeitpunkt und nicht über einen Zeitraum erfolgte. Das Blut (*Bld*) wurde zur Messung verwendet, das Ergebnis wird als quantitativer Wert angezeigt. Die letzte Angabe (—) zeigt an, dass für diesen Code keine festgelegte Messmethode existiert.

2.2 Alte HL7 Standards

HL7 (Health Level Seven) ist eine internationale Organisation, die daran arbeitet, die Zusammenarbeit zwischen Computersystemen im Gesundheitsbereich zu verbessern. Der Name „Level Seven“ bezieht sich

auf die siebte Ebene des ISO/OSI–Referenzmodells und zeigt, dass HL7 sich auf die Kommunikation auf Anwendungsebene konzentriert. HL7 wurde 1987 in den USA gegründet, um den steigenden Bedarf nach standardisiertem Austausch klinischer und administrativer Daten zu decken. Ziel der Organisation ist es, unabhängig von Systemarchitektur die Interoperabilität zwischen unterschiedlichen Gesundheitssystemen zu verbessern und eine gemeinsame Sprache für medizinische Daten zu schaffen [Wik25b]. In Deutschland gibt es HL7 ebenfalls schon seit den 1990er Jahren. HL7 Deutschland e.V. ist ein gemeinnütziger Verein, der sich für die Verbreitung dieser Standards einsetzt und mit Fachgruppen, Behörden und Unternehmen zusammenarbeitet. Der Verein ist Teil eines größeren Netzwerks und gehört zu einer übergeordneten Struktur, dem Spitzenverband IT–Standards im Gesundheitswesen (SITiG) [HL725].

2.2.1 HL7 v2

Im Jahr 1989 wurde die erste Version des HL7, nämlich HL7 v2 veröffentlicht und war erste Versuche, eine strukturierte und maschinenlesbare Kommunikation zwischen Softwarelösungen zu ermöglichen. Besonders schnell fand er seine Verbreitung in Krankenhäusern, weil es den Austausch klinischer und administrativer Informationen vereinfachte. Statt jeder Anbieter seine eigene Schnittstelle definieren musste, konnte jetzt jeder das gleiche Nachrichtformat verwenden. Dieses Nachrichtformat hat eine zeilenbasierte Struktur, bei der jede Zeile ein sogenanntes Segment darstellt. Das Zeichen "|" (Pipe) zeigt die Trennung verschiedener Felder innerhalb eines bestimmten Segments an. Innerhalb von Feldern können Unterfelder auch mit "^" (Caret) und Wiederholungen mit "~" getrennt werden [OAKA+25].

```
MSH|^~\&|GHH LAB|ELAB-3|GHH OE|BLDG4|200202150930||ORU^R01|CNTRL-3456|P|2.4<cr>
PID|||555-44-4444||EVERYWOMAN^EVE^E^^^^L|JONES|19620320|F|||153 FERNWOOD DR.^
^STATESVILLE^OH^35292||(206)3345232|(206)752-121||||AC555444444||67-A4335^OH^20030520<cr>
OBR|1|845439^GHH OE|1045813^GHH LAB|15545^GLUCOSE|||200202150730||||||||
555-55-5555^PRIMARY^PATRICIA P^^^^MD^^|||||||||F|||||444-44-4444^HIPPOCRATES^HOWARD H^^^^MD<cr>
OBX|1|SN|1554-5^GLUCOSE^POST 12H CFST:MCNC:PT:SER/PLAS:QN||^182|mg/dl|70_105|H|||F<cr>
```

Abbildung 2.1: Beispiel einer HL7 v2–Nachricht : `https://ringholm.com/docs/04300_en.htm`

In Abbildung 2.1 ist ein Beispiel für eine Nachricht im entsprechenden Format dargestellt. Sie besteht aus mehreren Segmenten und beginnt mit bestimmten Segmenttypen [Rin07]:

- **MSH**: Allgemeine Informationen wie Absender, Empfänger und Nachrichtentyp.
- **PID**: Stammdaten des Patienten (Name, Geburtsdatum, Adresse).
- **OBR**: Informationen zur angeforderten Untersuchung (Glukose).
- **OBX**: Das tatsächliche Ergebnis (Glukosewert von 182 mg/dl).

Obwohl seine technische Umsetzung vergleichsweise einfach ist, weist HL7 v2 einige grundlegende Schwächen auf, die sich negativ auf die Interoperabilität und die Verständlichkeit der Nachrichten auswirken [Rha25]:

1. **Fehlende einheitliche Anwendungsdatenmodellierung:** Die Datenstruktur ist im Standard nur implizit definiert. Ihre Darstellung und Speicherung hängen stark von der jeweiligen Anwendung ab. Dies führt dazu, dass klinische Informationssysteme HL7–Nachrichten unterschiedlich interpretieren und nur teilweise implementieren können.

2. **Fehlende formalisierte Methoden:** HL7 v2 bietet keine formale Modellierung von Datenfeldern und Nachrichtenstrukturen. Dadurch entstehen Inkonsistenzen im Standard, die das Verständnis der Beziehungen zwischen Nachrichtenelementen erschweren.

3. **Fehlende Benutzer- und Anwendungsdefinitionen:** Da der Standard keine klaren Rollenkonzepte vorgibt, entscheiden Softwareanbieter selbst, welche Teile des HL7–Nachrichtenschemas sie unterstützen. Das führt zu erheblichen Unterschieden in den Implementierungen, auch wenn zwei Systeme nominal HL7 v2 verwenden.

2.2.2 HL7 v3

Um strukturelle und methodische Schwächen von HL7 v2 zu beheben, wurde in 1990er Jahren die Entwicklung von HL7 v3 angefangen. Das Unterscheidungsmerkmal des Standards HL7 v3 von HL7 v2 ist, dass er eine strenge formale Methodik und ein objektorientiertes Datenmodell verwendet, das als Referenzinformationsmodell (RIM) bezeichnet wird. (siehe Abbildung 2.2) [HL7a]. Eine andere Neuheit ist, dass alle Informationen und Daten im XML-Format dargestellt werden, was die Menschenlesbarkeit erhöht [Bra22].

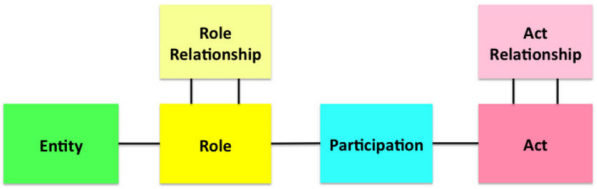


Abbildung 2.2: HL7 v3 Reference Information Model (RIM): `https://hl7.de/themen/hl7-v3-rim-das-referenzinformationsmodell/`

Dieses Modell enthält vier Basisklassen – **Entity, Role, Participation und Act** und zwei Verbindungsklassen: **RoleRelationship und ActRelationship**. Diese Klassen stellen reale medizinische Situationen dar. Beispielweise kann ein Patient als Entity auftreten, der in der Rolle „*Patient*“ **(Role)** an einer *Operation* **(Act)** teilnimmt, während ein *Chirurg* durch die Klasse **Participation** als „*Operateur*“ **(Role)** zugeordnet wird [HL7a].
Hl7 v3 Nachrichten haben eine Architektur, die aus drei Komponenten besteht: dem **Transmission Wrapper**, dem **Control Act Wrapper** und dem **Domain Content**, auch bekannt als **Payload**. Diese Komponenten werden benötigt, um den Nachrichtenfluss mehr verständlich und wiederverwendbar zu machen (siehe Abbildung 2.3) [DZI04].

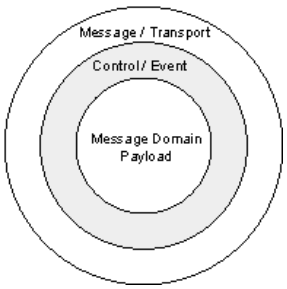


Abbildung 2.3: HL7 v3 Reference Information Model (RIM): `https://docs.oracle.com/health-sciences/health-hdr-81/HDRCS/wrappers.htm`

- **Transmission Wrapper:** Dieser Teil enthält technische Informationen wie Sender, Empfänger, Protokollversion und Sequenznummern, die für die korrekte Verpackung und Übermittlung der Nachricht erforderlich sind.

- **Control Act Wrapper:** Hier wird angegeben, welches Ereignis die Nachricht auslöst (z.B. eine Patientenaufnahme) und zusätzlich auch Information mit semantische Bedeutung z. B. „erstellen“ oder „aktualisieren“ zum verantwortlichen System oder Benutzer.
- **Domain Content oder Payload:** Dies ist der tatsächliche Inhalt der Nachricht, wie Patientendaten, Diagnosen oder Labordaten. Es wird je nach Verwendungszweck unterschiedlich hergestellt und erfüllt die spezifischen Anforderungen im klinischen Kontext.

Trotzdem brachte HL7 v3 ihre Neuerungen mit sich und löste einige Probleme der vorherigen Version von HL7 v2, gleichzeitig traten jedoch auch Schwierigkeiten auf. Die Schwierigkeit bestand im Anpassungsprozess dieser Version. Das bedeutete nicht, dass es unmöglich war, aber dafür waren Spezialisten für v3 erforderlich, die den Mechanismus des RIM genau verstanden und mit den entsprechenden Werkzeugen umgehen konnten. Leider gab es nur wenige solche Spezialisten, und der gesamte Prozess wäre kostenintensiv gewesen [Mer17].
Außerdem würde dieser Prozess viel Zeit in Anspruch nehmen, und den Anbietern müsste für eine bestimmte Periode gleichzeitig sowohl Unterstützung für HL7 v2 als auch für HL7 v3 angeboten werden [Hua19]. Noch ein interessanter Kritikpunkt wird vom Autor in folgendem Dokument erwähnt [Hol14], in dem er über die Nachrichtendichte spricht. Mit der Ankunft von HL7 v3 hat sich die Größe der XML-Elemente um ein Vielfaches erhöht, was auch die Komplexität dieser Version zeigt. Auch wenn diese Version nicht als erfolglos bezeichnet werden kann, ist die Bedeutung und Anwendbarkeit des vorherigen HL7 v2 bis heute noch weit voraus [Sol23].

2.2.3 HL7 CDA

Die Mehrheit der medizinischen Daten wurde in Freitext in einem unstrukturierten Format gespeichert. Da diese Texte aus verschiedenen IT–Systemen stammten, war die Verarbeitung und der Vergleich medizinischer Daten mit dieser Art der Speicherung erheblich herausfordernder. Jedes IT–System ist in Bezug auf seine technische Implementierung, Terminologie und Struktur unterschiedlich. Langfristig ist ein systematischer medizinischer Dokumentenaustausch ohne gut definierte semantische Standards unmöglich. Daher war ein einheitlicher Standard für klinische Dokumente erforderlich. Die CDA ist ein bedeutender Schritt in Richtung strukturierter, interoperabler klinischer Dokumentation, auch wenn es nicht alle diese Probleme vollständig adressiert [DAB+01].

Clinical Document Architecture (CDA) wurde im Jahr 2000 als offizieller ANSI-Standard veröffentlicht. CDA basiert auf dem HL7 v3 Reference Information Model (RIM) und heutzutage ist ein Bestandteil von HL7 Familie Version 3. In der Medizin war sie die erste Standard, die basierend auf Extensible Markup Language (XML) implementiert wurde [Hei03]. Sie wurde gezielt entwickelt, um den Menschen das Teilen von Dokumenten zu erleichtern, die jeder lesen kann, egal wie viel technische Erfahrung sie haben. Sie stellt auch sicher, dass alle verschlüsselten Informationen lange Zeit sicher aufbewahrt werden, sodass Gesundheitsdaten immer verfügbar sind [Med24].
Mit der Einführung der CDA wurde auch die Fähigkeit geschaffen, in ein Dokument nicht nur strukturierten Text, sondern auch andere Medien wie Bilder, Töne oder Videos zu enthalten. Dies vereinfacht die Darstellung des klinischen Zustands des Patienten beispielsweise durch Röntgenaufnahmen, Sprachaufnahmen oder Videoaufnahmen und verbessert somit die Kommunikation und die Qualität der medizinischen Versorgung [Hov10].
Es ist wichtig zu verstehen, dass der CDA als Standard die Struktur eines medizinischen Dokuments definiert. Es enthält keine spezifischen Regeln oder Anweisungen für die Übermittlung dieser Dokumente. Die Übertragung erfolgt auf verschiedene Arten, zum Beispiel [Bar25]:

- per E–Mail (*Multi–Purpose Internet Mail Extensions*)
- über das Internet (*Hypertext Transfer Protocol*)
- über das FTP (*File Transfer Protocol*)

Aber meistens wird in der Praxis die Übertragung innerhalb von HL7 v2 und HL7 v3 von Nachrichten durchgeführt. In der HL7–v2–Nachrichten wird sie als Anhang in ein OBX-Segment eingefügt und hat den Typ ED (Encapsulated Data), der diesen Anhang zulässt. In HL7 v3 wird sie als vollständige integrierte Datenstruktur in alle Nachrichten eingefügt, mit denen Dokumente ausgetauscht werden können [PP09].

Ein Dokument im CDA–Format gliedert sich grundsätzlich in zwei zentrale Bereiche: einen Kopfbereich (*Header*) und einen Inhaltsbereich (*Body*) (siehe Abbildung 2.4). Der CDA–Header speichert administrative und demografische Daten, die für die Identifikation und Kontextualisierung des Dokuments erforderlich sind. Typische Patienteninformationen wie (z.B. Name, Geburtsdatum), Informationen über den Verfasser des Dokuments, die behandelnde Einrichtung und das Datum und die Uhrzeit der Begegnungen sind in hier enthalten. Mithilfe dieser Informationen kann das klinische Dokument dem richtigen inhaltlichen Kontext zugeordnet werden, und es ist klar, von wem das Dokument verfasst wurde und für welchen Patienten es bestimmt ist. Der CDA–Body schließlich enthält den klinischen Inhalt des Dokuments. Dieser Inhalt kann strukturierter oder nicht-strukturierter Natur sein. Der strukturierte Inhalt besteht aus kodierten Elementen wie Diagnosen, Laborergebnisse oder Medikationsplänen, unstrukturierter Inhalt als formatierter Text, beispielsweise in Arztbriefen. CDA ermöglicht durch diese Aufteilung gleichermaßen eine maschinenlesbare Verwendung als auch eine menschenlesbare Darstellung für medizinisches Personal.

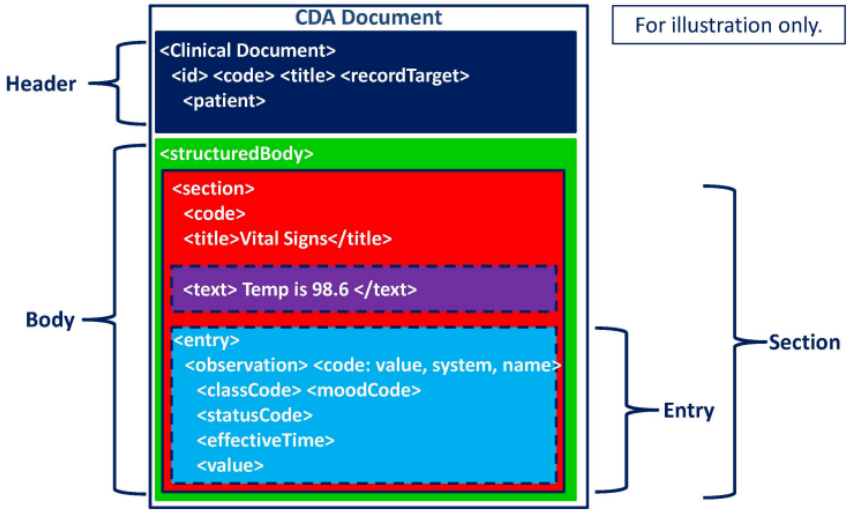


Abbildung 2.4: Struktur des CDA-Documents: `https://www.healthit.gov/sites/default/files/c-cda_and_meaningfulusecertification.pdf`

2.3 Moderner HL7 Standard – FHIR

Einer der nächsten von der Organisation HL7 erstellten Standards, das einen flexiblen Datenaustausch zwischen verschiedenen Softwaresystemen im Gesundheitswesen ermöglicht, ist FHIR. FHIR steht als Abkürzung für „Fast Healthcare Interoperability Resources“ und wurde im Jahr 2014 mit der ersten Version DSTU1 ins Leben gerufen und entwickelt sich seitdem ständig weiter, mit neuen Versionen. Jede neue Version erweitert bestehende Ressourcen, korrigiert Fehler und verfeinert technische Details. Obwohl die letzte Version R5 im März 2023 veröffentlicht wurde, bleibt ihre Vorgängerversion R4 in der aktuellen Zeit aufgrund ihres Status als „Normative“ noch häufiger im Einsatz. Dieser Status zeigt, dass diese Version stabil ist und keine weiteren Änderungen benötigt [Wik25a, Mil19].
Einer der entscheidenden Faktoren für die Entwicklung von FHIR ist der Paradigmenwechsel im Gesundheitswesen. Die Patienten möchten immer mehr Zugang zu ihren medizinischen Daten haben und diese kontrollieren, verwalten, an andere Ärzte oder Kliniken ins Ausland weitergeben. Darüber hinaus hat sich die technologische Umgebung weiterentwickelt, wobei internetbasierte Lösungen, Cloud–Dienste und mobile Anwendungen gegenüber stationären Systemen auf Personalcomputern bevorzugt werden. FHIR wurde geschaffen, um schnelle, einfache Verbindungen zwischen medizinischen Systemen und Patienten unabhängig von ihrem Standort zu ermöglichen und gleichzeitig moderne Technologien zu unterstützen und anpassungsfähig zu sein. Die Implementierbarkeit in FHIR ist ein anderer wichtiger Punkte, der sich im Vergleich zu den vorherigen Standards erheblich verbessert und erleichtert hat. FHIR umfasst viele Tools und Beispiele, die Programmierern helfen, diesen Standard schnell und leicht zu verstehen und zu studieren. Außerdem ist dieser Standard für alle frei verfügbar, ohne zusätzliche Einschränkungen, im Gegensatz zu seinen Vorgängern, da die alten HL7-Standards nur gegen Lizenzgebühr und vertragliche Vereinbarungen zugänglich waren, was deren Weiterentwicklung erschwerte. Durch diesen Vorteil gibt es bereits große Gemeinschaften von FHIR–Spezialisten, die ihr Wissen austauschen und Feedback erhalten können. Dies fördert die schnelle Weiterentwicklung und eine bessere Interoperabilität [HL7b].

2.3.1 Informationseinheiten in FHIR

Der Hauptbestandteil des Standards FHIR, durch den verschiedene medizinische Daten dargestellt werden können, sind Ressourcen. Jede Ressource besteht aus einer bestimmten Anzahl von Elementen, die in Kombination eine unikale Semantik für Ressourcen beschreiben. Ein charakteristisches Merkmal der Ressourcen ist ihre Atomarität, die es ermöglicht, verschiedene Operationen an diesen Ressourcen ohne irgendwelche externen Abhängigkeiten durchzuführen. Innerhalb einer Ressource gibt es Elemente, mit deren Hilfe auf andere Ressourcen verwiesen werden kann, um somit alle Szenarien im Gesundheitswesen unterschiedlicher Komplexität darzustellen. Diese Eigenschaft wird Modularität genannt, die die Skalierbarkeit im Gesundheitswesen gewährleistet. Derzeit gibt es ungefähr 150 verschiedene Ressourcen [Amo24] [Ray24]. Für bessere Klarheit werden die Ressourcen nach ihrem Typ in vier Klassen unterteilt [myS24]:

- **Administrative Ressourcen** (z.B. Patient, Practitioner)
  Diese Ressourcen sind fundamental und ohne sie ist es unmöglich, andere klinische und organisatorische Prozesse vorzustellen. Unterschiedliche demografische und verwaltungssachliche Informationen über die Personen, die bestimmte Gesundheitsdienste erhalten, sowie über diejenigen, die diese leisten, befinden sich in den Ressourcen dieser Klasse.

- **Klinische Ressourcen** (z.B. Observation, Medication)
  Ressourcen, die die klinische Verlauf des Patienten widerspiegeln, z.B. die Ergebnisse von Untersuchungen oder Analysen und Verfahren zu ihrer Behandlung, gehören zu dieser Klasse. Außerdem sind hier auch Ressourcen verbunden mit medizinischen Verschreibungen bezüglich der Einnahme von Medikamenten.

- **Infrastrukturelle Ressourcen** (z.B. Bundle, CapabilityStatement)
  Diese Ressourcen stellen keine klinischen Daten dar, sie regulieren die Kommunikation zwischen verschiedenen FHIR–Systemen. Sie dienen als technische Hilfsmittel beim Datenaustausch.

- **Finanzielle Ressourcen** (z.B. Coverage, PaymentNotice)
  In diesem Klasse sind Ressourcen gesammelt, die finanzielle Informationen verschiedener Art enthalten, wie Rechnungen für erbrachte medizinische Dienstleistungen und die Versicherungdeckung des Patienten.

2.3.2 Struktur der FHIR–Ressource

FHIR sorgt für die Speicherung von Ressourcen in den Formaten wie XML, JSON und RDF. Jede Ressource besteht aus vier Teilen: Die Identität und Metadaten, Narrative, Extensions (Erweiterungen) und strukturierte Daten [Nit17]. Als Beispiel wird die Observation–Ressource betrachtet, die in Abbildung 2.5 dargestellt ist. Die rot umrandete Teil ist Identität und Metadaten, die grüne ist Narrative die orangefarbene ist Extension und die blauen sind strukturierte Daten.

Identität und Metadaten

In diesem Teil werden solche Informationen angegeben wie ID der Ressource, die von einem bestimmten FHIR–Server ausgegeben wird. Es gibt einen Unterschied zwischen den Elementen *id* und *identifier*, der wichtig für das Verständnis ist: das *id*–Element wird vom FHIR-Server gesetzt und gibt den einzigartigen Wert dieser Ressource innerhalb des Servers an, im Gegensatz dazu wird das *identifier*–Element von einem externen System zugewiesen. Die Anzahl der gespeicherten Versionen dieser Ressource auf dem Server, die Zeit und das Datum der Aktualisierung sowie auch die kanonische URL, die obligatorisch ist, um die Ressource auf Übereinstimmung mit einem bestimmten Profil zu überprüfen, werden ebenfalls in diesem Teil angegeben [Was24].

Narritive

Dieser Teil ist eine Zusammenfassung für die in der „Ressource“ enthaltenen Informationen und zeigt sie in einer menschenlesbaren Form. Sie ist notwendig, um die in maschinenlesbarer Form präsentierten Informationen in eine verständliche und menschenlesbare Darstellung umzuwandeln. Eines der Elemente ist der *status*–Elemente, der zeigt, wie dieser Text erstellt wird, da er automatisch vom Server generiert oder von einer bestimmten Person geschrieben sein kann, einschließlich einer zusätzlichen Notiz. Der Text selbst befindet sich im *div*–Element [HL719].

Extensions

FHIR folgt der 80/20–Regel für Ressourcen. Elemente, die bereits in der Kernspezifikation vordefiniert sind, decken 80% der Informationen ab, die in Gesundheitssystemen ausgetauscht werden. Dies bedeutet, dass die meisten Implementierungen, die mit dieser Ressource arbeiten, alle diese Elemente verwenden werden, die 80% der Ressource ausmachen. Aus diesem Grund sollten der Aufbau dieser Ressourcen so weit wie möglich unverändert bleiben und einfach sein. Wenn zusätzliche Elemente benötigt werden, deren Verwendung in bestimmten Szenarien wichtig ist, werden diese als Extensions übertragen und sind die verbleibenden 20%, die die Ressource ergänzen [Yes23].
In Abbildung 2.5 ist ein Beispiel für die Verwendung der Extension „QuelleKlinischesBezugsdatum“ dargestellt. Nach Angabe des Datums und der Uhrzeit im *effectiveDateTime*–Element wird diese Extension ebenfalls hinzugefügt, um deutlich zu machen, worauf sich dieses Datum und diese Uhrzeit beziehen. Durch die Hinzufügung der Extension wird im Beispiel auf das Datum und die Uhrzeit der Probenentnahme (Specimen collection date) hingewiesen. Es gibt auch einen anderen Fall, der mit dieser Extension angegeben werden kann, zum Beispiel das Datum und die Uhrzeit, zu denen die Probe in das Labor

eingegangen ist (Date sample received in laboratory). Für beide Fälle wird ein bestimmter Code aus SNOMED CT sowie ein bestimmter zugewiesener Name zu diesen Codes verwendet. Diese Details sind bei Entscheidungsprozessen während der Behandlung sehr wichtig und die Extensions ermöglichen diese Präzisierung [Med25b].

Strukturierte Daten

Dieser Teil ist der größte und enthält verschiedene Elemente, die für jede Ressource vorbestimmt sind. Jedes Element hat seinen eigenen Datentyp, der ebenfalls in der Ressource angegeben wird. Für Observation Ressource wird durch diese Elemente verständlich, wer die Beobachtung durchführt, für welchen Patienten, was genau beobachtet wird und wie die Ergebnisse der Beobachtungen aussehen [Bii23].


```
{
  "resourceType": "Observation",
  "id": "76ce798d-d1db-4d15-a8f2-d82b134a0a76",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2025-07-02T19:04:10.903+00:00",
    "source": "#uwN6dFnNMqF57oLh",
    "profile": [ "https://www.medizininformatik-initiative.de/fhir/core/modul-labor/StructureDefinition/ObservationLab" ]
  },
  "text": {
    "status": "generated",
    "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\">Observation: O2-Sättigung = 99.4 %</div>"
  },
  "identifier": [ {
    "type": {
      "coding": [ {
        "system": "http://terminology.hl7.org/CodeSystem/v2-0203",
        "code": "OBI"
      } ]
    },
    "system": "https://exmaple.org/fhir/sid/test-lab-results",
    "value": "2713-6_7640677262",
    "assigner": {
      "identifier": {
        "system": "https://www.medizininformatik-initiative.de/fhir/core/CodeSystem/core-location-identifier",
        "value": "0e5c2d75-d337-4822-9944-56bf337b317e"
      }
    }
  } ],
  "status": "final",
  "category": [ {
    "coding": [ {
      "system": "http://loinc.org",
      "code": "26436-6",
      "display": "Laboratory studies (set)"
    }, {
      "system": "http://terminology.hl7.org/CodeSystem/observation-category",
      "code": "laboratory",
      "display": "Laboratory"
    } ]
  } ],
  "code": {
    "coding": [ {
      "system": "http://loinc.org",
      "code": "2713-6",
      "display": "Oxygen saturation Calculated from oxygen partial pressure in Blood"
    } ],
    "text": "O2-Sättigung"
  },
  "subject": {
    "reference": "Patient/f9c8fd17-1964-489b-9551-dbdbbd446c3d"
  },
  "encounter": {
    "reference": "Encounter/b8dd6544-fe43-4464-ba0b-c604a8caf24d"
  },
  "effectiveDateTime": "2025-07-02T19:04:10+00:00",
  "_effectiveDateTime": {
    "extension": [ {
      "url": "https://www.medizininformatik-initiative.de/fhir/core/modul-labor/StructureDefinition/QuelleKlinischesBezugsdatum",
      "valueCoding": {
        "system": "http://snomed.info/sct",
        "code": "399445004",
        "display": "Specimen collection date (observable entity)"
      }
    } ]
  },
  "issued": "2025-07-05T19:04:10+00:00",
  "performer": [ {
    "reference": "Organization/0e5c2d75-d337-4822-9944-56bf337b317e"
  } ],
  "valueQuantity": {
    "value": 99.4,
    "unit": "%",
    "system": "http://unitsofmeasure.org",
    "code": "%"
  },
  "referenceRange": [ {
    "low": {
      "value": 95,
      "unit": "%"
    },
    "high": {
      "value": 100,
      "unit": "%"
    },
    "type": {
      "coding": [ {
        "system": "http://terminology.hl7.org/CodeSystem/referencerange-meaning",
        "code": "normal",
        "display": "Normal Range"
      } ]
    }
  } ]
}
```


Abbildung 2.5: Exemplarische Observation–Ressource

2.3.3 FHIR–Profile und Implementierungsleitfäden

In internationalen Standards, insbesondere in FHIR, gibt es viele Freiräume. Jeder nutzt und interpretiert sie unterschiedlich. Der Grund dafür ist, dass Basisressourcen sehr allgemein gehalten sind und somit nicht immer für bestimmte Länder, Organisationen und folglich für ihre Anwendungsszenarien geeignet sind. Zum Beispiel gibt es in Deutschland einen Begriff wie Pflegestufe, der verschiedene Grade und Ebenen der Pflege eines Patienten klassifiziert [Joh21]. Dieser Begriff ist Teil des deutschen Bewertungssystems für die Pflege. Es ist in der FHIR–Spezifikation nicht enthalten, da in anderen Ländern ebenfalls ihre eigenen Klassifikationssysteme verwendet werden. Um bestimmte Ressourcen an spezifische Anforderungen und Regeln anzupassen, werden in FHIR Begriffe wie Profile und Implementierungsleitfäden verwendet.

Profilierung

$$Ressource + Extensions + Constraints = Profil \tag{2.2}$$

Diese Formel von Simone Heckmann [Hec22] beschreibt die Definition von Profilen. Das heißt, wenn einer bestimmten Ressource eine Reihe von *Constraints* (Einschränkung) angehängt wird, und die Ressource durch bestimmte Extensions ergänzt wird, entsteht daraus ein Profil dieser Ressource. Profile werden als *StructureDefinition*–Ressourcen dargestellt [Mic24]. In der StructureDefinition sind alle Anforderungen und Regeln für eine bestimmte Ressource festgelegt. Jeder StructureDefinition hat seine eigene kanonische URL. Profile können, ebenso wie Ressourcen, sich im Laufe der Zeit ändern und verschiedene Versionen haben. Die einzigartige kanonische URL hilft, Verwirrung zwischen den Versionen desselben Profils zu vermeiden.
Um den Unterschied zwischen einer Ressource und einem Profil besser zu verstehen, wird in Abbildung 2.6 die Patient–Ressource und ihr Profil demonstriert.

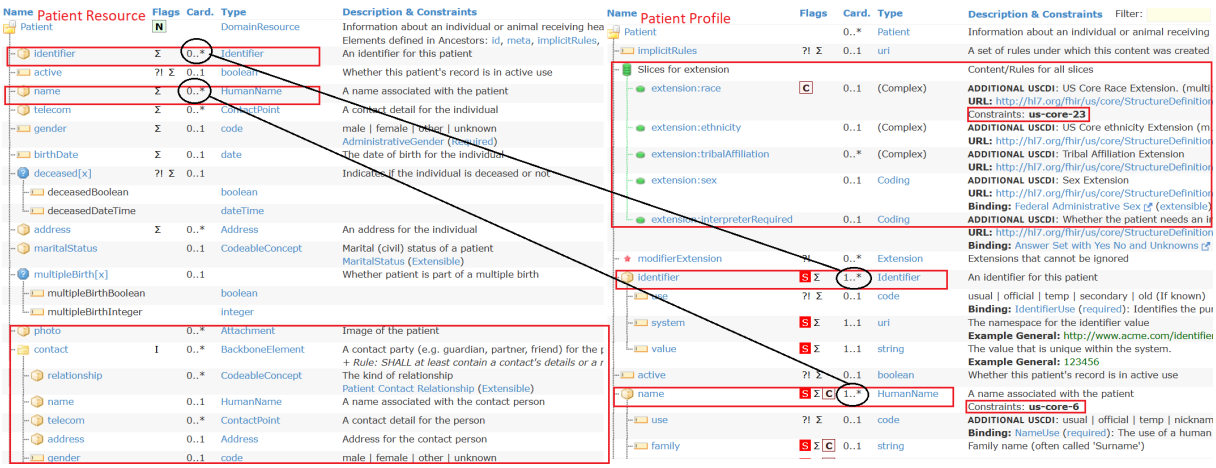


Abbildung 2.6: Unterschied zwischen Patient–Ressource und Patient–Profil

Dem Profil wurden zusätzliche Elemente hinzugefügt, die über verschiedene Extensions festgelegt werden. Eines davon ist die *race*–Extension, mit deren Hilfe die rassische Zugehörigkeit des Patienten angegeben wird. Darüber hinaus enthält dieses Element auch eine zusätzliche Einschränkung „*us-core-23*“. Der Patient kann verschiedenen Rassen angehören und sich mehreren Rassen zuordnen. Daher erlaubt diese Extension maximal sechs Werte (siehe Abbildung 2.7). Wenn eines der beiden Werte „ASKU“ oder „UKN“ angegeben wird, die auf Unbekanntheit der rassischen Zugehörigkeit des Patienten hinweisen, darf

kein anderer kodierter Wert vorhanden sein, der seine Rasse angibt. Das heißt, die Kombinationen wie {„UKN“, „asiatisch“} sind unmöglich und gelten als Fehler.
Außerdem beginnt im Profil die minimale Kardinalität der Elemente *identifier* und *name* mit eins, was bedeutet, dass die Angabe dieser Elemente bei Verwendung dieses Profils verpflichtend ist und mindestens einen Wert haben muss. Für das Element *name* ist die folgende Einschränkung „*us-core-6*“ definiert, die verlangt, dass bei Fehlen eines der Werte *name.given* oder *name.family* die Ursache des Fehlens mithilfe der *DataAbsentReason*–Extension erläutert werden muss. Einige Elemente, die in den Ressourcen verwendet werden, wie das Foto des Patienten oder Kontakte eines engen Angehörigen, sind nicht im Profil enthalten und werden in diesem nicht benutzt.

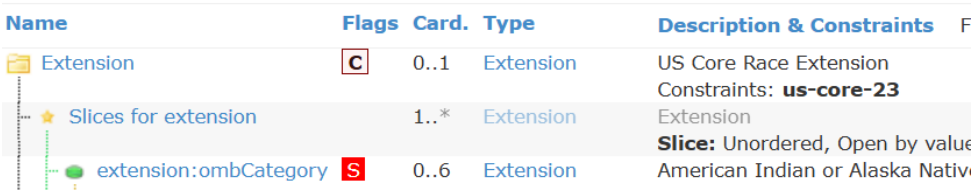


Abbildung 2.7: Maximale Anzahl von Werten: `https://build.fhir.org/ig/HL7/US-Core/StructureDefinition-us-core-race.html`

Implementierungsleitfaden

Profiles führen nur Einschränkungen auf einen bestimmten Ressourcentyp und ein Profil ist nicht ausreichend, um verschiedene Anwendungsfälle darzustellen. Um ein bestimmtes Szenario, zum Beispiel Patientenakte, darzustellen, werden verschiedene Profile benötigt, die für unterschiedliche medizinische Informationen verantwortlich sind. Implementierungsleitfäden fassen verschiedene Profile zusammen, die entwickelt wurden, um den lokalen Bedürfnissen gerecht zu werden. Sie fungieren als detaillierte technische Dokumentation, die die Anforderungen und die empfohlene Vorgehensweise für die Nutzung der Profile beschreibt. Alle Profile, Extensions, erlaubten Werte (Codesysteme und ValueSets), Beispielprofile und verschiedene Anweisungen zur Validierung dieser Ressourcen werden in einem einzigen IG–Paket zusammengeführt. Dieses Paket kann wie jede andere Ressource zwischen Systemen verschickt, heruntergeladen und lokal bei der Validierung verwendet werden. Für einen breiteren Zugriff werden verschiedene IG auf der Website Simplifier.net veröffentlicht [Hec21].

3 Stand der Technik

Testdaten finden zunehmend ihre Verwendung in verschiedenen Bereichen der Medizin, insbesondere in der Medizininformatik. Zur besseren Optimierung des Arbeitsprozesses arbeiten Fachleute in diesem Bereich an der Erstellung verschiedener Anwendungen für die Patientenversorgung und die Behandlungsplanung sowie Systeme, die den Austausch medizinischer Informationen ermöglichen. In den verschiedenen Phasen der Erstellung solcher Anwendungen und Systeme sowie bei der Testung verschiedener Algorithmen auf bestimmte Funktionalitäten benötigen Fachleute fiktive medizinische Daten [GGS23]. Aus diesem Grund steigt die Nachfrage nach ihrer Generierung. In einer solchen digitalen Umgebung müssen die Systeme nicht nur in der Lage sein, Daten auszutauschen, sondern diese auch richtig zu verarbeiten. Daher werden medizinische Daten unter Einhaltung des Standards FHIR gespeichert und in der im Rahmen von FHIR vorgegebenen Struktur und dem Format dargestellt [Khi21]. Dies ist ein wichtiger Faktor bei der Generierung von fiktiven medizinischen Daten, da diese auch im FHIR–Format generiert werden müssen, um später verwendet werden zu können. Diese Notwendigkeit hat die Entwicklung solcher Generatoren vorangetrieben.
In diesem Kapitel werden die derzeit verfügbaren Generatoren behandelt und beschrieben, warum sie unseren Anforderungen nicht entsprechen und warum die Notwendigkeit besteht, eine eigene Lösung zur Generierung zu erstellen. Darüber hinaus werden einige FHIR–Server beschrieben und miteinander verglichen.

3.1 Suchbegriffe und Suchmaschinen

Für die Recherche im Rahmen dieses Kapitels wurden verschiedene Suchbegriffe verwendet, darunter *FHIR Test Data Generation*, *BBMRI FHIR Generator*, *Synthea*, *FHIR Server Open Source*, *HAPI FHIR Installation Docker* und *Blaze FHIR Server*.

Die Suche wurde sowohl in allgemeinen Suchmaschinen wie `Google` als auch in wissenschaftlichen Datenbanken und Fachportalen wie `Google Scholar`, `IEEE Xplore`, `PubMed`, `SpringerLink` durchgeführt. Zusätzlich wurden Entwicklerressourcen wie `GitHub` und `Simplifier.net` herangezogen, um Implementierungsleitfäden und Quellcode–Referenzen zu berücksichtigen.

3.2 Generatoren

In diesem Abschnitt werden die derzeit vorhandenen Werkzeuge betrachtet, die allgemein FHIR–Ressourcen generieren. Es werden ihre Funktionsweise anhand von Beispielen der Generierung erläutert.

3.2.1 BBMRI FHIR Test Data Generator

Biobanking und Biomolecular Resources Research Infrastructure – European Research Infrastructure Consortium (BBMRI–ERIC) ist eine große Forschungsinfrastruktur, die im Jahr 2013 gemäß der Gesetzgebung der Europäischen Union gegründet wurde. Das Hauptziel ist es, Biobanken in den europäischen Ländern zu vereinen, um den Zugang zu verschiedenen biologischen Proben zu erleichtern, die für wissenschaftliche Forschungszwecke verwendet werden. Unter den teilnehmenden Ländern ist auch Deutschland, dessen offizieller Vertreter *German Biobank Network* (GBN) ist. Diese Organisation überwacht die Entwicklung der deutschen Biobanken, das heißt, dort werden verschiedene Proben gesammelt und gespeichert, sowie die zugehörigen Daten [KvJBH+20].

Zur Unterstützung dieses gesamten Prozesses wird der Standard FHIR verwendet und in Deutschland wird die BBMRI–Implementierungsleitfaden erstellt, um eine gemeinsame Suche und Nutzung von biomedizinischen Proben und Daten zu erreichen. In diesem Leitfaden gibt es eine kleine Anzahl von Profilen einiger Ressourcen, wie zum Beispiel Biobank, Collection, Patient, Spicemen und andere [Ger23]. Dieser Generator ist auf Grundlage des BBMRI–Implementierungsleitfaden erstellt. Es ist ein Open--Source CLI--Tool und wurde in der Programmiersprache Go entwickelt. Für jedes Profil gibt es ein Vorlage–Code, zum Beispiel ***specimen.go***, das eine Instanz des Profils Spicemen generiert. Die Werte, die bei den Elementen in dieser Ressource gewählt werden, stammen aus den Codesystemen und Valuesets, die im Rahmen dieses Implementierungsleitfadens definiert sind und bei der Generierung werden sie zufällig aus ihnen ausgewählt [KD23]. Um zu zeigen, wie die Auswahl der Werte für bestimmte Elemente im Profil erfolgt, wird im Listing 3.1 ein Teil des Codes als Beispiel aus der Datei *spicemen.go* gezeigt, der das spicemen.type-Element generiert. Dieses Element zeigt die Art des Biomaterials.

```go
func materialTypeCoding(r *rand.Rand) Object {
  return coding("https://fhir.bbmri.de/CodeSystem/SampleMaterialType",
    randMaterialType(r))
}

var materialTypes = []string{
  "tissue",
  "tissue-formalin",
  "tissue-frozen",
  "tissue-paxgene-or-else",
  "tissue-other",
  "liquid",
  "whole-blood",
  "blood-plasma",
  "blood-serum",
  "peripheral-blood-cells-vital",
  "buffy-coat",
  "bone-marrow",
  "csf-liquor",
  "ascites",
  "urine",
  "saliva",
  "stool-faeces",
  "liquid-other",
  "derivative",
  "dna",
  "cf-dna",
  "rna",
  "derivative-other",
}

func randMaterialType(r *rand.Rand) string {
  return materialTypes[r.Intn(len(materialTypes))]
}
```

Listing 3.1: Codeausschnitt aus der Datei *specimen.go*
Quelle: `https://github.com/samply/bbmri-fhir-gen/blob/master/gen/specimen.go`

- **`materialTypeCoding:`** Diese Funktion erstellt ein ***Coding***–Objekt, das aus den Unterelementen ***system*** und ***code*** bestehen soll. Der Wert für ***system*** ist bereits angegeben und verweist auf das spezielle Codesystem ***Sample Material Type***, das im Rahmen des BBMRI Implementierungsleitfadens definiert ist. Für den Wert des Unterelements ***code*** wird die Funktion **`randMaterialType`** aufgerufen.
- **`materialTypes:`** Es ist ein ***Slice***, in dem die Werte aus dem Codesystem ***Sample Material Type*** definiert sind.
- **`randMaterialType:`** Diese Funktion ruft einen Befehl auf, der zufällig Werte aus dem Slice **`materialTypes`** auswählt und dadurch das Unterelement ***code*** einfügt, um das ***spicemen.type***–Element korrekt darzustellen.

Generierung mit dem BBMRI Generator

Hier wird der Prozess der Generierung von Profilen nach einer vorgegebenen Anfrage betrachtet. Nachdem der Generator im angegebenen Ordner installiert wurde, ist es notwendig, in diesem Ordner im CLI den Befehl `bbmri-fhir-gen [directory] [flags]` auszuführen. An der Stelle von ***directory*** sollte der Ordner angegeben werden, in dem die generierten Daten gespeichert werden sollen. ***Flags*** dienen dazu, Parameter der Generierung anzugeben, zum Beispiel die Anzahl der zu generierenden Patienten. Wenn die Anzahl der Patienten nicht mit einer Zahl angegeben wird, generiert der Generator standardmäßig hundert Patienten [KD23]. In Listing 3.2 ist der Befehl angegeben, der die Generierung eines Patienten aufruft und die Ergebnisse im ***output***–Ordner speichert.

```
1 C:\Users\User\bbmri-fhir-gen> bbmri-fhir-gen output -n 1
```

Listing 3.2: Generierungsbefehl von BBMRI (Beispiel)

Am Ende werden zwei Dateien im JSON-Format generiert, nämlich ***transaction–0.json*** und ***biobank.json***. In Tabelle 3.1 ist die Anzahl der generierten Ressourcen innerhalb dieser Dateien angegeben. Die Datei ***transaction–0.json*** enthält in diesem Beispiel zahlreiche Diagnosen oder Zustände (Condition), Bioproben (Specimen), Ergebnisse klinischer Messungen (Observation), die mit einem bestimmten Patienten verbunden sind. Die Datei ***biobank.json*** stellt Informationen über verschiedene Biobanken, Zentren oder Einrichtungen dar, die an der Sammlung und Verarbeitung von Bioproben und deren Daten beteiligt sind.

Ressourcentyp	**transaction-0.json**	**biobank.json**
Patient	1	0
Condition	6	0
Specimen	14	0
Observation	4	0
Organization	0	11

Tabelle 3.1: Anzahl von Ressourcen

3.2.2 Synthea

Synthea, ein Open–Source–Simulator, der synthetische Patientendaten erzeugt, wurde von der MITRE Corporation entwickelt. Synthea hat sich zum Ziel gesetzt, realistische, aber künstlich generierte Gesundheitsdaten und elektronische Patientenakten bereitzustellen, die keine echten personenbezogenen Informationen enthalten. Die Entwickler benutzen diese synthetischen Daten, um verschiedene Gesundheitstechnologien zu testen und zu validieren. Außerdem finden sie ihre Anwendung auch in verschiedenen Forschungen und politischen Analysen [KSF+24]. Dieser Generator wurde in den USA entwickelt, weshalb er ausschließlich den US–Core–Implementierungsleitfaden sowie die Datengenerierung in allen FHIR–Versionen bis R4 unterstützt [Off22].
In Abbildung 3.1 wird das Arbeitsprinzip von Synthea gezeigt. Eines der Hauptelemente dieses Werkzeugs sind die Module (***Disease Modules***), die jeweils eine bestimmte Krankheit beschreiben und als Zustandsautomat im JSON–Format dargestellt werden. Zustandsautomaten für die entsprechenden Module werden aus ***Clinical Care Maps*** und ***Disease Incidence Prevalence Statistics*** vorbereitet. Clinical Care Map ist eine unterstützende Anleitung, die schrittweise die optimale Abfolge von Maßnahmen bei der medizinischen Versorgung eines Patienten mit einer bestimmten Krankheit beschreibt. Sie werden aus den klinischen Fachgesellschaften und begutachteten Forschungen gesammelt. Außerdem werden auch verschiedene öffentliche statistische Daten, die vom *Centers for Disease Control (CDC)* bereitgestellt werden, hinzugefügt.

Zu den Schlüsselfaktoren bei der Erstellung einer realistischen synthetischen Population von Patienten gehören demografische Daten (***Census Data Demographics***) und Konfigurationseinstellungen (***Configuration***). Daten aus Volkszählungen, Alters- und Geschlechtsverteilungen für jede Stadt, werden vom *US Census Bureau* bezogen. Diese Daten interagieren mit den Modulen, die von den Konfigurationsparametern gesteuert werden, und simulieren eine virtuelle Realität, in der synthetische Patientenakten erzeugt werden [WKN+18].

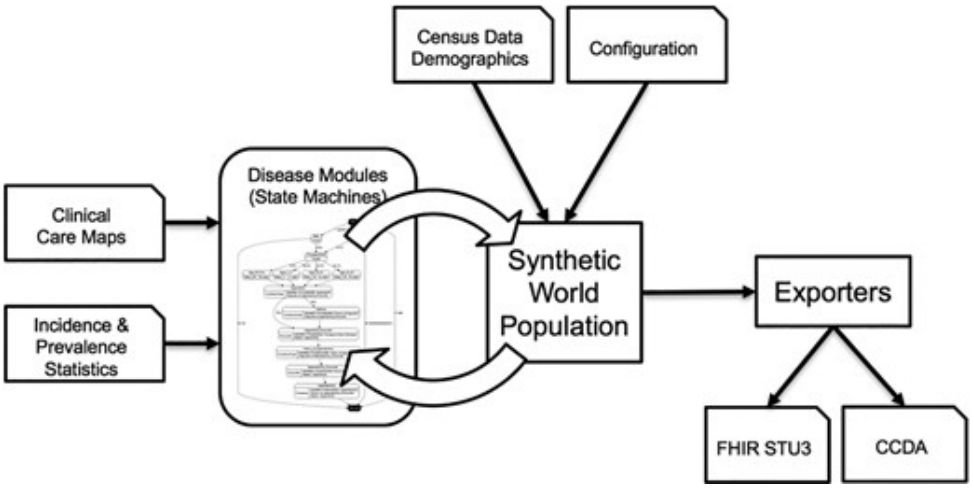


Abbildung 3.1: Architektur von Synthea `https://academic.oup.com/jamia/article/25/3/230/4098271`

Krankheitsmodule

Jedes Modul, das als Zustandsautomat dargestellt wird, besteht aus zwei Komponenten, nämlich aus Zustand (**State**) und Übergang (**Transition**). Zustand ist eine bestimmte Position oder ein bestimmter Abschnitt, in dem sich der Patient bei der Simulation einer Krankheit befindet. Der Zustand beschreibt, was im Moment mit dem Patienten passiert, zum Beispiel den Beginn der Krankheit, die Einnahme von Medikamenten, das Auftreten von Symptomen oder den Abschluss der Behandlung. In Synthea werden Zustände in Kontrollzustände (**control states**) und klinische Zustände (**clinical states**) unterteilt. Kontrollzustände, wie *Initial*, *Terminal*, *Delay*, *Guard*, steuern die Logik der Übergänge innerhalb des Moduls. Solche Zustände wie *Encounter*, *ConditionOnset*, *MedicationOrder* oder *Procedure* zu klinischen Zuständen gehören und verantwortlich sind, klinische Ereignisse und Aktionen zu modellieren.
Ein Übergang ist eine Regel, die den Wechsel von einem Zustand in einen anderen bestimmt. Sie werden auch nach ihren Kriterien für die Auswahl des nächsten Zustands unterschieden [WKN+18]:

- **Direkte Übergänge**, die keine Erfüllung einer bestimmten Bedingung erfordern.
- **Verteilte Übergänge**, die den nächsten Zustand auf Basis einer Wahrscheinlichkeit auswählen und dabei zufällig den einen oder anderen Weg bestimmen.
- **Bedingte Übergänge**, die nur dann durchgeführt werden, wenn eine spezifische Bedingung erfüllt ist. Am häufigsten werden sie im Guard-Zustand verwendet.
- **Komplexe Übergänge**, die alle Arten von Übergängen miteinander kombinieren.

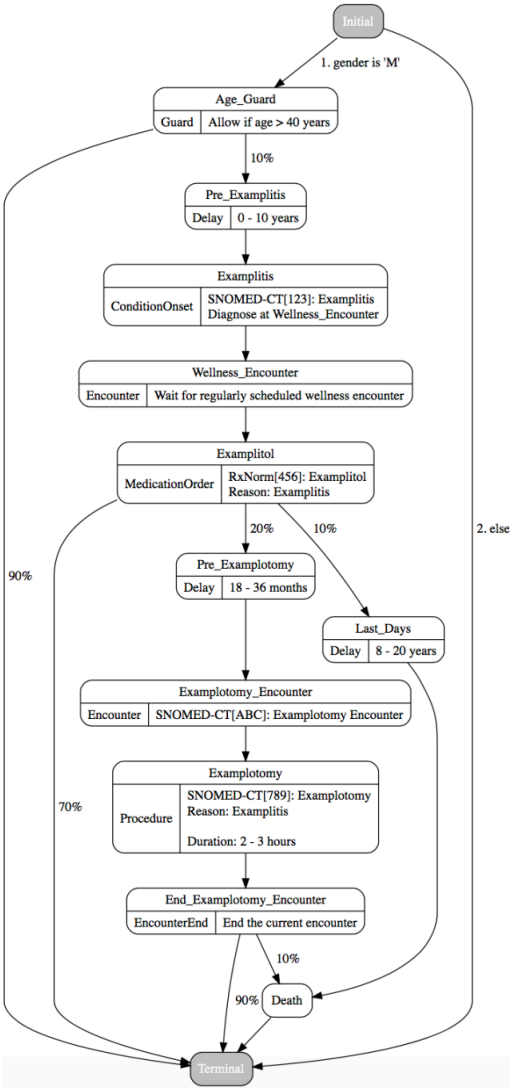


Abbildung 3.2: Exemplarischer Zustandsautomat
`https://synthetichealth.github.io/module-builder/#example_module`

In Abbildung 3.2 ist ein beispielhaftes Modul mit dem Namen Examplitis dargestellt, um die Funktionalität dieser Zustände und Übergänge in Synthea zu zeigen. Zuerst wird auf Basis der demografischen Daten in der synthetischen Realität ein Patient erzeugt. Zur Demonstration des Szenarios wird angenommen, dass unser Patient männlich ist und 47 Jahre alt ist. Nach der Erstellung des Patienten wird der **Initial**–Zustand aktiviert und abhängig vom Geschlecht des Patienten wird ein Übergang gewählt: Ist der Patient weiblich, so wechselt sie automatisch in den Zustand **Terminal** und die Simulation dieses Moduls für diesen Patienten wird beendet. Das bedeutet, dass nach der Generierung in ihrer Patientenakte diese Krankheit und alles, was damit zusammenhängt, nicht vorhanden sein wird. Da unser Patient männlich ist, trifft dieser Übergang nicht zu und wir wechseln in den Zustand **Age_Guard**. Dieser Zustand ist ein Kontrollzustand und hier wird in erster Linie das Alter des Patienten überprüft. Ist das Alter des Patienten unter 40 Jahren, wird die Simulation automatisch durch einen in Synthea definierten internen Mechanismus gestoppt, ohne in den Zustand **Terminal** zu wechseln. Unser Patient besteht die Filterung, da er älter als 40 Jahre ist.

Nach dieser Kontrollfilterung gibt es zwei Übergänge: Einer davon führt mit einer Wahrscheinlichkeit von 90% (0,9) in den Zustand **Terminal**, der andere mit einer Wahrscheinlichkeit von 10% (0,1) in den Zustand **Pre_Examplitis**. Um zu bestimmen, zu welchem Zustand unser Patient als nächstes übergeht, wird im Bereich von 0.0 bis 1.0 eine Zufallszahl „p“ generiert. Liegt die generierte Zahl im Intervall $0.1 < p \leq 0.9$ wird der Übergang gewählt, der zum **Terminal** führt. Liegt die Zahl dagegen $p < 0.1$, zum Beispiel 0.06, erfolgt der Übergang in den nächsten Kontrollzustand **Pre_Examplitis**. In der virtuellen Realität vergeht eine bestimmte Zeit, nämlich 10 Jahre, die für den Übergang in den nächsten Zustand erforderlich ist. Danach, wenn unser Patient bereits 57 Jahre alt ist, wird bei ihm die Diagnose **Examplitis** gestellt. Nach der Diagnosestellung wird der Patient zu einem geplanten Arztbesuch (**Wellness_Encounter**) geschickt. Im Rahmen dieses Besuchs wird ihm das Medikament Examplitol (**MedicationOrder**) verschrieben. Danach gibt es wieder drei Übergänge: In 70% der Fälle endet die Simulation, in 10% der Fälle wechselt der Patient in den Zustand **Last_Days** und stirbt nach einer bestimmten Zeit, und in 20% der Fälle setzt der Patient seinen Weg in den Zustand **Pre_Examplotomy** fort, der einen aufgeschobenen chirurgischen Eingriff modelliert. Hier wird eine Verzögerung von 18 bis 36 Monaten realisiert. Im Zustand **Examplotomy_Encounter** wird ein stationärer Aufenthalt modelliert, gefolgt von dem eigentlichen Eingriff (**Procedure**) mit einer Dauer von zwei bis drei Stunden. Nach Abschluss des chirurgischen Abschnitts erfolgt der Abschluss des Besuchs durch den Zustand **EncounterEnd**, und in 10% der Fälle stirbt der Patient (**Death**) oder in 90% der Fälle überlebt er und gelangt in den Endzustand **Terminal** [WC21].

Auf diese Weise ermöglicht Synthea eine vielfältige Generierung von Patientenakten, die letztlich im FHIR–Format als ***Bundle***–Ressource dargestellt werden, in der zahlreiche elementare FHIR–Ressourcen für den generierten Patienten enthält.

Aufruf der Generierung

Dieses Werkzeug ermöglicht es, Generierungsparameter wie die gewünschte Anzahl der generierten Patienten, deren Geschlecht, Altersbereich und Stadt oder Bundesstaat, in dem die Patienten wohnen, festzulegen. Die Generierung sollte aus dem Ordner synthea erfolgen, mit einem Aufruf, der wie folgt aussieht [HWS+23]:

- **Für Linux/macOS:**

```
./run_synthea [number_of_people] [gender] [min_age] [max_age] [region
[city]]
```

- **Für Windows:**

```
run_synthea.bat [number_of_people] [gender] [min_age] [max_age]
[region [city]]
```

Im Listing 3.3 ist ein Beispielbefehl für die Generierung von 1 Patientin weiblichen Geschlechts, die 28 Jahre alt ist und aus der Stadt Boston im Bundesstaat Massachusetts stammt. Die Befehlsparameter -p, -g und -a dienen dazu, die Generierung entsprechend den individuellen Anforderungen anzupassen. Bei „-a“ wird der Bereich des Alters spezifiziert, daher wird das gewünschte Alter des Patienten als Mindest– und Höchstwert angegeben. Nach Ausführung des Befehls wird eine Datei erzeugt, die eine enorme Anzahl verschiedener FHIR–Ressourcen innerhalb einer einzigen Bundle-Datei enthält (siehe Tabelle 3.2). Alle sind mit einem Patienten verbunden.

```
C:\Users\User\synthea>run_synthea.bat -p 1 -g F -a 28-28 Massachusetts
Boston
```

Listing 3.3: Generierungsbefehl von Synthea (Beispiel)

Ressourcentyp	Anzahl
Procedure	102
Observation	97
DiagnosticReport	47
Claim	39
ExplanationOfBenefit	39
DocumentReference	32
Encounter	32
Condition	20
SupplyDelivery	14
MedicationRequest	7
Immunization	7
CareTeam	3
CarePlan	3
Device	3
ImagingStudy	2
MedicationAdministration	1
Patient	1
Medication	1
Provenance	1

Tabelle 3.2: Anzahl der FHIR-Ressourcen im generierten Bundle.

3.2.3 Vergleich und Abgrenzung

Trotz der Tatsache, dass diese Werkzeuge FHIR–konforme Testdaten generieren, wurden sie jeweils entsprechend einer bestimmten Implementierungsleitfaden entwickelt, um in einem bestimmten Land, einer Organisation und sogar einem Projekt verwendet zu werden. Außerdem ist es ihnen auch nicht möglich, auf die Unterstützung anderer Implementierungsleitfäden eingestellt zu werden, sodass das generierte Profile dem gewählten Implementierungsleitfaden entspricht. Im Gegensatz zu BBMRI erlaubt Synthea, mehr Parameter für die Generierung festzulegen, wodurch man einem vorab generierten Patienten bestimmte demografische Daten zuweisen kann, und erstellt auch realistischere Szenarien, da es auf fertigen Modulen basiert. Aufgrund der fehlenden Unterstützung eines anderen Implementierungsleitfadens, nämlich des MII-Implementierungsleitfadens, sind diese Generatoren für die Verwendung im Rahmen dieser Arbeit nicht geeignet. Ein weiterer Grund ist, dass diese Generatoren nicht sehr flexibel hinsichtlich der Parameter sind. Zum Beispiel ist es nicht möglich, vor der Generierung, die Verwendung einer bestimmten Extension anzugeben oder einen bestimmten Wert für ein bestimmtes Element im Profil festzulegen, um zum Beispiel den Blutdruck oder die Temperatur.
Aus diesen Gründen ergibt sich die Notwendigkeit, eine Prototypanwendung zu entwickeln, die FHIR-konforme Testdaten entsprechend den im Rahmen des MII–Implementierungsleitfadens festgelegten strukturellen und semantischen Anforderungen generiert und eine größere Flexibilität im Vergleich zu den bestehenden Generatoren bietet.

3.3 FHIR–Server

In diesem Abschnitt werden zwei FHIR–Server vorgestellt, die nicht kommerziell sind und frei zur Nutzung zur Verfügung stehen. Außerdem werden die Vorteile des einen im Vergleich zum anderen Server angegeben.

3.3.1 Blaze–FHIR–Server

Der Blaze–FHIR–Server wurde speziell im Rahmen der Medizininformatik–Initiative (MII) entwickelt, um in allen Datenintegrationszentren eingesetzt zu werden. Blaze–FHIR–Server besitzt keine eigene Benutzeroberfläche, und der Zugriff auf die Daten erfolgt über HTTP–Anfragen [Smi24]. Um Zugriff auf die gewünschte Ressource zu erhalten, muss die Anfrage korrekt formuliert und die URL der Ressource auf dem Server angegeben werden. Dies kann für Nutzer, die keine technischen Kenntnisse zur Erstellung korrekter Serveranfragen besitzen, gewisse Schwierigkeiten mit sich bringen. Um die vollständige Liste der generierten Ressourcen zu sehen, wird folgende Anfrage ausgeführt:

```
1 http://<host>:<port>/fhir/<ResourceType>
```

Wenn man den Inhalt einer bestimmten Ressource sehen möchte, wird zusätzlich die ID dieser Ressource angegeben:

```
1 http://<host>:<port>/fhir/<ResourceType>/<id>
```

3.3.2 HAPI–FHIR–Server

Der HAPI–FHIR–Server ist der beliebteste Open–Source–Server und eignet sich optimal für kleine Projekte sowie für die Entwicklung und Implementierung wissenschaftlicher Lösungen im Gesundheitswesen [Ryb25]. Im Gegensatz zum Blaze–FHIR–Server verfügt er über eine benutzerfreundliche Oberfläche mit eigenen Bereichen für jeden Ressourcentyp (siehe Abbildung 3.3). Die an den Server gesendeten Ressourcen werden automatisch in den entsprechenden Abschnitt einsortiert. Dadurch ist es nicht notwendig, HTTP–Anfragen zu stellen, um auf die Ressourcen zuzugreifen. Um die Liste der Ressourcen auf dem Server zu sehen, genügt es, den entsprechenden Abschnitt aufzurufen.

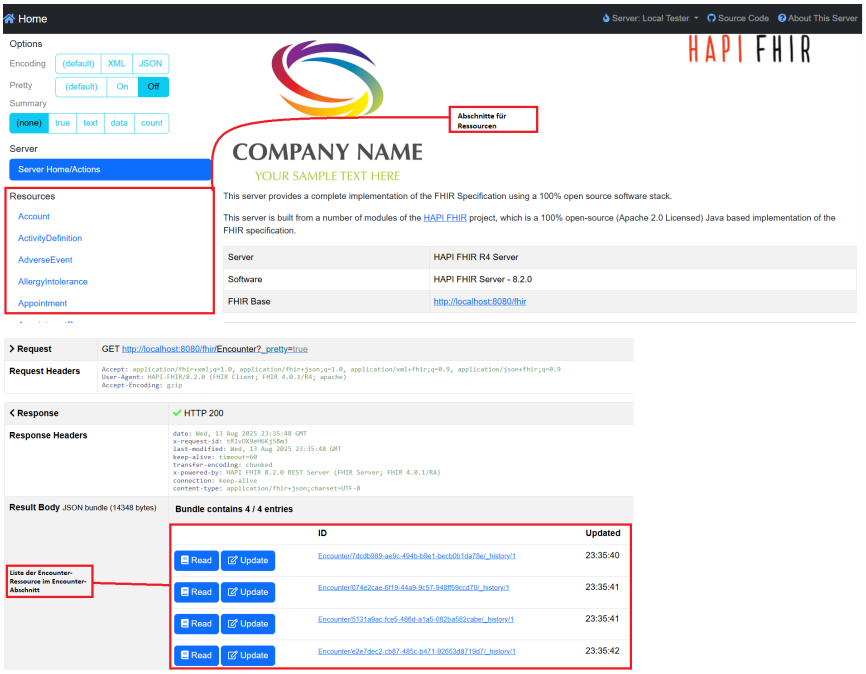


Abbildung 3.3: Web–Interface von HAPI–FHIR–Server

4 Konzept

In diesem Kapitel wird eine prototypische Anwendung vorgestellt, die es ermöglicht, Testdaten gemäß dem FHIR–Standard zu generieren. In der Abbildung 4.1 ist eine Übersicht der Funktionalität der Anwendung dargestellt: Basierend auf den eingestellten Parametern wird die Generierung ausgelöst, die generierten Daten auf der Benutzeroberfläche angezeigt und an den FHIR–Server gesendet. Der FHIR–Server speichert alle generierten Daten für eine weitere Validierung mithilfe des FHIR–Validators, um die Übereinstimmung der generierten Profile von Ressourcen mit dem MII–Implementierungsleitfaden nachzuweisen.

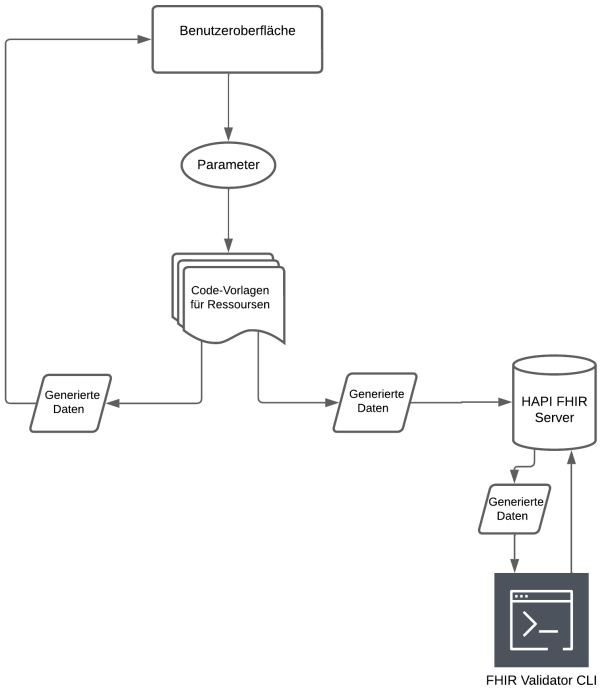


Abbildung 4.1: Systemarchitektur und Validierungsprozess

Der im Rahmen der Medizininformatik–Initiative wichtigste standardisierte Datensatz ist der Kerndatensatz. Er umfasst Basis- und Erweiterungsmodule. Die Basismodule sind fachübergreifend definiert und enthalten zentrale Informationen, die für die klinische Versorgung und medizinische Forschung relevant sind. Die Erweiterungsmodule hingegen decken speziellere Fachgebiete ab und werden je nach Forschungsanforderungen ergänzt und angepasst [Med25a].
Die entwickelte Anwendung generiert Ressourcenprofile, die den Profilen der Basismodule des Kerndatensatzes entsprechen, nämlich dem Patient–Profil aus dem Modul Person, dem Encounter–Profil aus dem Modul Fall sowie dem Observation–Profil aus dem Modul Laborbefund.

4.1 Benutzeroberfläche

In der Benutzeroberfläche der Anwendung gibt es mehrere Felder, in denen Parameter für jedes Profil ausgewählt werden können, auf deren Grundlage die Generierung erfolgen soll (siehe Abbildung 4.2). Für das Patient–Profil sind folgende Felder vorhanden:

- **Anzahl** der Patienten
- **Alter oder Altersintervall:** Im Fall der Generierung mehrerer Patienten erscheint die Möglichkeit, das minimale und maximale Alter der generierten Patienten zu bestimmen.
- **Geschlecht** der generierten Patienten.

Für das Encounter–Profil steht die Auswahl zur Verfügung:

- Konkrete **Art des medizinischen Falls**
- **Diagnosetyp:** Die Angabe des gewünschten Wertes im *diagnosis*–Element innerhalb des Encounter–Profils.
- **Aufnahmeanlass:** Die Angabe eines konkreten Grundes für die Hospitalisierung des Patienten.
- **Entlassungsgrund–Extensions:** Diese Felder ermöglicht es dem Benutzer, einen bestimmten Zustand des Patienten bei seiner Entlassung aus der medizinischen Einrichtung oder eine Entlassungsursache anzugeben.

Und schließlich gibt es für das Observation–Profil folgende Felder:

- **Labordaten:** In diesem Feld gibt der Benutzer die gewünschte Art der Laboruntersuchung an, die für den generierten Patienten durchgeführt wird.
- **Laborwert:** Dieses Feld ermöglicht es, einen konkreten Zahlenwert für die Laboruntersuchung anzugeben, der nach der Generierung im Profil erscheint.
- **QuelleKlinischesBezugsdatum-Extension:** Dieses Feld erlaubt dem Benutzer, aus Werten auszuwählen, die Informationen zum Datum im *effectiveDateTime*–Element übermitteln.

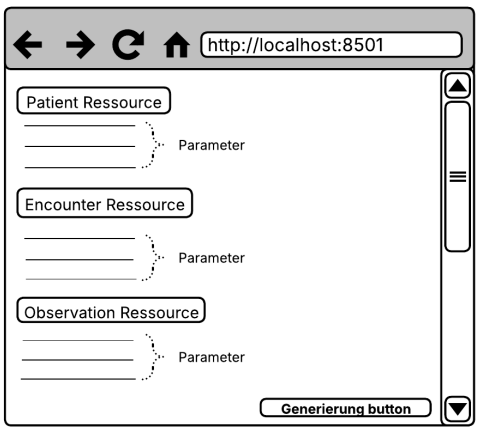


Abbildung 4.2: Parameterzuweisung für Ressourcen durch Benutzeroberfläche der Anwendung

Nach der Angabe den Parametern wird die Generierung ausgelöst, und die generierten Profile werden im JSON–Format auf der Benutzeroberfläche angezeigt.

4.2 Code–Vorlagen für Ressourcen

In diesem Schritt werden für jede generierte Ressource Vorlagen erstellt, innerhalb derer die Generierungslogik definiert wird. Der Generator erzeugt bereits valide Ressourcen wie Patient, Encounter und Observation, da die Generierungslogik auf den Anforderungen des MII–Implementierungsleitfadens basiert. Somit werden für jede Ressource alle Vorschriften (Constraints) eingehalten, und in die Elemente der Ressourcen werden nur erlaubte Werte (Codesysteme und Valuesets) eingefügt.
Um auf eine bestimmte Ressource innerhalb einer anderen Ressource zu verweisen, muss zunächst die referenzierte Ressource erstellt werden. Aus diesem Grund durchläuft jeder Generierungszyklus eine bestimmte Reihenfolge (siehe Abbildung 4.4). In der Abbildung 4.3 sind die Beziehungen zwischen den Ressourcen dargestellt, die über Elemente vom Typ *Reference* angegeben werden. Zuerst wird eine Organization–Ressource erstellt, danach eine Patient–Ressource, in der das *managingOrganization*–Element enthalten ist. Dieses Element dient dazu, die Organisation anzugeben, bei der der Patient registriert ist und medizinische Leistungen erhält. Hier wird ein Verweis auf die zuvor erstellte Organization–Ressource eingefügt.
Nachdem ein bestimmter Patient angelegt wurde, wird für ihn ein konkreter Fall der medizinischen Versorgung mit Hilfe der Encounter–Ressource erzeugt. Auch in dieser Ressource gibt es ein *diagnosis*–Element, das auf eine konkrete Diagnose oder einen Zustand verweist, der mit diesem Besuch in Verbindung steht. Deshalb wird zunächst eine Condition–Ressource erstellt, auf die dann bei der Erstellung der Encounter–Ressource verwiesen wird. Um anzugeben, dass dieser medizinische Besuch zu einem bestimmten Patienten gehört, wird innerhalb der Encounter–Ressource das *subject*-Element verwendet, das eine Referenz auf den in diesem Zyklus generierten Patienten enthält. Die letzte Ressource, die für den Patienten generiert wird, ist die Observation–Ressource, die Laborergebnisse und unterschiedliche Informationen zur durchgeführten Analyse darstellt. Sie wird nach Patient und Encounter erzeugt, da sie sowohl auf den Patienten (über das *subject*–Element) als auch auf den zuvor erstellten Encounter (über das *encounter*–Element) verweist. Damit ist ein vollständiger Erstellungszyklus für einen Patienten abgeschlossen. Abhängig von dem angegebenen Parameter zur Anzahl der Patienten wird dieser Zyklus entsprechend oft wiederholt.

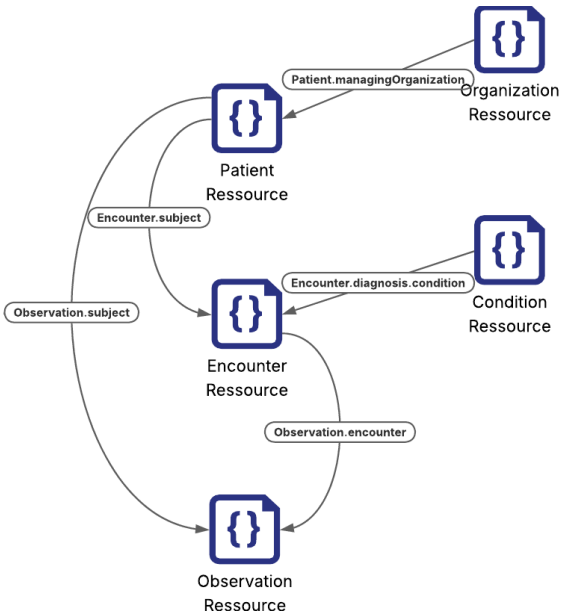


Abbildung 4.3: Reihenfolge der generierten Ressourcen in einem Generierungszyklus

4.3 Kommunikation mit dem Server

In dieser Arbeit wird der HAPI–FHIR–Server zur Speicherung der generierten Daten verwendet. Unmittelbar nach der Generierung einer Ressource wird diese bei jedem neuen Generierungszyklus an den Server gesendet. In der Abbildung 4.4 sind alle Prozesse der Datenübertragung an den Server in der Reihenfolge ihrer Generierung dargestellt.

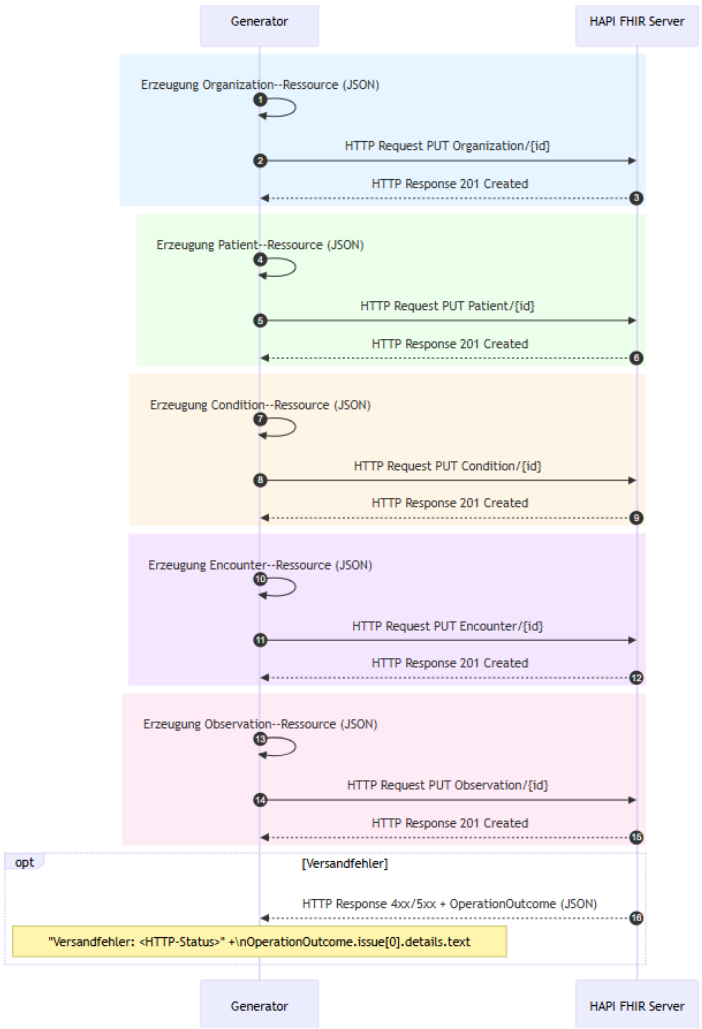


Abbildung 4.4: UML–Sequenzdiagramm

Mithilfe der PUT–Operation wird eine HTTP–Anfrage an den Server gestellt. Bei erfolgreicher Übermittlung der Ressource und Erstellung an den Server erfolgt eine HTTP–Antwort mit dem Statuscode 201 (Created). Im Falle eines Fehlers sendet der Server einen Statuscode der Serie 400 oder 500, je nach Art des Fehlers. Statuscodes der 400er–Serie weisen auf Fehler auf der Seite des Clients hin und erfordern eine Korrektur durch den Anfragenden. Im Gegensatz dazu signalisieren Statuscodes der 500er–Serie interne Fehler auf dem Server und zeigen an, dass die Probleme auf Serverseite liegen [Gup24]. Tritt während der Übertragung ein Fehler auf, sendet der Server eine HTTP–Antwort und teilt den konkreten Fehler in Textform mit, einschließlich einer Beschreibung.

4.4 FHIR Validator CLI

Um sich von der Richtigkeit einer bestimmten Ressource zu überzeugen, wurden sogenannte Validatoren entwickelt, mit denen Ressourcen auf Übereinstimmung mit der FHIR–Spezifikation geprüft werden können. Zudem kann der Validator so konfiguriert werden, dass er im Hinblick auf die FHIR–Version und den Implementierungsleitfaden überprüft, um genau ein bestimmtes Profil einer konkreten Ressource zu prüfen, das im Rahmen dieses Implementierungsleitfadens eigene Einschränkungen aufweist.
Zur Überprüfung der in dieser Arbeit generierten Profile wird der FHIR Validator CLI verwendet. Um dieses Werkzeug zu nutzen, muss es lokal installiert und über die Kommandozeile aufgerufen werden. Um ein lokal generiertes Profil zu prüfen, muss außerdem das Paket des Implementierungsleitfadens heruntergeladen werden, dessen Anforderungen von den generierten Profilen erfüllt sein müssen. Die Pakete sind über die Online–Plattform Simplifier (https://simplifier.net/) such– und herunterladbar.
Als Ergebnis zeigt der Validator verschiedene Arten von Meldungen mit unterschiedlichem Schweregrad an. Unter ihnen ist die wichtigste die *Error–Meldung*, die erscheint, wenn im zu prüfenden Profil ein Verstoß vorliegt, der der FHIR–Spezifikation widerspricht. Im Gegensatz dazu werden *Warn-* und *Informationsmeldungen* separat betrachtet und benötigen eine menschliche Bewertung, da sie nicht immer auf einen Fehler hinweisen, sondern Empfehlungen enthalten oder auf Besonderheiten in den Daten hinweisen. Zum Beispiel, wenn ein bestimmter Wert für ein Element in der Ressource nicht zur Basisspezifikation FHIR gehört, aber im Rahmen eines bestimmten Projekts zulässig ist [KBV24].

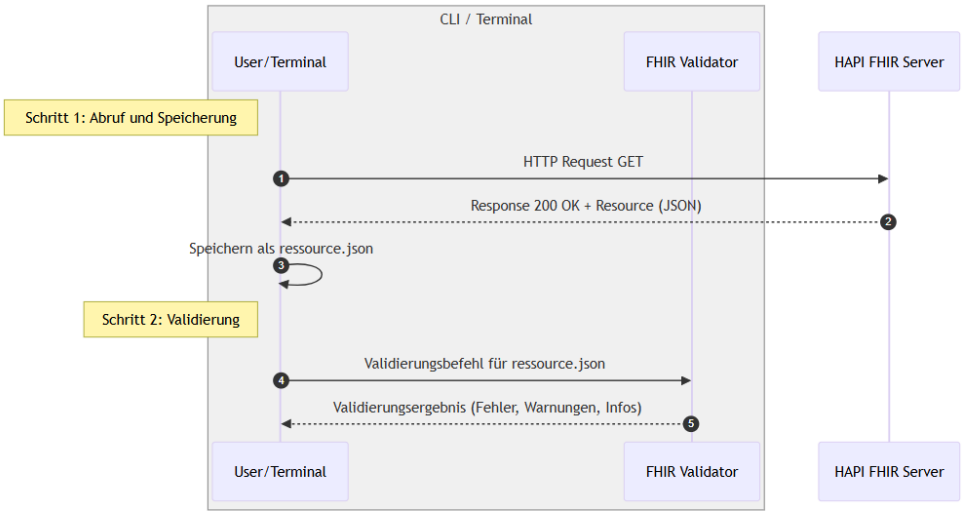


Abbildung 4.5: Use–Case–Diagramm

In der Abbildung 4.5 ist die Abfolge der Schritte zur Nutzung des FHIR–Validators dargestellt. Bevor die Validierung aufgerufen wird, muss die auf dem Server befindliche Ressource lokal auf dem Computer gespeichert werden. Mittels der GET–Operation wird eine HTTP–Anfrage an den Server gestellt, der daraufhin den Inhalt der Ressource zusammen mit einer Antwort und dem Statuscode 200 (OK) zurücksendet. Der Inhalt der Ressource wird dann lokal als JSON–Datei gesichert. Nach dem Speichern der Ressource erfolgt der Aufruf des Validierungsbefehls. Der Validator prüft die Ressource hinsichtlich des erforderlichen Implementierungsleitfadens und liefert die Validierungsergebnisse zurück.

5 Implementierung

In diesem Kapitel wird der technische Teil der Entwicklung der prototypischen Anwendung behandelt, um die Generierungslogik und Funktionalität im Detail zu beschreiben. Die Implementierung umfasst die Modellierung der Ressourcen gemäß den Profilen der Medizininformatik–Initiative (MII) sowie eine Benutzeroberfläche zur Parametrisierung und Orchestrierung des Generierungsprozesses und die Übergabe der Ressourcen an einen HAPI–FHIR–Server über REST–Anfragen. Die Anwendung ist in Python (Version 3.12.3) implementiert und besteht aus mehreren Modulen, von denen jedes für einen eigenen Teil der Logik verantwortlich ist (siehe Abbildung 5.1). Außerdem werden die Ergebnisse der Generierung sowie die Überprüfung der Profile im Hinblick auf die entsprechenden Module aus dem MII–Implementierungsleitfaden präsentiert.

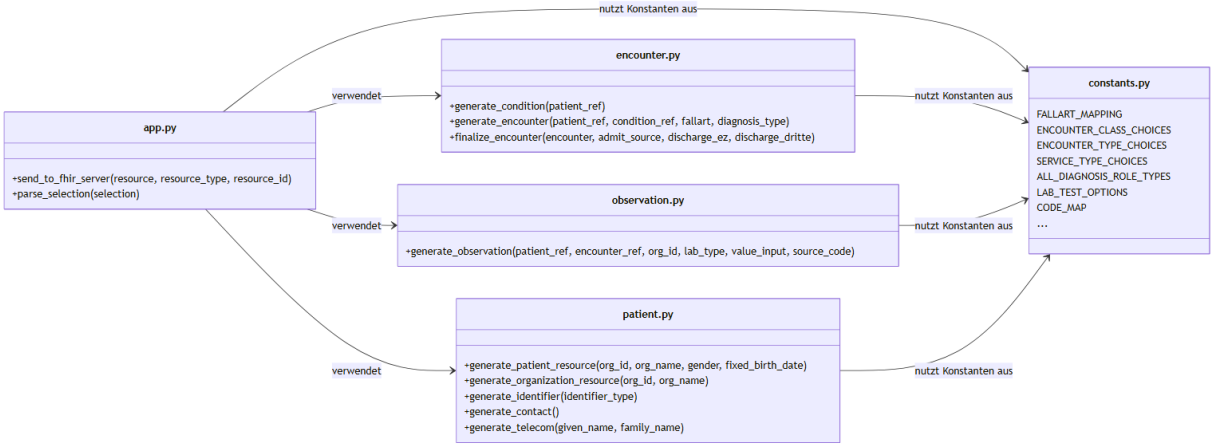


Abbildung 5.1: Klassendiagramm für Anwendungsmodule

5.1 Verwendete Bibliotheken und Werkzeugen

In diesem Abschnitt werden die bei der Implementierung verwendeten Python–Bibliotheken aufgelistet, von denen jede eine spezifische Funktion bei der Generierung, Strukturierung und Übertragung der FHIR–Ressourcen erfüllt. Darüber hinaus werden auch die Werkzeuge aufgeführt, die für die Speicherung und Validierung der generierten Ressourcen verwendet werden.

- **fhir.resources (Version 8.0.0)**: Eine zentrale Bibliothek, die auf dem Framework Pydantic basiert und FHIR-Ressourcen als Pydantic–Modelle implementiert. Dadurch wird eine valide und strukturierte Datendarstellung gemäß der FHIR-Spezifikation gewährleistet.
  - **Pydantic (Version 2.11.7)** Eine Bibliothek zur Definition und Validierung von Datenmodellen, bei der sogenannte Pydantic–Modelle eingesetzt werden. Pydantic–Modelle sind streng typisierte Datenobjekte, die automatisch prüfen, ob Eingabedaten den vorgegebenen Strukturen und Datentypen entsprechen.
- **Faker (Version 37.0.2)**: Diese Bibliothek dient zur Erzeugung von fiktiven Testdaten, insbesondere von Personendaten wie Namen, Adressen oder Geburtsdaten für Patienten.

- **requests (Version 2.32.3)**: Ermöglicht die Ausführung von HTTP–Anfragen und wird in dieser Arbeit eingesetzt, um die generierten Ressourcen nach ihrer Erstellung an den FHIR–Server zu übertragen.
- **UUID (Version 3.12.3)**: Standardmodul von Python, das zur Erstellung eindeutiger Identifikatoren verwendet wird. Jeder Ressource wird auf diese Weise eine individuelle ID zugewiesen.
- **random (Version 3.12.3)**: Standardmodul von Python, das für die Erzeugung von Zufallswerten oder die zufällige Auswahl von Elementen genutzt wird.
- **html (Version 3.12.3)**: Standardmodul von Python zur korrekten Maskierung und Umwandlung von HTML–Sonderzeichen, beispielsweise für die Darstellung im Feld `text.div`.
- **datetime (Version 3.12.3)**: Standardmodul von Python für die Verarbeitung von Datums– und Zeitangaben. Es wird genutzt, um etwa Geburtsdaten, Zeitintervalle oder Messzeitpunkte in den FHIR–Ressourcen korrekt zu erzeugen.
- **Streamlit (Version 1.44.1)**: Mit Hilfe dieser Bibliothek wird eine interaktive Weboberfläche für die Anwendung bereitgestellt, über die Nutzer Parameter auswählen und den Generierungsprozess steuern können.
- **HAPI–FHIR–Server (Version 8.2.0)**: Dient als lokaler Speicher– und Endpunktserver für die erzeugten Ressourcen und ermöglicht deren Verwaltung sowie Abruf über REST–Anfragen.
  - **Docker (Version 28.0.4)**: Das Werkzeug, mit dem Anwendungen samt aller wichtigen Zusatzprogramme und Bibliotheken in abgeschlossenen Bereichen ausgeführt werden können. Dadurch wird sichergestellt, dass Software unabhängig von der zugrunde liegenden Systemkonfiguration konsistent läuft. In dieser Arbeit wird Docker verwendet, um den HAPI–FHIR–Server schnell und reproduzierbar bereitzustellen.
- **FHIR Validator CLI (Version 6.5.28)**: Ein Validierungswerkzeug, das die erzeugten Ressourcen auf die Übereinstimmung mit den offiziellen Profilen verschiedener FHIR–Implementierungsleitfäden überprüft und detailliertes Feedback zu deren Konformität liefert.

5.2 Verwendete Konstanten

Konstanten dienen als vordefinierte Werte für Ressourcenelemente und werden in Form von Listen oder Wörterbüchern bereitgestellt. Ein Großteil dieser Konstanten basiert auf zugelassenen CodeSystemen und ValueSets, da viele Elemente nur einen begrenzten Satz möglicher Werte annehmen dürfen. Daneben existieren jedoch auch fiktive Beispieldaten, die für Elemente verwendet werden können, bei denen keine festen Vorgaben hinsichtlich der zulässigen Werte bestehen. Zusätzlich stehen spezielle Wörterbücher zur Verfügung, mit denen sich bestimmte Anwendungsfälle oder Parameter gezielt modellieren lassen. Nachfolgend sind alle verwendeten Sets mit einer kurzen Beschreibung aufgeführt:

`CONTACT_PURPOSE_CHOICES` : Zulässige Kontaktarten für Organisationen (z.B. administrativ).

`ORG_TYPE_CHOICES` : Organisationstypen wie Krankenhaus, Labor, Pflegeeinrichtung.

`IDENTIFIER_METADATA` Definierte Patienten-Identifier (z.B. KVZ10, interne Fallnummer).

`MARITAL_STATUS_OPTIONS` Familienstand des Patienten (ledig, verheiratet, geschieden, verwitwet).

`FALLART_MAPPING` Mapping für Fallarten (ambulant, stationär) inkl. `class`, `type`, Extensions.

`SERVICE_TYPE_CHOICES` Fachabteilungen der Einrichtung (z.B. Kardiologie, Radiologie).

ENCOUNTER_PARTICIPANT_TYPE_CHOICES Rollen der beteiligten Personen (z.B. Arzt, Pflegekraft).

ALL_DIAGNOSIS_ROLE_TYPES Zulässige Diagnoserollen (Hauptdiagnose, Nebendiagnose usw.).

ADMIT_SOURCE_OPTIONS Strukturierte Aufnahmegründe (z.B. Einweisung, Geburt).

ENTLASSUNG_ERSTE_ZWEITE / DRITTE Entlassungsgründe nach DKGEV-Spezifikation.

ENCOUNTER_CLASS_CHOICES Begegnungsklassen (ambulant, stationär, Notfall).

ENCOUNTER_TYPE_CHOICES Differenzierte Encounter-Typen nach Art des Kontakts.

LAB_TEST_OPTIONS Auswahl möglicher Labortests (z.B. Glukose, HDL/LDL).

CODE_MAP Metadaten zu Labortests: LOINC-Code, Einheit, Referenzwerte.

5.3 Benutzeroberfläche

In diesem Abschnitt werden die Aktionen behandelt, die über die Benutzeroberfläche der Anwendung ausgeführt werden, wie zum Beispiel das Festlegen von Parametern für die generierten Ressourcen und deren Übergabe innerhalb des Generierungszyklus.

5.3.1 Parameterzuweisung

Parameter steuern die Generierung von FHIR-Ressourcen, damit die Daten

- **kontrollierbar und reproduzierbar** sind: zur Verwendung fixer Werte in bestimmten Elementen,
- **variabel** sind: zur Erzeugung von Ressourcen für unterschiedliche medizinische Szenarien,
- **profilkonform** sind: Auswahl von vorgegebenen Elementwerten gemäß dem MII.

```
import streamlit as st
# ...
st.title("FHIR Person Generator")
st.header("Patient Resource")

num_resources = st.number_input(
    "Anzahl der zu generierenden Person-Ressourcen:",
    min_value=1, max_value=100, value=1)

if num_resources == 1:
    fixed_age = st.number_input(
        "Geben Sie das Alter der Person an:",
        min_value=0, max_value=120, value=30)
    min_age = max_age = fixed_age
else:
    min_age = st.slider("Mindestalter", 0, 100, 18)
    max_age = st.slider("Das maximale Alter", min_age, 120, 90)

gender_option = st.selectbox(
    "Geschlecht der Person angeben",
    ["random", "male", "female", "other", "unknown"])
```

Listing 5.1: Parameter von Patient–Ressource

Listing 5.1 zeigt die Parametrisierung der Patient–Ressource in der Streamlit–Oberfläche. Der wichtigste Parameter ist die Anzahl der zu generierenden Patienten, die mit `st.number_input()` festgelegt wird. Diese Methode erzeugt ein Eingabefeld, in dem eine numerische Größe definiert werden kann. Wenn die Anzahl 1 beträgt, wird genau ein Generationszyklus ausgeführt und eine einzelne Patient-Instanz erzeugt. In diesem Fall kann über `st.number_input()` auch ein festes Alter (`fixed_age`) angegeben werden. Wenn die gewünschte Anzahl größer als 1 ist, werden die Parameter `min_age` und `max_age` mit

`st.slider()` bestimmt. Dieses Widget stellt einen Schieberegler bereit, mit dem Intervallwerte komfortabel gewählt werden können. Auf diese Weise wird das Alter bei der Generierung zufällig innerhalb des definierten Bereichs gesetzt. Der letzte Parameter ist das Geschlecht, das über `st.selectbox()` angegeben wird. Diese Methode erzeugt ein Dropdown–Menü, aus dem eine einzelne Option aus einer vordefinierten Liste ausgewählt werden kann. Damit lässt sich das Geschlecht entweder als fixer Wert oder, falls die Option *random* gewählt wird, stochastisch bestimmen.

```
# ... Parameter von Patienten-Ressource
ADMIT_SOURCE_OPTIONS = [
    {"code": "E", "display": "Einweisung durch einen Arzt"},
    #...
    {"code": "G", "display": "Geburt"},
    {"code": "B", "display": "Begleitperson oder mitaufgenommene Pflegekraft"}]
st.header("Encounter Resource")
fallart_option = st.selectbox("Fallart (Art des medizinischen Falls):", list(FALLART_MAPPING.keys()))
diagnosis_type_option = st.selectbox(
    "Diagnosis type (optional):",
    options=["zufällig wählen"] + [f'{entry["display"]} ({entry["code"]})' for entry in
    ALL_DIAGNOSIS_ROLE_TYPES])
admit_source_option = st.selectbox("Aufnahmeanlass (admitSource):", options=[f'{x["display"]}
({x["code"]})' for x in ADMIT_SOURCE_OPTIONS])
discharge_ez_option = st.selectbox("Entlassungsgrund - ErsteUndZweiteStelle:", options=[f'{x["display"]}
({x["code"]})' for x in ENTLASSUNG_ERSTE_ZWEITE])
discharge_dritte_option = st.selectbox("Entlassungsgrund - DritteStelle:", options=[f'{x["display"]}
({x["code"]})' for x in ENTLASSUNG_DRITTE])
```

Listing 5.2: Parameter von Encounter–Ressource

In Listing 5.2 wird gezeigt, wie die Parameter für die Encounter–Ressource ausgewählt werden können. Für jeden dieser Parameter wird mit `st.selectbox()` ein Dropdown-Menü erstellt, in dem der Benutzer aus vorgegebenen Listen passende Werte auswählen kann. So wird mit `fallart_option` die Art des medizinischen Falls festgelegt. Die verfügbaren Werte stammen dabei direkt aus der Mapping–Struktur `FALLART_MAPPING`. Für den Diagnosetyp (`diagnosis_type_option`) gibt es zusätzlich zur Option „zufällig wählen“ eine Liste aller möglichen Diagnoserollen (`ALL_DIAGNOSIS_ROLE_TYPES`), die jeweils mit Code und Bezeichnung angezeigt werden.
Der Parameter `admit_source_option` beschreibt den Aufnahmeanlass und bezieht seine Werte aus der Liste `ADMIT_SOURCE_OPTIONS`, in der sowohl Bezeichnungen als auch die zugehörigen Codes enthalten sind. Schließlich ermöglichen `discharge_ez_option` und `discharge_dritte_option` die Auswahl des Entlassungsgrundes. Während ersterer auf die Liste `ENTLASSUNG_ERSTE_ZWEITE` zurückgreift, basiert letzterer auf den Einträgen von `ENTLASSUNG_DRITTE`.

```
# ... Parameter von Patient-Ressource
# ... Parameter von Encounter-Ressource
st.header("Observation Resource")
selected_lab_test = st.selectbox("Labordaten fuer:", list(LAB_TEST_OPTIONS.keys()))
value_input = st.text_input("Laborwert manuell eingeben (leer lassen fuer} Zufallswert):")
source_code_display = st.selectbox(
    "QuelleKlinischesBezugsdatum:",
    ["Specimen collection date (observable entity)",
     "Date sample received in laboratory (observable entity)"]
)
```

Listing 5.3: Parameter von Encounter–Ressource

Schließlich besteht auch die Möglichkeit, Parameter für die gewünschte Generierung der Observation-Ressource zu konfigurieren (siehe Listing 5.3). Über das Dropdown-Menü `st.selectbox()` wird der Typ des medizinischen Labortests ausgewählt, der generiert werden soll. Abhängig vom gewählten Testtyp kann zusätzlich ein konkreter Zahlenwert für das Ergebnis des Labortests angegeben werden. Außerdem stehen zwei vordefinierte Optionen für die *QuelleKlinischesBezugsdatum*–Extension zur Auswahl, die im Rahmen der MII–Spezifikation vorgegeben sind.

5.3.2 Generierungsaufruf

Nach der Auswahl der Parameter in der Benutzeroberfläche kann die eigentliche Generierung ausgelöst werden (siehe Listing 5.4).

```
fake = Faker('de_DE')
# Parameterübergabe für Ressourcen (oben im UI definiert)
# ...
generate_button = st.button("Generieren und an einen FHIR-Server senden")
if generate_button:
    with st.spinner("Generierung und Sendung von Daten..."):
        for i in range(num_resources):
            org_id = str(uuid.uuid4())
            org_name = fake.company()
            organization = generate_organization_resource(org_id, org_name)
            # ...
            birth_date = fake.date_of_birth(
                minimum_age=min_age, maximum_age=max_age).isoformat()
            gender = None if gender_option == "random" else gender_option

            patient = generate_patient_resource(
                org_id=org_id, org_name=org_name,
                gender=gender, fixed_birth_date=birth_date)
```

Listing 5.4: Start des Generierungszyklus

Zunächst wird die Locale `Faker('de_DE')` gesetzt, um realistische deutsche Testdaten (z. B. Firmennamen oder Geburtsdaten) zu erzeugen. Anschließend wird ein Button definiert, über dessen Betätigung der Generierungsprozess gestartet wird. Zur Visualisierung wird `st.spinner()` eingesetzt, das während der Laufzeit einen Ladeindikator anzeigt. Danach wird eine `for`–Schleife aufgerufen, die sich exakt so oft wiederholt, wie es durch den zuvor gesetzten Parameter `num_resources` bestimmt ist. In jeder Iteration werden einzelne Elemente erzeugt, die als Parameter für die Generierungsfunktionen dienen. Für den Aufruf der Funktion `generate_organization_resource()` wird mit `uuid.uuid4()` eine eindeutige ID erstellt und mit `fake.company()` ein fiktiver Organisationsname generiert. Diese Werte werden an die Funktion übergeben, wodurch eine beispielhafte Organization–Ressource erzeugt wird. Für die Patient–Ressource wird ein Geburtsdatum erzeugt, das den zuvor ausgewählten Altersparametern entspricht, und anschließend im ISO–Format gespeichert. Das Geschlecht wird gemäß der Benutzerauswahl gesetzt; falls die Option „random“ gewählt ist, wird der Wert `None` übergeben, sodass die Generatorfunktion das Geschlecht zufällig bestimmen kann. Schließlich werden alle relevanten Parameter (*org_id*, *org_name*, *gender*, *birth_date*) an die Funktion `generate_patient_resource()` übergeben.

```
def parse_selection(selection: str):
    if " (" in selection and selection.endswith(")"):
        display, code = selection.rsplit(" (", 1)
        return code.rstrip(")"), display
    return "", selection

# Parameterübergabe fuer Ressourcen (oben im UI definiert)
# ...
generate_button = st.button("Generieren und an einen FHIR-Server senden")
if generate_button:
    with st.spinner("Generierung und Sendung von Daten..."):
        for i in range(num_resources):
            # Aufruf von Generierung der Organization--Ressource
            # Aufruf von Generierung der Patient--Ressource
            condition = generate_condition(f"Patient/{patient.id}")
            encounter = generate_encounter(
                patient_ref=f"Patient/{patient.id}",
                condition_ref=f"Condition/{condition.id}",
                fallart=fallart_option,
                diagnosis_type=diagnosis_type_option)

            admit_code, admit_display = parse_selection(admit_source_option)
            discharge_ez_code, discharge_ez_display = parse_selection(discharge_ez_option)
            discharge_dritte_code, discharge_dritte_display = parse_selection(discharge_dritte_option)

```

```
26              encounter_dict = finalize_encounter(encounter,
27              admit_source={"code": admit_code, "display": admit_display},
28              discharge_ez={"code": discharge_ez_code, "display": discharge_ez_display},
29              discharge_dritte={"code": discharge_dritte_code, "display": discharge_dritte_display})
30
31               observation = generate_observation(
32                      patient_ref=f"Patient/{patient.id}",
33                      encounter_ref=f"Encounter/{encounter.id}",
34                      org_id=org_id,
35                      lab_type=selected_lab_test,
36                      value_input=value_input,
37                      source_code=source_code_display)
```

Listing 5.5: Fortsetzung des Generierungsaufruf

Als Nächstes wird innerhalb der `for`–Schleife eine beispielhafte Condition–Ressource erstellt (siehe Listing 5.5). Die Funktion zur Erzeugung dieser Ressource erhält als Parameter die ID des zuvor generierten Patienten. Nach der Erstellung der Condition–Ressource wird eine Funktion aufgerufen, die den größten Teil der Encounter–Ressource erzeugt. Diese Funktion verwendet die Patienten– und Condition–ID sowie die bereits vor der Generierung übergebenen Parameter *Fallart* und *Diagnosetyp*.

Um auch die weiteren Parameter zu berücksichtigen und die Encounter–Ressource zu vervollständigen, wird anschließend die Funktion `finalize_encounter()` aufgerufen. An diese Funktion werden das zuvor erzeugte Encounter–Objekt sowie die ebenfalls vor der Generierung gewählten Parameter (Aufnahmeanlass und Entlassungsgründe) übergeben. In der Benutzeroberfläche werden die Felder für diese Parameter in der Form **Displayname (Code)** dargestellt. Dieses Format ermöglicht ein einfaches Extrahieren der Werte. Das Extrahieren übernimmt die Funktion `parse_selection()`, die den gewählten Eintrag in Displayname und Code trennt. Mit Hilfe dieser Funktion werden die ausgewählten Parameter in Display– und Codewerte zerlegt und an die Funktion `finalize_encounter()` weitergegeben, um die endgültige Encounter–Ressource zu ergänzen.
Der letzte Aufruf ist der Funktionsaufruf zur Generierung für Observation–Ressource. Als Parameter nimmt diese Funktion auch die zugewiesenen Generierungsparameter sowie einige zuvor generierte Ressourcenelemente entgegen, um sie innerhalb ihrer Elemente für die Referenzierung zu verwenden.

5.4 Generierungsfunktionen

In diesem Abschnitt werden die Funktionen zur Generierung von Ressourcen, die nach der Parametrisierung aufgerufen werden und die Struktur der Ressource aufbauen, ausführlich behandelt. Die Funktionen dienen dabei als Vorlagen für die zu generierenden Ressourcen und bestimmte Ressourcenprofile. Es werden auch Funktionen betrachtet, die spezifische Elemente der Ressourcen generieren, da diese Elemente häufig aus mehreren Unterelementen bestehen und es daher zweckmäßiger ist, bestimmte Elemente separat innerhalb einer eigenen Funktion zu erstellen.

5.4.1 Funktionen für Organization–Ressource

```
1  fake = Faker('de-DE')
2
3  def generate_contact():
4      purpose_choice = random.choice(CONTACT_PURPOSE_CHOICES)
5      return ExtendedContactDetail.model_construct(
6          purpose=CodeableConcept.model_construct(
7              coding=[Coding.model_construct(
8                  system="http://terminology.hl7.org/CodeSystem/contactentity-type",
9                  code=purpose_choice["code"],
10                 display=purpose_choice["display"])]),
11         telecom=[
12             ContactPoint.model_construct(system="phone",value=fake.phone_number(),use="work"),
13             ContactPoint.model_construct(system="email",value=fake.email(),use="work")],
14         address=Address.model_construct(
15             line=[fake.street_address()], city=fake.city(), postalCode=fake.postcode(), country="DE"))
16
17 def generate_organization_resource(org_id, org_name):
```

```
18      org_type = random.choice(ORG_TYPE_CHOICES)
19      return Organization.model_construct(
20          id=org_id,
21          name=org_name,
22          type=[CodeableConcept.model_construct(
23                  coding=[
24                      Coding.model_construct(
25                          system="http://terminology.hl7.org/CodeSystem/organization-type",
26                          code=org_type["code"],
27                          display=org_type["display"])])],
28          contact=[generate_contact()],
29          text=Narrative.model_construct(
30              status="generated",
31              div=f"<div xmlns='http://www.w3.org/1999/xhtml'>Generated
                Organization:{html.escape(org_name)}</div>"))
```

Listing 5.6: Erstellung der Organization–Ressource

Im Listing 5.6 wird die Funktion zur Generierung der eigentlichen Organization–Ressource dargestellt. Darüber hinaus existiert die Funktion `generate_contact()`, die das *contact*–Element innerhalb der Organization–Ressource mitsamt allen zugehörigen Unterelementen erzeugt. Zunächst wählt diese Funktion mit Hilfe der Methode `choice()` ein bestimmtes Wertepaar (Code und entsprechender Display–Wert) zufällig aus der Liste `CONTACT_PURPOSE_CHOICES` aus. Diese Liste enthält die zulässigen Werte und repräsentiert die Typen von Kontaktverbindungen. Nach der Auswahl wird unter Verwendung der Klasse `ExtendedContactDetail` und der Methode `model_construct()` ein entsprechendes Pydantic–Modell für dieses Element erzeugt. Das Unterelement *purpose* übernimmt dabei die zufällig ausgewählten Werte aus der Liste. Das nächste Unterelement *telecom* übergibt die Kontaktdaten und verwendet Faker, um eine fiktive Telefonnummer und eine E–Mail–Adresse zu generieren. Schließlich werden im Unterelement *address* ebenfalls fiktive Daten zur Darstellung des Standorts der Organisation erzeugt. Die Funktion wird innerhalb der Funktion `generate_organization_resource()` im *contact*–Element aufgerufen.
Die Funktion `generate_organization_resource()` selbst wählt ebenfalls zunächst zufällig einen zulässigen Wert aus einer Liste `ORG_TYPE_CHOICES` zur Angabe des Organisationstyps aus und erstellt anschließend ein Pydantic–Modell der Klasse Organization, wobei ihr eine bestimmte ID und ein Name mittels `org_id` und `org_name` zugewiesen werden. Diese Werte sind bereits vor dem Aufruf der Funktion definiert, da sie als Parameter übergeben werden müssen (siehe Listing 5.4). Darüber hinaus enthält die Ressource ein *text*–Element vom Typ `Narrative`, das zur Angabe einer textuellen Beschreibung der Ressource dient. In diesem Beispieltext wird lediglich der Name der erstellten Organisation angegeben. Für die Übergabe wird dabei die Methode `escape()` verwendet. Sie sorgt für die korrekte Maskierung, da einige Organisationsnamen Apostrophe, kaufmännische Und–Zeichen oder andere unzulässige Zeichen enthalten können. Ohne diese Maskierung könnte der Server den Ressourceneintrag zurückweisen, sodass der generierte Datensatz kein *text*–Element enthalten würde.

5.4.2 Funktionen für Patienten–Ressource

Nach der Erstellung der Organization beginnt die Generierung des Profils der Patient-Ressource. Hierfür werden zusätzliche Bibliotheken sowie fehlende Klassen aus der Bibliothek `fhir.resources` importiert, um die Patient-Ressource konstruieren zu können. Ebenso wie bei der Organization wurden auch für den Patienten einzelne Hilfsfunktionen definiert, die innerhalb der Hauptfunktion zur Generierung der Patient–Ressource aufgerufen werden, um komplexe Elemente mit ihren zahlreichen Unterelementen zu erzeugen (siehe Listing 5.7).
Als erstes kommt die Funktion `generate_identifier()` zum Einsatz, welche den Inhalt des *identifier*–Elements abhängig vom gewählten Identifikatortyp erstellt. Dieses Element repräsentiert nicht die interne ID der Ressource selbst, sondern die sogenannte Business-ID des Patienten.

```python
def generate_identifier(identifier_type):
    meta = random.choice(IDENTIFIER_METADATA[identifier_type])
    iknr_value = fake.unique.numerify(text="#########")
    if identifier_type == "KVZ10":
        kvz_value = fake.random_uppercase_letter() + fake.unique.numerify(text="#########")
        return Identifier(
            use="official",
            type=CodeableConcept(
                coding=[Coding(
                    system="http://fhir.de/CodeSystem/identifier-type-de-basis",
                    code=identifier_type,
                    display=IDENTIFIER_DISPLAY_MAP[identifier_type])]),
            system="http://fhir.de/sid/gkv/kvid-10",
            value=kvz_value,
            assigner=Reference(
                display=meta["assigner"],
                identifier={
                    "system": "http://fhir.de/sid/arge-ik/iknr",
                    "value": iknr_value}))
    else:
        return Identifier(
            use="official",
            type=CodeableConcept(
                coding=[Coding(
                    system="http://terminology.hl7.org/CodeSystem/v2-0203",
                    code=identifier_type,
                    display=IDENTIFIER_DISPLAY_MAP[identifier_type]
                )]),
            system=meta["system"],
            value=fake.unique.bothify(text="??##??##"),
            assigner=Reference(
                display=meta["assigner"],
                identifier={
                    "system": "http://fhir.de/sid/arge-ik/iknr",
                    "value": iknr_value}))

def generate_telecom(given_name,family_name):
    phone = ContactPoint(
        system="phone",
        value=fake.phone_number(),
        use="mobile" )
    domain = fake.free_email_domain()
    email_address = f"{given_name.lower()}.{family_name.lower()}@{domain}"
    email = ContactPoint(
        system="email",
        value=email_address,
        use="home")
    return [phone, email]
```

Listing 5.7: Funktion für Erstellung identifier– und telecom–Element

Vor der Konstruktion der Pydantic–Modelle wird aus der Liste `IDENTIFIER_METADATA` zufällig einer von zwei zulässigen, im Rahmen der MII definierten Werte ausgewählt, die unterschiedliche Identifikatortypen darstellen. Anschließend wird, abhängig vom Typ, ein *identifier*–Element mit allen dazugehörigen Unterelementen erzeugt. Im Fall der Krankenversichertennummer (KVZ10) muss ein strenges Format eingehalten werden, das mit einem Großbuchstaben beginnt und anschließend aus neun Ziffern besteht. Dies wird durch die Methoden `random_uppercase_letter()` und `numerify()` umgesetzt. Innerhalb des *identifier*–Elements befindet sich außerdem das Unterelement *assigner*, das Informationen über die Organisation enthält, die diesen Patientenidentifikator ausgestellt und verwaltet hat. Jede Organisation verfügt dabei über ein eigenes Institutionskennzeichen, das als ID fungiert. Zur Erzeugung dieses Wertes wird die Methode `numerify()` verwendet, womit ein entsprechendes `iknr_value` generiert wird.
Die zweite Hilfsfunktion, die ebenfalls das *telecom*–Element mit all seinen Unterelementen generiert, ist `generate_telecom()`. Diese Funktion nimmt anstelle von Parametern den Vor– und Nachnamen des Patienten entgegen und verwendet diese Daten zur Erstellung einer fiktiven E–Mail–Adresse für den Patienten. Darüber hinaus erzeugt die Funktion auch eine fiktive Telefonnummer für den Patienten.

```python
def generate_patient_resource(org_id, org_name, gender, fixed_birth_date):
    if gender is None:
        gender = random.choice(["male", "female", "other", "unknown"])
    if gender == "male":
        given_name = fake.first_name_male()
    elif gender == "female":
        given_name = fake.first_name_female()
    else:
        given_name = fake.first_name()
    family_name = fake.last_name()
    full_name = f"{given_name} {family_name}"
    gender = gender
    gender_extension = None
    if gender == "other":
        amtlich_choices = [
            {"code": "D", "display": "divers"},
            {"code": "X", "display": "unbestimmt"}]
        selected = random.choice(amtlich_choices)
        gender_extension = Extension(
            url="http://fhir.de/StructureDefinition/gender-amtlich-de",
            valueCoding=Coding(
                system="http://fhir.de/CodeSystem/gender-amtlich-de",
                code=selected["code"],
                display=selected["display"]))
    marital = random.choice(MARITAL_STATUS_OPTIONS)
    birth_date = fixed_birth_date or fake.date_of_birth(minimum_age=18, maximum_age=90).isoformat()
```

Listing 5.8: Funktion für Erstellung Patient–Ressource

Die Funktion `generate_patient_resource()` ist die Hauptfunktion, die den größten Teil der Patient–Ressource generiert (siehe Listing 5.8). Diese Funktion nimmt als Parameter die ID und den Namen der zuvor erzeugten Organization–Ressource sowie die bei der Parametrisierung gewählten Werte wie Geschlecht und Alter entgegen.

Bevor die Pydantic–Modellinstanz `Patient` erstellt wird, werden zunächst die für den Patienten relevanten Grunddaten generiert. Um die Generierung etwas logischer zu gestalten, wird in Abhängigkeit vom gewählten Geschlecht auch der Vorname des Patienten männlich oder weiblich erzeugt. Dafür greifen Bedingungen, die je nach Geschlecht die Methoden `first_name_male()` bzw. `first_name_female()` verwenden. Für den Patienten wird außerdem ein Familienname mit der Methode `last_name()` generiert. Für den Sonderfall, wenn das Geschlecht als `other` angegeben ist, wird zusätzlich eine deutsche Spezialisierung berücksichtigt. Hierbei wird mit einer Extension (`gender-amtlich-de`) eine amtliche Kodierung („divers“ oder „unbestimmt“) eingefügt, die die MII–Anforderungen erfüllt. Auch der Familienstand wird zufällig aus den vordefinierten `MARITAL_STATUS_OPTIONS` gewählt. Das Geburtsdatum wird entweder aus einem übergebenen Fixwert (`fixed_birth_date`) übernommen oder mit `fake.date_of_birth()` im Bereich von 18 bis 90 Jahren zufällig generiert.

```python
def generate_patient_resource(org_id, org_name, gender, fixed_birth_date):
#... (Grunddaten)
    patient = Patient.model_construct(
        id=str(uuid.uuid4()),
        meta=Meta.model_construct(profile=[
        "https://www.medizininformatik-initiative.de/fhir/core/modul-person/StructureDefinition/Patient"]),
        text=Narrative.model_construct(
            status="generated",
            div=f"<div xmlns='http://www.w3.org/1999/xhtml'>Generated person:
            {html.escape(full_name)}</div>" ),
        identifier=[generate_identifier(random.choice(list(IDENTIFIER_METADATA.keys())))],
        active=True,
        name=[HumanName.model_construct(
            use="official",
            family=family_name,
            given=[given_name])],
        gender=gender,
        gender__ext={"extension": [gender_extension]} if gender_extension else None,
        birthDate=birth_date,
        deceasedBoolean=False,
        telecom=generate_telecom(given_name, family_name),
        address=[Address.model_construct(
            use="home",
```

```
            type="both",
            line=[fake.street_address()],
            city=fake.city(),
            postalCode=fake.postcode(),
            country="DE")],
        maritalStatus=CodeableConcept.model_construct(
            coding=[Coding.model_construct(
                system="http://terminology.hl7.org/CodeSystem/v3-MaritalStatus",
                code=marital["code"],
                display=marital["display"])]),
        communication=[PatientCommunication.model_construct(
            language=CodeableConcept.model_construct(
                coding=[Coding.model_construct(
                    system="urn:ietf:bcp:47",
                    code="de",
                    display="German")]),
            preferred=True)],
        managingOrganization=Reference.model_construct(
            reference=f"Organization/{org_id}",
            display=org_name))
    return patient
```

Listing 5.9: Fortsetzung der Funktion für Erstellung der Patient–Ressource

Nach der Definition der benötigten Grunddaten wird mithilfe der Klasse `Patient` und der Methode `model_construct()` eine Pydantic–Modellinstanz der Patient–Ressource erstellt, in der alle ihre Elemente definiert werden (siehe Listing 5.9). Für den Patienten wird zunächst eine eindeutige ID mit der Methode `uuid4()` generiert. Im *meta*–Element wird unter Verwendung der Klasse `Meta` die URL des Profils angegeben, dem diese Ressource entspricht. Für die textuelle Darstellung dient zusätzlich das *text*–Element.

Darüber hinaus werden die Elemente *identifier* und *telecom* eingefügt, wobei jeweils die zuvor beschriebenen Hilfsfunktionen zur Generierung dieser Inhalte aufgerufen werden. In den Elementen *name*, *gender* und *birthDate* werden die zuvor bestimmten Grunddaten eingetragen. Um sicherzustellen, dass der generierte Patient als lebend gilt, erhält das Element *deceasedBoolean* stets den festen Wert `False`.

Der Wohnort des Patienten wird im *address*–Element mithilfe der Methoden von Faker erzeugt. Zusätzlich wird der Familienstand zufällig aus der vordefinierten Liste `MARITAL_STATUS_OPTIONS` ausgewählt. Die vom Patienten gesprochene Sprache wird standardmäßig als Deutsch angegeben und mit dem korrekten Codewert aus dem CodeSystem `urn:ietf:bcp:47` versehen, wie es von FHIR vorgeschrieben ist. Als letztes Element wird *managingOrganization* genutzt, in dem auf die zuvor generierte Organization–Ressource verwiesen wird.

5.4.3 Funktion für Condition–Ressource

Die nächste Beispielressource, die erzeugt wird, um in der anschließend generierten Encounter–Ressource darauf zu verweisen, ist die Condition–Ressource. Nach dem Import der benötigten Klassen zur Erstellung der Ressource nimmt die Funktion `generate_condition` (siehe Listing 5.10) die ID der zuvor erstellten Patient–Ressource entgegen, um darauf zu verweisen und die Zugehörigkeit zu einem bestimmten Patienten darzustellen.

```
def generate_condition(patient_ref: str) -> Condition:
    return Condition.model_construct(
        id=str(uuid.uuid4()),
        subject=Reference.model_construct(reference=patient_ref),
        code=CodeableConcept.model_construct(
            coding=[Coding.model_construct(
                system="http://snomed.info/sct",
                code="386661006",
                display="Fever")]),
        text=Narrative.model_construct(
            status="generated",
            div="<div xmlns='http://www.w3.org/1999/xhtml'>Condition: Fever</div>"))
```

Listing 5.10: Funktion für Erstellung der Condition–Ressource

Der klinische Inhalt wird im *code*–Element über ein *CodeableConcept* beschrieben, das ein einzelnes *Coding* aus dem Terminologiesystem SNOMED CT enthält. Hier wird exemplarisch der Code mit der Bezeichnung „Fever“ gesetzt.

5.4.4 Funktionen für Encounter–Ressource

Der zweite generierte Ressourcentyp, der ebenfalls den MII–Profilen entspricht, ist das Profil der Encounter-Ressource. Die erste Funktion zur Erstellung, `generate_encounter()`, erzeugt den größten Teil der Elemente dieser Ressource als Pydantic–Modell. Im Listing 5.11 ist diese Funktion dargestellt, und vor der eigentlichen Konstruktion des Pydantic–Modells wird die Logik zur Auswahl der Werte für die einzelnen Elemente festgelegt und die Grunddaten bestimmt. Die Funktion nimmt die IDs der zuvor erstellten Patient– und Condition–Ressourcen sowie die bei der Parametrisierung ausgewählten Werte wie *Fallart* und *Diagnosetyp* entgegen.

```
def generate_encounter(patient_ref: str, condition_ref: str, fallart: str, diagnosis_type: str) ->
Encounter:
    status_code = random.choice(["planned","in-progress", "onleave", "finished", "cancelled",
    "entered-in-error", "unknown"])
    random_service_type = random.choice(SERVICE_TYPE_CHOICES)
    participant_type = random.choice(ENCOUNTER_PARTICIPANT_TYPE_CHOICES)
    if diagnosis_type.startswith("zufällig"):
        use_type = random.choice(ALL_DIAGNOSIS_ROLE_TYPES)
    else:
        selected_code = diagnosis_type.split("(")[-1].rstrip(")")
        use_type = next((entry for entry in ALL_DIAGNOSIS_ROLE_TYPES if entry["code"] == selected_code),
        None)
        if not use_type:
            use_type = random.choice(ALL_DIAGNOSIS_ROLE_TYPES)
    encounter_id = str(uuid.uuid4())
    fallart_info = FALLART_MAPPING.get(fallart, {})
    fall_class_code = fallart_info.get("class")
    if isinstance(fall_class_code, list):
        fall_class_code = random.choice(fall_class_code)
    class_coding = Coding.model_construct(
        system="http://terminology.hl7.org/CodeSystem/v3-ActCode",
        code=fall_class_code,
        display=ENCOUNTER_CLASS_CHOICES.get(fall_class_code, "unknown"))

    encounter_types = fallart_info.get("type")
    if encounter_types:
        if isinstance(encounter_types, list):
        selected_type = random.choice(encounter_types)
        else:
        selected_type = encounter_types
    else:
    random_type = random.choice(ENCOUNTER_TYPE_CHOICES)
        selected_type = random_type["code"]

    encounter_type_field = [
        CodeableConcept.model_construct(
            coding=[Coding.model_construct(
                system="http://fhir.de/CodeSystem/kontaktart-de",
                code=selected_type,
                display=next(item["display"] for item in ENCOUNTER_TYPE_CHOICES if item["code"] ==
                selected_type))])]
    extensions = []
    if "extension" in fallart_info:
        nested_extensions = []
        for ext in fallart_info["extension"].get("extension", []):
            nested_extensions.append(Extension.model_construct(
                url=ext["url"],
                valueCoding=Coding.model_construct(
                    system=ext["valueCoding"]["system"],
                    code=ext["valueCoding"]["code"])))
        extensions.append(Extension.model_construct(
            url=fallart_info["extension"]["url"],
            extension=nested_extensions))
```

Listing 5.11: Funktion für Erstellung der Encounter–Ressource

Zunächst wird zufällig einer der zulässigen Werte für das *status*–Element gewählt. Anschließend erfolgen dieselben Schritte für die Elemente *serviceType* und *participant*, deren Werte aus den vordefinierten Listen `SERVICE_TYPE_CHOICES` und `ENCOUNTER_PARTICIPANT_TYPE_CHOICES` stammen.

Für den Diagnosetyp gilt eine besondere Bedingung, da dieser bereits bei der Parametrisierung ausgewählt wird. Wenn der Wert zufällig gewählt wurde, wird ein zufälliger Eintrag aus der Liste `ALL_DIAGNOSIS_ROLE_TYPES` bestimmt. Die Zeichenkette in der Benutzeroberfläche liegt in der Form Diagnosetyp (Code) vor, sodass mit der Methode `split()` der Teil innerhalb der Klammern extrahiert wird, der dem eigentlichen Code für dieses Element entspricht (`selected_code`). Anschließend wird mit der Funktion `next()` der zugehörige Anzeigename (Display) anhand des Codes aus der Liste `ALL_DIAGNOSIS_ROLE_TYPES` ermittelt. In Abhängigkeit von der gewählten *Fallart* muss für die generierte Encounter–Ressource zudem ein entsprechender Wert für die Elemente *class*, *type* und in manchen Fällen auch eine spezifische *Extension* gesetzt werden. All dies ist im Wörterbuch `FALLART_MAPPING` vordefiniert. Zunächst wird mit `get()` die vom Benutzer gewählte Fallart in `fallart_info` geladen. Danach wird der zum ausgewählten `fallart_info` gehörige Schlüssel class ausgelesen und dessen Wert in `fall_class_code` gespeichert. Bei einigen Fallarten kann es sich um mehrere mögliche Werte handeln, sodass in diesem Fall einer davon zufällig ausgewählt wird. Anschließend wird ein Objekt `class_coding` erstellt, das ein Pydantic-Modell für das *class*–Element darstellt, in welchem der verarbeitete `fall_class_code` eingesetzt wird. Nach demselben Prinzip wird auch das *type*–Element für die gewählte Fallart verarbeitet. Wenn mehrere Werte möglich sind, wird einer davon zufällig gewählt; liegt hingegen nur ein einzelner Wert vor, so wird dieser direkt in `selected_type` übernommen. Anschließend wird ein Objekt `encounter_type_field` erzeugt, das als Pydanic-Modell für das *type*–Element fungiert und den zuvor ermittelten Wert `selected_type` enthält. Schließlich hängt auch die Verwendung von *Extension* von der gewählten Fallart ab. Zunächst wird geprüft, ob in `fallart_info` eine Extension vorhanden ist. Falls ja, werden die enthaltenen Unterspezifikationen durchlaufen und für jede mithilfe der Klasse `Extension` eine neue Erweiterung erzeugt, die sowohl ein *url*–Element als auch einen Wert vom Typ `valueCoding` besitzt. Dieser Typ verweist auf ein spezifisches Codesystem und den zugehörigen Code, sodass die semantische Bedeutung eindeutig definiert bleibt. Alle diese Subextensions werden in einer Liste `nested_extensions` gesammelt und schließlich in eine übergeordnete Extension eingebettet, die ebenfalls ein *url*–Element trägt und die zuvor erzeugten Subextensions enthält.

```
def generate_encounter(patient_ref: str, condition_ref: str, fallart: str, diagnosis_type: str) ->
Encounter:
# ... (Grunddaten)
encounter = Encounter.model_construct(
        id=encounter_id,
        meta=Meta.model_construct(
        profile=[
              'https://www.medizininformatik-initiative.de/fhir/core/modul-fall/\allowbreak
              StructureDefinition/KontaktGesundheitseinrichtung|2025.0.0'])
        identifier=[
          Identifier.model_construct(
              type=CodeableConcept.model_construct(
                 coding=[
                     Coding.model_construct(
                        system="http://terminology.hl7.org/CodeSystem/v2-0203",
                        code="VN")]),
        system="http://medizininformatik-initiative.de/fhir/NamingSystem/Aufnahmenummer/MusterKrankenhaus",
        value="F_" + str(fake.unique.random_int(min=10000, max=99999)))],
        status=status_code,
        class_fhir=class_coding,
        type=encounter_type_field,
        extension=extensions if extensions else None,
        serviceType = CodeableConcept.model_construct(
                coding=[Coding.model_construct(
                    system="http://fhir.de/CodeSystem/dkgev/Fachabteilungsschluessel",
                    code=random_service_type["code"],
                    display=random_service_type["display"])]),
        subject=Reference.model_construct(reference=patient_ref),
        participant=[
            EncounterParticipant.model_construct(
                type=[CodeableConcept.model_construct(
```

```
                    coding=[Coding.model_construct(
                        system=participant_type["system"],
                        code=participant_type["code"],
                        display=participant_type["display"]))])],
        diagnosis=[
            EncounterDiagnosis.model_construct(
                condition=[
                    Reference.model_construct(reference=condition_ref)],
                use=[
                    CodeableConcept.model_construct(
                            coding=[Coding.model_construct(
                            system=use_type["system"],
                            code=use_type["code"],
                            display=use_type["display"])])], rank=1)],
    text=Narrative.model_construct(
            status="generated",
            div="<div xmlns='http://www.w3.org/1999/xhtml'>Generated Encounter (R4)</div>"))
    return encounter
```

Listing 5.12: Fortsetzung der Funktion für Erstellung der Encounter–Ressource

Nachdem die Grunddaten für die Encounter–Ressource bestimmt wurden, wird im nächsten Schritt mit der Klasse `Encounter` die eigentliche Pydantic–Modellinstanz erstellt. Innerhalb dieser Instanz werden alle zuvor ermittelten Werte den entsprechenden Elementen der Encounter–Ressource zugewiesen. Zunächst wird eine eindeutige ID generiert und im *meta*–Element die URL des MII–Profils angegeben, um die Profilkonformität der generierten Ressource sicherzustellen. Im *identifier*–Element ist es im Rahmen des MII erforderlich, ein festgelegtes Pattern zu verwenden. Dieses Element muss stets den Code „VN“ enthalten, was für die Aufnahmenummer steht. Das Subelement `system` des *identifier*–Elements verweist zusätzlich auf die URL des Krankenhauses; in diesem Fall wird eine exemplarische URL für die Kennzeichnung eines Musterkrankenhauses angegeben. Zur Generierung der konkreten Nummer für das *identifier*–Element wird die Methode `fake.unique.random_int()` verwendet.

Das *status*–Element wird mit dem zuvor bestimmten Wert `status_code` befüllt, während für das *class*–Element das vorbereitete Objekt `class_coding` genutzt wird. Analog dazu wird das `type`–Element mit dem Objekt `encounter_type_field` gesetzt. Sofern für die gewählte Fallart Extensions definiert sind, werden diese über das Feld `extension` in die Ressource integriert.

Das *serviceType*–Element verweist auf die Fachabteilung und übernimmt den zuvor ausgewählten Wert aus `random_service_type`. Für die Verknüpfung mit dem Patienten wird das *subject*–Element verwendet, in das die ID der generierten Patient–Ressource eingefügt wird. Um darzustellen, wer an dem medizinischen Fall beteiligt ist, wird im *participant*–Element das zuvor gewählte Objekt aus `participant_type` übergeben. Schließlich enthält das *diagnosis*–Element sowohl die Referenz auf die zuvor erstellte Condition–Ressource als auch den ausgewählten und in `use_type` gespeicherten Diagnosetyp.
Innerhalb der Encounter–Ressource existiert das Element *hospitalization*, in dem als Subelemente die bei der Parametrisierung gewählten Extensions übergeben werden müssen. Darüber hinaus bestehen für die Encounter–Ressource vordefinierte Regeln, die sich auf das *period*–Element beziehen. Die Funktion `generate_encounter_resource()` erzeugt jedoch kein *hospitalization*–Element, da die Bibliothek `fhir.resources` die FHIR-Version R4 nicht unterstützt, sondern auf R5 basiert. In der Version R5 trägt das entsprechende Element innerhalb der Encounter–Ressource die Bezeichnung *admission*. Aus diesem Grund kann die Bibliothek nur die Klasse *admission* importieren. Die Verwendung dieser Benennung widerspricht allerdings der Semantik des MII–Implementierungsleitfadens, da die aktuell im Rahmen der MII eingesetzte FHIR–Version R4 ist. Würde daher in der generierten Ressource *admission*–Element anstelle von *hospitalization*–Element verwendet, wäre die Ressource nicht valide in Bezug auf den MII–Implementierungsleitfaden. Für die Ergänzung dieses Elements und den Abschluss der Ressourcenerzeugung ist die Funktion `finalize_encounter()` verantwortlich. Diese berücksichtigt zusätzlich die im MII–Profil vordefinierten Constraints für die Encounter–Ressource, indem das *period*–Element in Abhängigkeit der in `generate_encounter_resource()` gesetzten Werte für *status*– und *class*–Element erzeugt wird.

```
def finalize_encounter(encounter: Encounter, admit_source: dict, discharge_ez: dict, discharge_dritte:
dict) -> dict:
    encounter_dict = encounter.model_dump(by_alias=True, exclude_unset=True)
    encounter_dict["hospitalization"] = {
        "admitSource": {
            "coding": [{
                "system": "http://fhir.de/CodeSystem/dgkev/Aufnahmeanlass",
                "code": admit_source["code"],
                "display": admit_source["display"]}]},
        "dischargeDisposition": {
            "extension": [{
                "url": "http://fhir.de/StructureDefinition/Entlassungsgrund",
                "extension": [{
                        "url": "ErsteUndZweiteStelle",
                        "valueCoding": {
                            "code": discharge_ez["code"],
                            "system":
                            "http://fhir.de/CodeSystem/dkgev/EntlassungsgrundErsteUndZweiteStelle",
                            "display": discharge_ez["display"]}},
                    {
                        "url": "DritteStelle",
                        "valueCoding": {
                            "code": discharge_dritte["code"],
                            "system": "http://fhir.de/CodeSystem/dkgev/EntlassungsgrundDritteStelle",
                            "display": discharge_dritte["display"]}}]}]}}
```

Listing 5.13: Ergänzung des hospitalization–Elements

Im Listing 5.14 ist die Funktion `finalize_encounter()` dargestellt, die als Parameter das zuvor generierte Pydantic–Modell der Encounter–Ressource sowie alle verbleibenden bei der Parametrisierung gesetzten Werte entgegennimmt.
Zunächst wird, um Änderungen an dem generierten Pydantic–Modell der Encounter–Ressource vorzunehmen, das Modell mit der Methode `model_dump()` in ein Python–Dictionary serialisiert. Nach der Umwandlung des Ressourcengehalts in ein Dictionary wird ein Block für das *hospitalization*–Element erstellt, der die Subelemente *admitSource* und *dischargeDisposition* enthält. In diese Subelemente werden anschließend die Werte aus den übergebenen und in `admit_source`, `discharge_ez` und `discharge_dritte` gespeicherten Parameter an den vorgesehenen Stellen eingefügt.

```
def finalize_encounter(encounter: Encounter, admit_source: dict, discharge_ez: dict, discharge_dritte:
dict) -> dict:
     #...
    encounter_dict["hospitalization"] = {
        # ...   }
    status_code = encounter_dict.get("status")
    class_code = encounter_dict.get("class", {}).get("code")
    period_dict = {}
    extensions = encounter_dict.get("extension", [])

    if status_code in ["cancelled", "entered-in-error"]:
        if "period" in encounter_dict:
            del encounter_dict["period"]

    if status_code == "planned":
        if "period" in encounter_dict:
            del encounter_dict["period"]
        planned_start = datetime.now(timezone.utc).isoformat()
        planned_end = (datetime.now(timezone.utc) + timedelta(hours=1)).isoformat()
        extensions.append({
            "url": "http://hl7.org/fhir/5.0/StructureDefinition/extension-Encounter.plannedStartDate",
            "valueDateTime": planned_start })
        extensions.append({
            "url": "http://hl7.org/fhir/5.0/StructureDefinition/extension-Encounter.plannedEndDate",
            "valueDateTime": planned_end })

    elif status_code == "finished":
        now = datetime.now(timezone.utc).isoformat()
        if not period_dict.get("end"):
            period_dict["end"] = now
        if class_code == "IMP":
            if not period_dict.get("start"):
```

```
32                  period_dict["start"] = (datetime.now(timezone.utc) - timedelta(hours=1)).isoformat()
33              if period_dict.get("start") and period_dict.get("end"):
34                  encounter_dict["period"] = period_dict
35
36          else:
37              if period_dict.get("end"):
38                  encounter_dict["period"] = {"end": period_dict["end"]}
39
40      elif status_code == "in-progress":
41          if period_dict.get("start"):
42              encounter_dict["period"] = {"start": period_dict["start"]}
43          else:
44              now = datetime.now(timezone.utc).isoformat()
45              encounter_dict["period"] = {"start": now}
46
47      elif status_code == "onleave":
48          if period_dict.get("start"):
49              encounter_dict["period"] = {"start": period_dict["start"]}
50          else:
51              now = datetime.now(timezone.utc).isoformat()
52              encounter_dict["period"] = {"start": now}
53
54      elif status_code == "unknown":
55          if not period_dict.get("start"):
56              now = datetime.now(timezone.utc).isoformat()
57              period_dict["start"] = now
58          encounter_dict["period"] = {"start": period_dict["start"]}
59
60      return encounter_dict
```

Listing 5.14: Ergänzung des period–Elements

Das nächste Vorgehen dieser Funktion ist die Ergänzung des *period*–Elements in Abhängigkeit von den Werten der Elemente *class*– und *status*. Dieses Element beschreibt den zeitlichen Rahmen des gesamten medizinischen Falls. Die bereits erzeugten Werte für *class*– und *status* werden aus dem `encounter_dict` entnommen und als `status_code` bzw. `class_code` gespeichert. Außerdem wird ein `period_dict` erstellt, in dem die generierten Zeitstempel hinterlegt werden, sowie das `encounter_dict`, da bestimmte Zeitangaben über spezielle Extensions abgebildet werden. Anschließend wird mithilfe von Bedingungen und den jeweiligen Werten entschieden, ob das *period*–Element mit seinen Subelementen im finalen `encounter_dict` gespeichert oder im Gegenteil daraus entfernt wird. So gilt beispielsweise, dass ein Encounter, der mit dem Status *finished* abgeschlossen wurde, zwingend ein Enddatum besitzen muss. Handelt es sich dabei um einen stationären Aufenthalt (*class = IMP*), sind sowohl ein Beginn– als auch ein Endzeitpunkt verpflichtend anzugeben. Für geplante Kontakte ist dagegen vorgegeben, dass keine Angaben zu Start– oder Endzeitpunkten im regulären *period*–Element enthalten sein dürfen. Stattdessen wird empfohlen, hierfür die speziellen Extensions für geplanten Beginn und geplantes Ende zu verwenden. Befindet sich ein Encounter im Status *in-progress*, muss mindestens ein Startzeitpunkt angegeben werden, um den laufenden Kontakt zeitlich einordnen zu können. Gleiches gilt für Encounter mit dem Status *onleave*, da auch hier der Zeitpunkt des Beginns erfasst werden muss. Für Kontakte, die den Status *unknown* tragen, wird zumindest empfohlen, einen Startzeitpunkt zu hinterlegen, um die Ressource semantisch aussagekräftiger zu gestalten. Nach der Überprüfung dieser Bedingungen gibt die Funktion die vollständige Encounter–Ressource in Form eines Python–Dictionaries zurück.

5.4.5 Funktion für Observation–Ressource

Die letzte Generierungsfunktion ist die Funktion, die eine Observation–Ressource erstellt. Im Listing 5.15 ist die Funktion `generate_observation` dargestellt, welche die IDs der zuvor generierten Organization–, Patient– und Encounter–Ressourcen entgegennimmt sowie die bei der Parametrisierung übergebenen Werte wie `lab_type`, `value_input` und `source_code`. Bevor die Pydantic–Modellinstanz für die Observation–Ressource erzeugt wird, werden die Grunddaten bestimmt. Auf Grundlage des vor der Generierung ausgewählten Labortests, der über `lab_type` übergeben wird, werden aus dem Dictionary `CODE_MAP` sämtliche Metadaten zu diesem Test extrahiert, wie der zugehörige LOINC–Code mit Displayname, die Maßeinheit für diesen Labortest (*unit*) sowie die Grenzwerte des

normalen Referenzbereichs. Alle extrahierten Metadaten zu diesem Labortest werden in der Variable `selected` gespeichert. Der Messwert wird entweder direkt aus der Benutzereingabe (`value_input`) übernommen oder, falls kein Wert angegeben wurde, zufällig innerhalb des in den Metadaten hinterlegten Referenzintervalls generiert. Zusätzlich werden zur zeitlichen Einordnung zwei Zeitstempel gesetzt: der Messzeitpunkt (`effectiveDateTime`) entspricht dem aktuellen Zeitpunkt der Generierung, während der Ausgabestempel (`issued`) standardmäßig auf drei Tage nach diesem Zeitpunkt gesetzt wird. Für die Verwendung der im Rahmen des MII–Implementierungsleitfadens definierten Extension wird ein `effective_extension` erstellt, das eine Pydantic–Modellinstanz erzeugt und auf Grundlage des vor der Generierung ausgewählten `source_code` an den entsprechenden Stellen eingefügt wird. Außerdem wird eine Liste `reference_range` erstellt, in die je nach Typ des Labortests die passenden Werte für den normalen Referenzbereich eingetragen werden.

```
def generate_observation(patient_ref, encounter_ref, org_id, lab_type, value_input, source_code):
    selected = CODE_MAP[lab_type]
    value = float(value_input) if value_input else round(random.uniform(selected["min"], selected["max"]),
    1)
    now = datetime.now(timezone.utc)
    effective_datetime = now.isoformat(timespec="seconds")
    issued_datetime = (now + timedelta(days=3)).isoformat(timespec="seconds")
    source_code_map = {
        "Specimen collection date (observable entity)": "399445004",
        "Date sample received in laboratory (observable entity)": "281271004"}

    effective_extension = Extension.model_construct(
    url="https://www.medizininformatik-initiative.de/fhir/core/modul-labor/StructureDefinition/
    QuelleKlinischesBezugsdatum"
        valueCoding=Coding.model_construct(
            system="http://snomed.info/sct",
            code=source_code_map[source_code],
            display=source_code))

    reference_range = []
    if lab_type in ["HDL Cholesterol", "LDL Cholesterol"]:
        reference_range.append({"text": ">50" if lab_type == "HDL Cholesterol" else "<100"})
    else:
        reference_range.append(ObservationReferenceRange.model_construct(
            low=Quantity.model_construct(value=selected["min"], unit=selected["unit"]),
            high=Quantity.model_construct(value=selected["max"], unit=selected["unit"]),
            type=CodeableConcept.model_construct(
                coding=[Coding.model_construct(
                    system="http://terminology.hl7.org/CodeSystem/referencerange-meaning",
                    code="normal",
                    display="Normal Range")])))
```

Listing 5.15: Funktion für Erstellung der Observation–Ressource

Nach der Bestimmung der erforderlichen Grunddaten wird die Pydantic–Modellinstanz der Observation–Ressource erstellt (siehe Listing 5.16). Es wird eine eindeutige ID generiert sowie die URL des MII–Profils angegeben, dem die Ressource entsprechen muss. Im *identifier*–Element ist im Rahmen des MII die Verwendung eines Patterns vorgeschrieben, wobei der Wert immer `OBI` lauten muss. Dies bedeutet, dass für jede Beobachtung ein eindeutiger Identifikator angegeben werden soll. Zur Erstellung eines solchen Identifikators wird die Methode `randint()` genutzt. Im Subelement *assigner* muss zudem die Organisation angegeben werden, wobei die ID der zuvor generierten Organization–Ressource eingefügt wird. Für das *category*–Element ist die Nutzung von zwei Patterns erlaubt. Die Kategorie kann entweder über einen LOINC–Code oder über das Codesystem aus der FHIR–Basisspezifikation angegeben werden. In diesem Fall werden beide Varianten parallel genutzt. Die in `selected` gespeicherten Werte werden entsprechend im *code*–Element eingesetzt. Darüber hinaus verweisen die Elemente *subject*, *encounter* und *performer* jeweils auf die zuvor generierten Patient–, Encounter– und Organization–Ressourcen. Der vom Benutzer eingegebene oder zufällig erzeugte Messwert wird im Element *valueQuantity* übergeben. Die Observation–Ressource stellt die letzte generierte Ressource innerhalb eines Generierungszyklus dar und markiert somit den Abschluss eines vollständigen Zyklus für einen Patienten.

```
def generate_observation(patient_ref, encounter_ref, org_id, lab_type, value_input, source_code):
#... (Grunddaten)
    obs = Observation.model_construct(
        id=str(uuid.uuid4()),
        meta=Meta.model_construct(
            profile=["https://www.medizininformatik-initiative.de/fhir/core/modul-labor/StructureDefinition/
            ObservationLab"]),
        text=Narrative.model_construct(
            status="generated",
            div=f"<div xmlns='http://www.w3.org/1999/xhtml'>Observation: {lab_type} = {value} {selected['
            unit']}</div>"),
        identifier=[Identifier.model_construct(
            type=CodeableConcept.model_construct(
                coding=[Coding.model_construct(
                    system="http://terminology.hl7.org/CodeSystem/v2-0203",
                    code="OBI")]),
            system="https://exmaple.org/fhir/sid/test-lab-results",
            value=f'{selected["code"]}_{random.randint(1000000000,9999999999)}',
            assigner=Reference.model_construct(
                identifier={
                    "system": "https://www.medizininformatik-initiative.de/fhir/core/CodeSystem/core-
                    location-identifier",
                    "value": org_id}))],
        status="final",
        category=[CodeableConcept.model_construct(
            coding=[
                Coding.model_construct(code="26436-6", system="http://loinc.org", display="Laboratory
                studies (set)"),
                Coding.model_construct(code="laboratory", system="http://terminology.hl7.org/CodeSystem/
                observation-category", display="Laboratory")])],
        code=CodeableConcept.model_construct(
            coding=[Coding.model_construct(
                code=selected["code"],
                system="http://loinc.org",
                display=selected["display"])],
            text=lab_type),
        subject=Reference.model_construct(reference=patient_ref),
        encounter=Reference.model_construct(reference=encounter_ref),
        effectiveDateTime=effective_datetime,
        effectiveDateTime__ext={"extension": [effective_extension]},
        issued=issued_datetime,
        performer=[Reference.model_construct(reference=f"Organization/{org_id}")],
        valueQuantity=Quantity.model_construct(
            value=value,
            unit=selected["unit"],
            system="http://unitsofmeasure.org",
            code=selected["unit"]),
        referenceRange=reference_range)
    return obs
```

Listing 5.16: Fortsetzung der Funktion für Erstellung der Observation–Ressource

5.5 Ergebnisse der Implementierung und Validierungsprozess

Vor der Nutzung der erstellten Prototypanwendung sind einige Schritte erforderlich. Der erste Schritt besteht in der Einrichtung des HAPI–FHIR–Servers, wofür die Installation von Docker und anschließend die Erstellung eines entsprechenden Docker–Containers notwendig ist. Zu diesem Zweck werden die folgenden Befehle ausgeführt:

```
docker pull hapiproject/hapi:latest
docker run -p 8080:8080 hapiproject/hapi:latest
```

Der erste Befehl lädt das Docker–Image des HAPI FHIR Servers herunter. Anschließend wird der Befehl ausgeführt, der den Docker–Container startet und mit dem Flag `-p` den Netzwerkport des Computers mit dem Port des Containers verbindet. Dadurch wird der Zugriff auf den Server über die URL

http://localhost:8080/ ermöglicht. Ohne diesen Schritt gibt die Anwendung eine Fehlermeldung aus, da die generierten Ressourcen nicht zur Speicherung an den Server übermittelt werden können.

Nach dem erfolgreichen Start des Docker–Containers wird das Modul der Anwendung `app.py` mit folgendem Befehl gestartet, wodurch die Benutzeroberfläche geöffnet wird:

```
streamlit run app.py
```

Nach der Generierung werden die Ressourcen auf dem HAPI–FHIR–Server im JSON–Format gespeichert. Zu ihrer Überprüfung ist es erforderlich, diese zunächst vom Server herunterzuladen und auf den lokalen Rechner zu übertragen. Für diesen Prozess wird cURL verwendet, das HTTP–Anfragen über die Kommandozeile ermöglicht. So wird der folgende Befehl aufgerufen:

```
curl -X GET
"http://localhost:8080/fhir/<Name_der_Ressource>/<ressource_id>"
-H "Accept:application/fhir+json"
-o ressource.json
```

Mit dem Flag `-X` wird die `GET`–Methode aufgerufen, die die generierte Ressource zurückliefert, welche sich unter einer bestimmten URL auf dem HAPI FHIR Server befindet. In die URL müssen der Ressourcentyp und die entsprechende ID eingefügt werden. Das Flag `-H` steht für den Anfrage–Header und teilt dem HAPI– FHIR–Server mit, dass die Ressource im JSON–Format erwartet wird. Das Flag `-o` dient der Speicherung und legt den Dateinamen fest, unter dem die zurückgelieferte Ressource gespeichert werden soll. Nachdem die zu prüfende JSON–Datei lokal gespeichert wurde, kann der Validierungsbefehl aufgerufen werden:

```
java -jar validator_cli.jar <example.json> -ig <Paketname>
```

Zuerst wird die Validator–Datei angegeben, dann der Name der zu prüfenden JSON–Datei, und am Ende wird über das Flag `-ig` der Name des Implementierungsleitfaden–Pakets eingefügt, das alle Anforderungen bezüglich des zu prüfenden Profils enthält.
Zur Demonstration der Korrektheit und der Übereinstimmung der generierten Ressourcen mit den Profilen der MII wird eine exemplarische Generierung sowie der anschließende Validierungsprozess durchgeführt. Auf diese Weise wird ein Patient generiert, der 27 Jahre alt und männlich ist (siehe Abbildung 5.2). Dieser Patient durchläuft eine Untersuchung oder Behandlung auf ärztliche Anweisung, was in der Encounter–Ressource dargestellt wird. Zudem unterzieht sich der Patient einem Test auf Hämoglobin, dessen Laborergebnis einen Wert von 4,8 % aufweist. Alle diese Labordaten des Patienten werden in der generierten Observation–Ressource zusammengeführt.

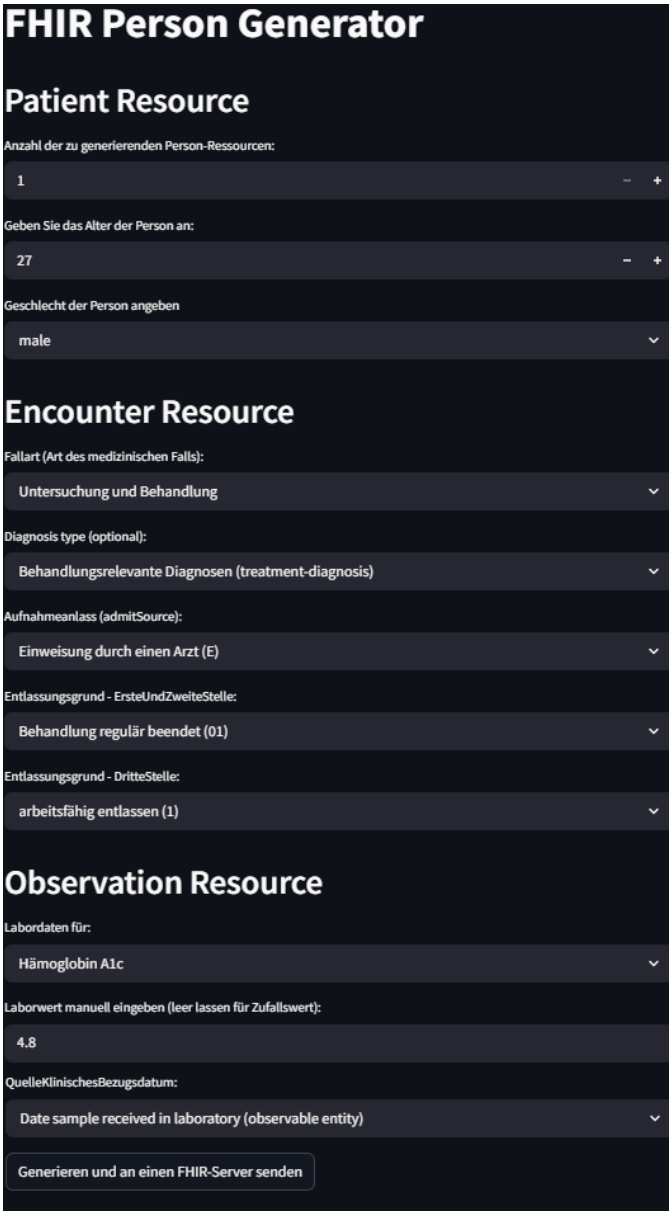


Abbildung 5.2: Benutzeroberfläche der Anwendung

Validierung der Patient–Ressource

Im Listing 5.17 ist ein Ausschnitt einer generierten Patient–Ressource dargestellt. Die anschließende Validierung mit dem offiziellen MII-Profil zeigt, dass die Ressource den Spezifikationen entspricht (siehe Listing 5.18). Das Validierungsergebnis beinhalten keine Fehler und lediglich eine Information (`note`), die bestätigt, dass alle Elemente gemäß den Regeln verarbeitet und gültig sind.

```
1 {
2   "resourceType": "Patient",
3   "id": "d4116d94-cf0b-4b96-9066-4e4dfea04d53",
4   "meta": {
```

```
    "profile": [
    "https://www.medizininformatik-initiative.de/fhir/core/modul-person/StructureDefinition/Patient|2025.0.0"
    ]},
#...
  "name": [{
    "use": "official",
    "family": "Bolander",
    "given": [ "Alex" ]}],
  "gender": "male",
  "birthDate": "1998-03-02"
#...
}
```

Listing 5.17: Beispiel einer generierten Patient–Ressource

```
C:\Users\User>curl -X GET "http://localhost:8080/fhir/Patient/d4116d94-cf0b-4b96-9066-4e4dfea04d53" ^
  -H "Accept:application/fhir+json" -o patient.json
#...

C:\Users\User>java -jar validator_cli.jar patient.json -ig
de.medizininformatikinitiative.kerndatensatz.person#2025.0.0
#...
Validate patient.json
  Validate Patient against http://hl7.org/fhir/StructureDefinition/Patient
  Validate Patient against
  https://www.medizininformatik-initiative.de/fhir/core/modul-person/StructureDefinition/Patient|2025.0.0
#...
Success: 0 errors, 0 warnings, 1 notes
Information: All OK
```

Listing 5.18: Speicherung und Validierung der generierten Patient–Ressource

Validierung der Encounter–Ressource

In Listing 5.19 ist eine generierte Encounter–Ressource abgebildet. Die Validierung gegen das MII–Profil der Encounter–Ressource bestätigt die strukturelle Korrektheit der Ressource (siehe im Listing 5.20). Neben der Erfolgsmeldung wird eine *Information* bezüglich des *status*–Elements ausgegeben. Der Grund hierfür ist, dass das CodeSystem in der Spezifikation als *draft* gekennzeichnet ist, da die darin enthaltenen Werte regelmäßig ergänzt oder auch entfernt werden können. Aus diesem Grund zeigt der Validator diese Meldung an. Da jedoch keine Fehlermeldung ausgegeben wurde, bedeutet dies, dass im *status*–Element ein gültiger Wert verwendet wurde und die Informationsmeldung daher ignoriert werden kann.

```
{
  "resourceType": "Encounter",
  "id": "8aa3897a-259b-4e8d-8ab1-485a51e5294d",
  "meta": {
    "profile": [
    "https://www.medizininformatik-initiative.de/fhir/core/modul-fall/StructureDefinition/..." ]},
  "type": [ {
    "coding": [ {
      "system": "http://fhir.de/CodeSystem/kontaktart-de",
      "code": "ub",
      "display": "Untersuchung und Behandlung"}]}],
  "status": "finished",
  #...
  "diagnosis": [ {
    "use": {
      "coding": [ {
        "system": "http://terminology.hl7.org/CodeSystem/diagnosis-role",
        "code": "treatment-diagnosis",
        "display": "Behandlungsrelevante Diagnosen"}]}}],
  #...
  "hospitalization": {
    "admitSource": {
      "coding": [ {
        "system": "http://fhir.de/CodeSystem/dgkev/Aufnahmeanlass",
        "code": "E",
```

```
        "display": "Einweisung durch einen Arzt"}]},
    "dischargeDisposition": {
      "extension": [ {
        "url": "http://fhir.de/StructureDefinition/Entlassungsgrund",
        "extension": [ {
          "url": "ErsteUndZweiteStelle",
          "valueCoding": {
            "system": "http://fhir.de/CodeSystem/dkgev/EntlassungsgrundErsteUndZweiteStelle",
            "code": "01",
            "display": "Behandlung regulär beendet"}},
            {
          "url": "DritteStelle",
          "valueCoding": {
            "system": "http://fhir.de/CodeSystem/dkgev/EntlassungsgrundDritteStelle",
            "code": "1",
            "display": "arbeitsfähig entlassen"
          }}]}]}}}
```

Listing 5.19: Beispiel einer generierten Encounter–Ressource

```
C:\Users\User>curl -X GET "http://localhost:8080/fhir/Encounter/8aa3897a-259b-4e8d-8ab1-485a51e5294d" ^
  -H "Accept:application/fhir+json" -o encounter.json
# ...

C:\Users\User>java -jar validator_cli.jar encounter.json -ig de.medizininformatikinitiative.kerndatensatz.
-fall#2025.0.0
# ...
Validate encounter.json
  Validate Encounter against http://hl7.org/fhir/StructureDefinition/Encounter
  Validate Encounter against https://www.medizininformatik-initiative.de/fhir/core/modul-fall/
  -StructureDefinition/KontaktGesundheitseinrichtung|2025.0.0
# ...
Success: 0 errors, 0 warnings, 1 notes
Information @Encounter.status (line 23, col23): Reference to draft CodeSystem http://hl7.org/fhir/encounter-
-status|4.0.1
```

Listing 5.20: Speicherung und Validierung der generierten Encounter–Ressource

Validierung der Observation–Ressource

Listing 5.21 zeigt eine generierte Observation–Ressource, die die Laborwerte des Patienten enthält. Die Validierung mit dem MII-Profil der Observation–Ressource verläuft erfolgreich (siehe im Listing 5.22). Auch hier werden keine Fehler oder Warnungen ausgegeben; die Ressource entspricht vollständig den Vorgaben des gewählten Profils.

```
{
  "resourceType": "Observation",
  "id": "a109751b-0081-403e-92b1-c6efc77b8452",
  "meta": {
    "profile": [ "https://www.medizininformatik-initiative.de/fhir/core/modul-labor/StructureDefinition/
    -ObservationLab" ]
  },
  #...
  "code": {
    "coding": [ {
      "system": "http://loinc.org",
      "code": "17856-6",
      "display": "Hemoglobin A1c/Hemoglobin.total in Blood by HPLC"
    } ],
    "text": "Hämoglobin A1c"
  },
  #...
  "effectiveDateTime": "2025-08-31T20:01:47+00:00",
  "_effectiveDateTime": {
    "extension": [ {
      "url": "https://www.medizininformatik-initiative.de/fhir/core/modul-labor/StructureDefinition/
      -QuelleKlinischesBezugsdatum",
      "valueCoding": {
        "system": "http://snomed.info/sct",
        "code": "399445004",
```

```
        "display": "Specimen collection date (observable entity)"
      }
    } ]
  },
  #...
  "valueQuantity": {
    "value": 4.8,
    "unit": "%",
    "system": "http://unitsofmeasure.org",
    "code": "%"
  }
  #...
}
```

Listing 5.21: Beispiel einer generierten Observation–Ressource

```
C:\Users\User>curl -X GET "http://localhost:8080/fhir/Observation/a109751b-0081-403e-92b1-c6efc77b8452" -H
-"Accept:application/fhir+json" -o observation.json
# ...

C:\Users\User>java -jar validator_cli.jar observation.json -ig de.medizininformatikinitiative.kerndatensatz.
-laborbefund#2025.0.2
# ...
Validate observation.json
  Validate Observation against http://hl7.org/fhir/StructureDefinition/Observation
  Validate Observation against https://www.medizininformatik-initiative.de/fhir/core/modul-labor/
  -StructureDefinition/ObservationLab|2025.0.2
# ...
Success: 0 errors, 0 warnings, 1 notes
Information: All OK
```

Listing 5.22: Speicherung und Validierung einer Observation–Ressource

Die Validierung jeder generierten Ressource verläuft erfolgreich und fehlerfrei im Hinblick auf das jeweilige Modulpaket des MII–Implementierungsleitfadens. Dies bedeutet, dass die durch diese Anwendung erzeugten Ressourcen vollständig den Anforderungen und Einschränkungen ihrer Profile entsprechen, wie sie im Rahmen des MII–Implementierungsleitfadens definiert sind.

6 Zusammenfassung und Ausblick

Die vorliegende Bachelorarbeit befasste sich mit der prototypischen Entwicklung einer Anwendung zur *Constraint*–getriebenen Generierung von medizinischen Testdaten auf Basis des HL7–FHIR–Standards und im Kontext des MII–Implementierungsleitfaden. Im Verlauf der Arbeit wurden theoretische Grundlagen erarbeitet, bestehende Generatoren analysiert und eine eigene Implementierung konzipiert und umgesetzt. Das vorliegende Kapitel fasst die zentralen Resultate zusammen und gibt einen Überblick über potenzielle Erweiterungen sowie zukünftige Forschungsarbeiten.

6.1 Zusammenfassung der Arbeit

In dieser Arbeit lag der Schwerpunkt auf der Untersuchung, wie FHIR–konforme Testdaten generiert werden können und welche strukturellen und semantischen Anforderungen dabei erfüllt sein müssen, damit die erzeugten Daten den im Rahmen des MII–Kerndatensatz definierten Profilen entsprechen.
Zur Beantwortung der ersten Forschungsfrage werden in Kapitel 3 bestehende Werkzeuge zur Generierung synthetischer medizinischer Daten untersucht. Dieses Kapitel ermöglicht eine detaillierte Analyse der Funktionsweise bereits vorhandener Generatoren und liefert die Grundlage für die Entwicklung einer eigenen Methodik zur Generierung von FHIR–konformen Daten. In Kapitel 4 wird anschließend die vorgeschlagene Methodik erläutert, bei der Ressourcen nach einer bestimmten logischen Abfolge aus vordefinierten Vorlagen erzeugt werden, innerhalb derer die notwendigen Transformationen stattfinden.
Die zweite Forschungsfrage spezifiziert das Ziel der Arbeit weiter, indem sie die Generierung durch verschiedene Regeln einschränkt, wie etwa die verpflichtende Verwendung von Extensions, die im Rahmen des MII–Implementierungsleitfadens vorgegeben sind, oder Abhängigkeiten zwischen einzelnen Elementen einer Ressource. Zur Sicherstellung der semantischen Korrektheit werden vorab definierte und zulässige Werte für die Ressourcenelemente verwendet. In Kapitel 5 werden für jede Vorlage, in der die Struktur der Ressource und die dazugehörigen Regeln implementiert sind, auch alle obligatorischen Anforderungen beschrieben und erläutert, die für die Konformität mit den MII–Profilen notwendig sind.
Zur Automatisierung und Vereinfachung der Generierung wurde in dieser Arbeit eine prototypische Anwendung entwickelt, die es im Unterschied zu bestehenden Generatoren ermöglicht, gezielt mehr Parameter für jede Ressource auszuwählen und damit unterschiedliche medizinische Szenarien realistisch abzubilden. Zur Demonstration der Korrektheit und der Einhaltung der Vorgaben des MII–Implementierungsleitfadens wird zudem ein Validierungsprozess durchgeführt. Die mit Hilfe der prototypischen Anwendung generierten Ressourcen bestehen die Validierung erfolgreich, ohne Fehler– oder Warnmeldungen zu erzeugen, und belegen damit ihre Eignung für den Einsatz in verschiedenen Test– und Forschungsszenarien im Rahmen der MII.

6.2 Ausblick der Arbeit

Aufbauend auf den in dieser Arbeit erzielten Ergebnissen lassen sich verschiedene Ansatzpunkte für zukünftige Weiterentwicklungen identifizieren:

- **Integration weiterer Ressourcentypen:** Ein erster möglicher Schritt besteht darin, zusätzliche FHIR–Ressourcen in den Generator zu integrieren. Durch die Erstellung von Vorlagen könnte eine breitere medizinische Informationsbasis abgebildet werden, sodass der Generator für vielfältigere Szenarien in Forschung und Praxis eingesetzt werden kann.

- **Erweiterung der Parametrisierungsmöglichkeiten:** Eine feinere Steuerung der Parameter für die einzelnen Ressourcen würde es erlauben, spezifischere Testfälle zu generieren und somit den praktischen Nutzen der Anwendung weiter zu erhöhen.
- **Generierung mehrerer Werte pro Element:** Derzeit wird für jedes Ressourcenelement genau ein Wert erzeugt, obwohl die FHIR-Spezifikation häufig die Verwendung mehrerer Werte vorsieht (z. B. mehrere Identifier oder Kontaktmöglichkeiten). Eine entsprechende Erweiterung würde die erzeugten Ressourcen realistischer und praxisnäher machen.
- **Unterstützung zusätzlicher Ausgabeformate:** Neben JSON könnten die Ressourcen auch in alternativen Formaten wie XML oder RDF ausgegeben werden. Dies würde die Flexibilität der Anwendung erhöhen und eine breitere Kompatibilität mit bestehenden Systemen ermöglichen.
- **Integration eines Validierungsmechanismus:** Schließlich wäre es denkbar, einen Validator direkt in die Anwendung zu integrieren. Dadurch könnten die generierten Ressourcen unmittelbar nach ihrer Erstellung überprüft und eventuelle Abweichungen von den Profilen sofort sichtbar gemacht werden, ohne den Umweg über externe Werkzeuge gehen zu müssen.

Durch die genannten Erweiterungen könnte der Generator noch breiter eingesetzt, besser an die Anforderungen der MII angepasst und zugleich benutzerfreundlicher gestaltet werden.

Literaturverzeichnis

[Act12] Acton, Q. Ashton: *Health and Medical Informatics*. In: *Issues in Information Science: Informatics*, Seite 115. ScholarlyEditions, Atlanta, 2012.

[Amo24] Amos Bordowitz: *FHIR Resources: Everything You Need to Know*. https://outburn.health/fhir-resources-everything-you-need-to-know/, 2024. Zugriff am 30.06.2025.

[ATW+24] Albashiti, F., R. Thasler, T. Wendt et al.: *Die Datenintegrationszentren – Von der Konzeption in der Medizininformatik-Initiative zur lokalen Umsetzung in einem Netzwerk Universitätsmedizin*. Bundesgesundheitsblatt, 67:630–633, 2024.

[AvdHW+22] Antwerpes, Frank, Natascha van den Höfel, Fiona Walter, Bijan Fink und Oliver Labrenz: *ICD-11*, 2022.

[Aye18] Ayegbayo, Folasayo: *Logical Observation Identifier Names and Codes (LOINC). A Modern Approach to Disease Classification and Clinical Coding*. Healthcare Coding and Technology Institute, Lagos, 2. Auflage, 2018. New Improved Edition.

[Bar25] Barney, Nick: *Clinical Document Architecture (CDA)*. https://www.techtarget.com/searchhealthit/definition/Clinical-Document-Architecture-CDA, 2025. Zugriff am 22.06.2025.

[BG21] Benson, T. und G. Grieve: *Principles of Health Interoperability: SNOMED CT, HL7 and FHIR*. Springer Nature, 4. Auflage, 2021.

[Bii23] Biiovska, Iryna: *HL7 FHIR Data Model Explained: Resources and Tools*. https://kodjin.com/blog/introduction-to-fhir-data-model/, 2023. Zugriff am 03.07.2025.

[Bra22] Braunstein, Mark L.: *Health Information Exchange: HL7 Messaging*. In: *Health Informatics on FHIR: How HL7's API is Transforming Healthcare*, Kapitel 5.13, Seite 133. Springer Nature Switzerland AG, Cham, Switzerland, 2022.

[Bun25a] Bundesinstitut für Arzneimittel und Medizinprodukte (BfArM): *Deutsche Übersetzung von LOINC*. https://www.bfarm.de/DE/Kodiersysteme/Terminologien/LOINC-UCUM/LOINC-und-RELMA/Deutsche-Uebersetzung-LOINC/_node.html, 2025. Zugriff am 20.06.2025.

[Bun25b] Bundesinstitut für Arzneimittel und Medizinprodukte (BfArM): *ICD 10 GM – International Statistical Classification of Diseases and Related Health Problems, 10th Revision, German Modification*. https://www.bfarm.de/EN/Code-systems/Classifications/ICD/ICD-10-GM/_node.html, zuletzt aufgerufen am 09.06.2025, 2025.

[Bun25c] Bundesministerium für Gesundheit: *Krebs-Forschungsdatenzentrum - KI-gestützte Evidenzgenerierung aus versorgungsnahen Daten klinischer Krebsregister, GKV-Routinedaten, Klinikdaten und deren Linkage (onkoFDZ).* https://www.bundesgesundheitsministerium.de/ministerium/ressortforschung/handlungsfelder/forschungsschwerpunkte/krebsregisterdaten/onkofdz.html, zuletzt aufgerufen am 06.06.2025, 2025.

[DAB+01] Dolin, Robert H., Liora Alschuler, Calvin Beebe, Patrick Biron, Alan Boyer, David Essin, Eric Kimber, Trish Lincoln und John E. Mattison: *The HL7 Clinical Document Architecture.* Journal of the American Medical Informatics Association, 8(6):552–569, 2001.

[DZI04] Duplaga, Marek, Krzysztof Zieliński und David Ingram. In: *Transformation of Healthcare with Information Technologies*, Kapitel 2.2, Seite 330. Springer, Berlin, Heidelberg, 2004.

[Eur16] European Parliament: *Verordnung (EU) 2016/679 des Europäischen Parlaments und des Rates vom 27. April 2016.* https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX%3A32016R0679, zuletzt aufgerufen am 09.06.2025, 2016. 119, S. 10, 38–39.

[For25] Forschen für Gesundheit: *Daten und Proben analysieren.* https://forschen-fuer-gesundheit.de/daten-und-bioproben/daten-und-proben-analysieren/#:~:text=Bioproben%20erhalten%20und%20analysieren,k%C3%B6nnen%20Sie%20beim%20Forschungsdatenportal%20erfragen., zuletzt aufgerufen am 06.06.2025, 2025.

[Ger23] German Biobank Node: *BBMRI FHIR Implementation Guide.* https://samply.github.io/bbmri-fhir-ig/index.html, 2023. Zugriff am 09.07.2025.

[GGS23] Gonzales, Aldren, Guruprabha Guruswamy und Scott Smith: *Synthetic data in health care: A narrative review.* PLOS Digital Health, 2(1):1–2, 2023.

[Gup24] Gupta, Lokesh: *HTTP Status Codes.* https://restfulapi.net/http-status-codes/, 2024. Zugriff am 01.09.2025.

[Hec21] Heckmann, Simone: *Was ist eigentlich ein „ImplementationGuide“?* https://magazin.hl7.de/was-ist-eigentlich-ein-implementationguide/, 2021. Zugriff am 07.07.2025.

[Hec22] Heckmann, Simone: *Einführung in HL7 FHIR.* https://www.devdays.com/wp-content/uploads/2022/08/DD22_220606_HeckmannSimone_Einfuhrung-Simone-Heckmann.pdf, 2022. Seiten 22–24, Zugriff am 05.07.2025.

[Hei03] Heitmann, K.: *Clinical Document Architecture.* In: *Advanced Health Telematics and Telemedicine: The Magdeburg Expert Summit Textbook*, Band 96 der Reihe *Studies in Health Technology and Informatics*, Seite 280. IOS Press, Amsterdam, The Netherlands, 2003.

[HL7a] HL7 Deutschland e.V.: *HL7 V3 RIM – Das Referenz-Informationsmodell.* https://hl7.de/themen/hl7-v3-rim-das-referenzinformationsmodell/, zuletzt aufgerufen am 04.06.2025.

[HL7b] HL7 Deutschland e.V.: *Warum FHIR?* https://hl7.de/themen/hl7-fhir-mobile-kommunikation-und-mehr/warum-fhir/. Zugriff am 22.06.2025.

[HL719] HL7 International: *FHIR R4 - Narrative*. https://hl7.org/fhir/R4/narrative.html, 2019. Zugriff am 30.06.2025.

[HL725] HL7 Deutschland e.V.: *HL7 Deutschland – Über uns*. https://hl7.de/ueber-hl7/hl7-deutschland/, 2025. Zugriff am 30.05.2025.

[Hol14] Holmes, B. C.: *The HL7 Games: Catching FHIR – Healthcare Standards after v3*. https://www.slideshare.net/slideshow/whitepaper-catching-fhir3web-final/47914691, zuletzt aufgerufen am 04.06.2025, 2014.

[Hov10] Hovenga, Evelyn J. S.: *National Standards in Health Informatics*. In: *Health Informatics: An Overview*, Band 151 der Reihe *Studies in Health Technology and Informatics*, Seite 149. IOS Press, Amsterdam, The Netherlands, 2010.

[Hua19] Huang, H. K.: *PACS-Based Multimedia Imaging Informatics: Basic Principles and Applications*. Seite 127. Springer, Hoboken, USA, 3. Auflage, 2019.

[HWS+23] Hall, Dylan, Jason Walanoski, Lee Suprenant, Alan Viars, Marc Hadley und Geoff Low: *Synthea Patient Generator*. https://github.com/synthetichealth/synthea/blob/master/README.md, 2023. Zugriff am 13.05.2025.

[Joh21] Johner, Christian: *FHIR: In drei Schritten zum eigenen Profil*. https://www.johner-institut.de/blog/iec-62304-medizinische-software/fhir-profile/#section_scroll1, 2021. Zugriff am 04.07.2025.

[Jun25] Jungmann, Sven: *19 Kernbotschaften von dem Autor von „Wie gesund wollen wir sein“*. https://svenjungmann.medium.com/19-kernbotschaften-von-dem-autor-von-wie-gesund-wollen-wir-sein-9f75163cd51b, zuletzt aufgerufen am 06.06.2025, 2025.

[KBV24] KBV – Kassenärztliche Bundesvereinigung: *Validation von FHIR-Ressourcen*. https://hub.kbv.de/pages/viewpage.action?pageId=273318219, 2024. Zugriff am 06.08.2025.

[KD23] Kiel, Alexander und Noemi Deppenwiese: *BBMRI FHIR Test Data Generator*. https://github.com/samply/bbmri-fhir-gen, 2023. Zugriff am 09.07.2025.

[KF16] Krämer, Alexander und Florian Fischer: *Semantik-Standards*. In: *eHealth in Deutschland: Anforderungen und Potenziale innovativer Versorgungsstrukturen*, Seite 42. Springer Berlin, Heidelberg, 2016.

[Khi21] Khilnani, Reshma: *Getting Started with Synthetic FHIR Data*. https://www.medplum.com/blog/2021/12/13/synthea, 2021. Zugriff am 10.07.2025.

[KSF+24] Katalinic, Mika, Martin Schenk, Stefan Franke, Alexander Katalinic, Thomas Neumuth, Andreas Dietz, Matthaeus Stoehr und Jan Gaebel: *Generation of a Realistic Synthetic Laryngeal Cancer Cohort for AI Applications*. Cancers, 16(3):639, 2024.

[KvJBH+20] Klingler, Corinna, Magdaléna von Jagwitz-Biegnitz, Mara Lena Hartung, Michael Hummel und Cornelia Specht: *Evaluating the German Biobank Node as Coordinating Institution of the German Biobank Alliance: Engaging with Stakeholders via Survey Research*. Biopreservation and Biobanking, 18(2):64–75, 2020.

[Med24] Meditecs: *HL7 Standards Overview*. https://www.meditecs.com/kb/hl7-standards/, 2024. Zugriff am 30.06.2025.

[Med25a] Medizininformatik-Initiative (MII): *Der Kerndatensatz der Medizininformatik-Initiative*. https://www.medizininformatik-initiative.de/de/der-kerndatensatz-der-medizininformatik-initiative, 2025. Zugriff am 14.08.2025.

[Med25b] Medizininformatik-Initiative (MII): *Terminologien*. https://www.medizininformatik-initiative.de/Kerndatensatz/KDS_Laborbefund_V2025/MIIIGModulLaborbefund-TechnischeImplementierung-Terminologien.html, 2025. Zugriff am 30.06.2025.

[Mer17] Meredith, James: *Why did HL7 Version 3 fail?* https://www.archetextur.es/why-did-hl7-version-3-fail/, zuletzt aufgerufen am 04.06.2025, 2017.

[Mic24] Microsoft Corporation: *FHIR profile: the basics*. https://learn.microsoft.com/en-us/azure/healthcare-apis/fhir/store-profiles-in-fhir, 2024. Zugriff am 02.07.2025.

[Mil19] Miliard, Mike: *Cerner touts adoption of normative FHIR R4 standard*. https://www.healthcareitnews.com/news/cerner-touts-adoption-normative-fhir-r4-standard, 2019. Zugriff am 22.06.2025.

[myS24] myScribe GmbH: *Der HL7 FHIR Standard im Überblick: Was sind FHIR-Ressourcen?* https://www.myscribe.de/blog/der-hl7-fhir-standard-im-uberblick, 2024. Zugriff am 30.06.2025.

[Nit17] Nitor Infotech: *Demystifying FHIR Components and Resources*. https://www.nitorinfotech.com/blog/demystifying-fhir-components-and-resources/, 2017. Zugriff am 30.06.2025.

[OAKA+25] Osamika, D., B. S. Adelusi, M. T. C. Kelvin-Agwu, A. Y. Mustapha, A. Y. Forkuo und N. Ikhalea: *A Critical Review of Health Data Interoperability Standards: FHIR, HL7, and Beyond*. World Scientific News, 203:194–233, 2025.

[Off22] Office of the National Coordinator for Health Information Technology (ONC): *Synthetic Health Data Generation to Accelerate Patient-Centered Outcomes Research (PCOR)*. https://www.healthit.gov/sites/default/files/page/2022-03/20220314_Synthetic%20Data%20Final%20Report_508.pdf, 2022. Zugriff am 12.07.2025.

[PJGBF+23] Pedrera-Jiménez, M., N. García-Barrio, S. Frid et al.: *State-of-the-Art FHIR-Based Data Model and Structure Implementations: Systematic Scoping Review*. JMIR Medical Informatics, 12(1), 2023. Zugriff am 06.06.2025.

[PP09] Pavelka, Philipp und Petranka Grozdanova Petrova: *HL7 – Integration von IT-Systemen*. https://cewebs.cs.univie.ac.at/activecc/cs/index.php?m=F&t=Bakk&c=afile&CEWebS_what=WZ.AF.MIS.II.VU&CEWebS_rev=17&CEWebS_file=HL7_Pr~228~sentation.pdf, 2009. Zugriff am 22.06.2025.

[Ray24] Raylytic Software GmbH: *Ist FHIR® ein brennendes Thema? Eine Einführung zu den Vorteilen von FHIR für Gesundheitsorganisationen*. https://www.raylytic.com/ist-fhir-ein-brennendes-thema-eine-einfuhrung-zu-den-vorteilen-von-fhir-fur-gesundheitsorganisationen/, 2024. Zugriff am 30.06.2025.

[Reg25] Regenstrief Institute: *About LOINC – Origins of LOINC*. https://loinc.org/about/#:~:text=Origins%20of%20LOINC, 2025. Zugriff am 15.06.2025.

[Rha25] Rhapsody Health: *The HL7 Evolution*. https://rhapsody.health/resources/the-hl7-evolution/, zuletzt aufgerufen am 01.06.2025, 2025.

[Rin07] Ringholm GmbH: *HL7 V2.x Messaging – An Overview. Abschnitt 2.1*. https://ringholm.com/docs/04300_en.htm, zuletzt aufgerufen am 30.05.2025, 2007.

[Ryb25] Rybakov, Andrii: *Top 6 FHIR Server Options: A Comprehensive Comparison for Your Healthcare Needs*. https://spsoft.com/tech-insights/top-six-fhir-server-options/, 2025. Zugriff am 04.08.2025.

[Smi24] Smith.Care: *Blaze: A link between research and patient care – 5 questions for Alexander Kiel on the FHIR Blaze Server*. https://www.smith.care/en/2024/11/25/5-questions-for-alexander-kiel-on-blaze/, 2024. Zugriff am 13.08.2025.

[SNO21] SNOMED International: *Germany's BfArM joins SNOMED International for nationwide use of SNOMED CT*. https://www.snomed.org/news/germany%E2%80%99s-bfarm-joins-snomed-international-for-nationwide-use-of-snomed-ct, zuletzt aufgerufen am 10.06.2025, 2021.

[Sol23] Solanki, Praveen: *Evolution of Healthcare Standards: From HL7 v2 to FHIR*. https://www.linkedin.com/pulse/evolution-healthcare-standards-from-hl7-v2-fhir-praveen-solanki-eq8mf, zuletzt aufgerufen am 04.06.2025, 2023.

[VFA24] VFA: *Nutzen von Gesundheitsdaten für Forschung und Patienten*. https://www.vfa.de/de/gesundheit-versorgung/gesundheitspolitik/abcgesundheitspolitik/nutzen-von-gesundheitsdaten-schnell-erklaert, zuletzt aufgerufen am 06.06.2025, 2024.

[Was24] Waseem, Muhammad: *3 Common and 3 Less Common Attributes of FHIR's Meta Element*. https://community.intersystems.com/post/3-common-and-3-less-common-attributes-fhir%E2%80%99s-meta-element, 2024. Zugriff am 30.06.2025.

[WC21] Walanovski, Jason und Nacho Correas: *Generic Module Framework: Complete Example*. https://github.com/synthetichealth/synthea/wiki/Generic-Module-Framework%3A-Complete-Example, 2021. Zugriff am 13.07.2025.

[Wik25a] Wikipedia Contributors: *Fast Healthcare Interoperability Resources*. https://de.wikipedia.org/wiki/Fast_Healthcare_Interoperability_Resources, 2025. Zugriff am 22.06.2025.

[Wik25b] Wikipedia Contributors: *Health Level 7 (HL7)*. https://de.wikipedia.org/wiki/HL7, zuletzt aufgerufen am 30.05.2025, 2025.

[WKN+18] Walonoski, Jason, Mark Kramer, Joseph Nichols, Andre Quina, Chris Moesel, Dylan Hall, Carlton Duffett, Kudakwashe Dube, Thomas Gallagher und Scott McLachlan: *Synthea: An approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record*. Journal of the American Medical Informatics Association, 25(3):230–238, 2018.

[Wor25a] World Health Organization: *History of the development of the ICD*. https://cdn.who.int/media/docs/default-source/classification/icd/historyoficd.pdf, zuletzt aufgerufen am 09.06.2025, 2025.

[Wor25b] World Health Organization (WHO): *ICD 11: International Classification of Diseases, 11th Revision – 2022 Release.* https://icdcdn.who.int/icd11referenceguide/en/refguide.pdf, zuletzt aufgerufen am 09.06.2025, 2025. Seiten 26–28, 34–35.

[Yes23] Yesakov, Eugene: *FHIR Components & Patient Resource Types.* https://kodjin.com/blog/understanding-fhir-components-fhir-resources/, 2023. Zugriff am 30.06.2025.

A Anhang

Hier wird der vollständig generierte Inhalt der Ressourcen mit allen erzeugten Elementen dargestellt.


```
{
  "resourceType": "Patient",
  "id": "d4116d94-cf0b-4b96-9066-4e4dfea04d53",
  "meta": {
    "profile": [
    "https://www.medizininformatik-initiative.de/fhir/core/modul-person/StructureDefinition/Patient|2025.0.0"
    ]
  },
  "text": {
    "status": "generated",
    "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\">Generated person: Alex Bolander</div>"
  },
  "identifier": [ {
    "use": "official",
    "type": {
      "coding": [ {
        "system": "http://terminology.hl7.org/CodeSystem/v2-0203",
        "code": "MR",
        "display": "Medical record number"
      } ]
    },
    "system": "urn:oid:1.3.6.1.4.1.10469.2.2",
    "value": "TL03hl61",
    "assigner": {
      "identifier": {
        "system": "http://fhir.de/sid/arge-ik/iknr",
        "value": "308128514"
      },
      "display": "Pacific Medical Records"
    }
  } ],
  "active": true,
  "name": [ {
    "use": "official",
    "family": "Bolander",
    "given": [ "Alex" ]
  } ],
  "telecom": [ {
    "system": "phone",
    "value": "0397617684",
    "use": "mobile"
  }, {
    "system": "email",
    "value": "alex.bolander@hotmail.de",
    "use": "home"
  } ],
  "gender": "male",
  "birthDate": "1998-03-02",
  "deceasedBoolean": false,
  "address": [ {
    "use": "home",
    "type": "both",
    "line": [ "Hethurgasse 522" ],
    "city": "Garmisch-Partenkirchen",
    "postalCode": "23805",
    "country": "DE"
  } ],
  "maritalStatus": {
    "coding": [ {
      "system": "http://terminology.hl7.org/CodeSystem/v3-MaritalStatus",
      "code": "S",
      "display": "Never Married"
    } ]
  },
```

```
  "communication": [ {
    "language": {
      "coding": [ {
        "system": "urn:ietf:bcp:47",
        "code": "de",
        "display": "German"
      } ]
    },
    "preferred": true
  } ],
  "managingOrganization": {
    "reference": "Organization/8c761c92-b168-4932-b326-f48e99c31283",
    "display": "Ortmann AG"
  }
}
```


Listing A.1: Generierte `Patient`–Ressource


```
{
  "resourceType": "Encounter",
  "id": "8aa3897a-259b-4e8d-8ab1-485a51e5294d",
  "meta": {
    "profile": [ "https://www.medizininformatik-initiative.de/fhir/core/modul-fall/StructureDefinition/
    -KontaktGesundheitseinrichtung|2025.0.0" ]
  },
  "text": {
    "status": "generated",
    "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\">Generated Encounter (R4)</div>"
  },
  "identifier": [ {
    "type": {
      "coding": [ {
        "system": "http://terminology.hl7.org/CodeSystem/v2-0203",
        "code": "VN"
      } ]
    },
    "system": "http://medizininformatik-initiative.de/fhir/NamingSystem/Aufnahmenummer/MusterKrankenhaus",
    "value": "F_21694"
  } ],
  "status": "finished",
  "class": {
    "system": "http://terminology.hl7.org/CodeSystem/v3-ActCode",
    "code": "IMP",
    "display": "inpatient encounter"
  },
  "type": [ {
    "coding": [ {
      "system": "http://fhir.de/CodeSystem/kontaktart-de",
      "code": "ub",
      "display": "Untersuchung und Behandlung"
    } ]
  } ],
  "serviceType": {
    "coding": [ {
      "system": "http://fhir.de/CodeSystem/dkgev/Fachabteilungsschluessel",
      "code": "3500",
      "display": "Zahn- und Kieferheilkunde, Mund- und Kieferchirurgie"
    } ]
  },
  "subject": {
    "reference": "Patient/d4116d94-cf0b-4b96-9066-4e4dfea04d53"
  },
  "participant": [ {
    "type": [ {
      "coding": [ {
        "system": "http://terminology.hl7.org/CodeSystem/v3-ParticipationType",
        "code": "PPRF",
        "display": "primary performer"
      } ]
    } ]
  } ],
  "period": {
    "end": "2025-08-31T20:01:47.858330+00:00"
```

```
  },
  "diagnosis": [ {
    "condition": {
      "reference": "Condition/5448fe75-cc6a-4da7-9868-c21780c236f9"
    },
    "use": {
      "coding": [ {
        "system": "http://terminology.hl7.org/CodeSystem/diagnosis-role",
        "code": "AD",
        "display": "Admission diagnosis"
      } ]
    }
  } ],
  "hospitalization": {
    "admitSource": {
      "coding": [ {
        "system": "http://fhir.de/CodeSystem/dgkev/Aufnahmeanlass",
        "code": "E",
        "display": "Einweisung durch einen Arzt"
      } ]
    },
    "dischargeDisposition": {
      "extension": [ {
        "url": "http://fhir.de/StructureDefinition/Entlassungsgrund",
        "extension": [ {
          "url": "ErsteUndZweiteStelle",
          "valueCoding": {
            "system": "http://fhir.de/CodeSystem/dkgev/EntlassungsgrundErsteUndZweiteStelle",
            "code": "01",
            "display": "Behandlung regular beendet"
          }
        }, {
          "url": "DritteStelle",
          "valueCoding": {
            "system": "http://fhir.de/CodeSystem/dkgev/EntlassungsgrundDritteStelle",
            "code": "1",
            "display": "arbeitsfhig entlassen"
          }
        } ]
      } ]
    }
  }
}
```


Listing A.2: Generierte `Encounter`–Ressource


```
{
  "resourceType": "Observation",
  "id": "a109751b-0081-403e-92b1-c6efc77b8452",
  "meta": {
    "profile": [ "https://www.medizininformatik-initiative.de/fhir/core/modul-labor/StructureDefinition/
    -ObservationLab" ]
  },
  "text": {
    "status": "generated",
    "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\">Observation: Hmoglobin A1c = 4.8 %</div>"
  },
  "identifier": [ {
    "type": {
      "coding": [ {
        "system": "http://terminology.hl7.org/CodeSystem/v2-0203",
        "code": "OBI"
      } ]
    },
    "system": "https://exmaple.org/fhir/sid/test-lab-results",
    "value": "17856-6_4294033836",
    "assigner": {
      "identifier": {
        "system": "https://www.medizininformatik-initiative.de/fhir/core/CodeSystem/core-location-identifier
        -",
        "value": "8c761c92-b168-4932-b326-f48e99c31283"
      }
    }
```

```
  } ],
  "status": "final",
  "category": [ {
    "coding": [ {
      "system": "http://loinc.org",
      "code": "26436-6",
      "display": "Laboratory studies (set)"
    }, {
      "system": "http://terminology.hl7.org/CodeSystem/observation-category",
      "code": "laboratory",
      "display": "Laboratory"
    } ]
  } ],
  "code": {
    "coding": [ {
      "system": "http://loinc.org",
      "code": "17856-6",
      "display": "Hemoglobin A1c/Hemoglobin.total in Blood by HPLC"
    } ],
    "text": "H moglobin A1c"
  },
  "subject": {
    "reference": "Patient/d4116d94-cf0b-4b96-9066-4e4dfea04d53"
  },
  "encounter": {
    "reference": "Encounter/8aa3897a-259b-4e8d-8ab1-485a51e5294d"
  },
  "effectiveDateTime": "2025-08-31T20:01:47+00:00",
  "_effectiveDateTime": {
    "extension": [ {
      "url": "https://www.medizininformatik-initiative.de/fhir/core/modul-labor/StructureDefinition/
      -QuelleKlinischesBezugsdatum",
      "valueCoding": {
        "system": "http://snomed.info/sct",
        "code": "399445004",
        "display": "Specimen collection date (observable entity)"
      }
    } ]
  },
  "issued": "2025-09-03T20:01:47+00:00",
  "performer": [ {
    "reference": "Organization/8c761c92-b168-4932-b326-f48e99c31283"
  } ],
  "valueQuantity": {
    "value": 4.8,
    "unit": "%",
    "system": "http://unitsofmeasure.org",
    "code": "%"
  },
  "referenceRange": [ {
    "low": {
      "value": 4.0,
      "unit": "%"
    },
    "high": {
      "value": 5.6,
      "unit": "%"
    },
    "type": {
      "coding": [ {
        "system": "http://terminology.hl7.org/CodeSystem/referencerange-meaning",
        "code": "normal",
        "display": "Normal Range"
      } ]
    }
  } ]
}
```


Listing A.3: Generierte `Observation`–Ressource

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und ohne unerlaubte Hilfe Dritter verfasst habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Quellen entnommen wurden, sind als solche kenntlich gemacht und mit vollständigen Quellenangaben versehen. Es wurden ausschließlich die in der Arbeit aufgeführten Quellen verwendet.

Für sprachliche und grammatikalische Überprüfungen habe ich das KI–Tool ChatGPT–5 genutzt. Die Nutzung beschränkte sich ausschließlich auf Korrekturen in Bezug auf Sprache und Ausdruck. Bei der Erstellung des inhaltlichen Textes, bei der Entwicklung des Konzepts und Darstellung der wissenschaftlichen Ergebnisse wurde ChatGPT nicht verwendet.

Rostock, den 9. September 2025
Najaf Abdiyev