Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента	*Байдака Леоніда Васильовича*
	(ПІБ)
академічної групи	*122М-20-1*
	(шифр)
спеціальності	*122 Комп'ютерні науки*
	(код і назва спеціальності)
освітньої програми	*«122 Комп'ютерні науки»*
	(назва освітньої програми)
на тему:	*Розробка та дослідження ефективності впровадження веб-сервісу для валідації і зберігання медичних документів за стандартом FHIR v4.*

_____*Л.В. Байдак*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
розділів кваліфікаційної роботи				
спеціальний	*Проф. Бердник М.Г.*			
економічний	*Доц. Касьяненко Л.В.*			

Рецензент				

Нормоконтролер	*Доц. Реута О.В.*			

Дніпро
2022

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)
_____ І.М. Удовик
(підпис) (прізвище, ініціали)
« » _____ 20 ____ Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

студенту *122м-20-1* *Байдаку Леоніду Васильовичу*
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Розробка та дослідження ефективності впровадження веб-сервісу для валідації і зберігання медичних документів за стандартом FHIR v4.*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 10.12.2021 р. № 1036-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – процес розробки та впровадження FHIR-based сервісу для валідації, зберігання та REST взаємодії з медичними документами.

Предмет досліджень – методи обміну медичною інформацією і забезпечення інтероперабельності між медичними закладами.

Мета роботи – підвищення ефективності системи інтеграції між різними медичними закладами.

Методи дослідження – базуються на основних принципах системного аналізу, функціонального аналізу, теорії баз даних. Також були використані методи емпіричного узагальнення на основі даних.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна – запропоновано методи покращення і спрощення ведення документообігу задля підвищення ефективності системи інтеграції між різними медичними закладами.

Практична цінність полягає в тому, що результат дослідження можна використовувати в медичних закладах задля стандартизації обігу медичних документів.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати досліджень мають бути подані у вигляді, що дозволяє побачити та оцінити безпосереднє використання запропонованих методів. В результаті роботи повинен бути розроблений сервіс на основу стандарту FHIR v4.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз теми та постановка задачі.	15.09.2021-30.09.2021
Дослідження існуючих підходів до інтеграції та стандартизації медичного документообігу.	01.10.2021-05.11.2021
Створення сервісного додатку для валідації, зберігання та REST взаємодії з медичними документами за стандартом FHIR v4.	06.11.2021-20.01.22

6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект від реалізації результатів роботи очікується позитивним завдяки розробці алгоритму та програмного забезпечення для обігу і валідації медичних документів за стандартом FHIR v4.

Соціальний ефект від реалізації результатів роботи очікується позитивним, завдяки полегшенню роботи розробників та працівників в сфері охорони здоров’я.

Завдання видав _____ *Бердник М. Г.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Байдак Л.В.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *15.09.2021 р.*

Термін подання дипломного проекту до ЕК *24.01.2022*

РЕФЕРАТ

Пояснювальна записка: 85 с., 28 рис., 3 табл., 3 дод., 39 джерел.

Об'єкт дослідження: процес розробки та впровадження FHIR сервісу для валідації, зберігання та REST взаємодії з медичними документами.

Предмет дослідження: методи обміну медичною інформацією і забезпечення інтероперабельності між медичними закладами.

Мета магістерської роботи: покращення умов обміну медичною інформацією та забезпечення інтероперабельності між медичними закладами.

Методи дослідження: методи дослідження базуються на основних принципах системного аналізу, функціонального аналізу, теорії баз даних. Також були використані методи емпіричного узагальнення на основі даних.

Наукова новизна: запропоновано методи покращення і спрощення ведення документообігу задля підвищення ефективності інтеграції між різними медичними закладами та забезпечення високого рівня інтероперабельності.

Прогнози щодо розвитку досліджень: дослідження, описані в кваліфікаційні роботі, можуть стати базисом для впровадження веб-сервісу для валідації і зберігання документів в справжніх закладах охорони здоров’я.

Практична цінність полягає в тому, що результат дослідження можна використовувати в медичних закладах задля об'єднання і обміну медичною інформацією та забезпечення високого рівня інтероперабельності.

У розділі «Економіка» проведено розрахунки трудомісткості розробки програмного забезпечення, витрат на створення ПО і тривалості його розробки.

Список ключових слів: стандартизація, інтероперабельність, FHIR, HL7, REST API, клієнт-серверна архітектура, Java, MS SQL, Transact-SQL.

ABSTRACT

Explanatory note: 85 p., 28 fig., 3 tables, 3 applications, 39 sources.

Object of research: process of developing and implementing an FHIR-based service for validation, storage and REST of interaction with medical documents.

Subject of research: methods of exchanging medical information and ensuring interoperability between medical institutions.

Purpose of Master's thesis: improving the efficiency of exchanging medical information and ensuring interoperability between medical institutions.

Research methods: research methods are based on main principles of system analysis, functional analysis and the theory of databases. Also data-based empirical generalization methods were used.

Originality of research methods of improving and simplifying document management are proposed in order to increase the efficiency of integration between different medical institutions and ensure interoperability.

Forecasts for the development of research: the research described in the qualification work can be the basis for the implementation of a web service for validation and storage of documents in real health care facilities.

Practical value of the results is that the result of the study can be used in medical institutions to combine and exchange medical information and ensure interoperability.

In the Economics section the complexity, costs and duration of software development were calculated.

Keywords: standardization, interoperability, FHIR, HL7, REST API, client-server architecture, Java, MS SQL, Transact-SQL.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД –	База даних
ПЗ –	Програмне забезпечення
ПК –	Персональний комп’ютер
ОС –	Операційна система
API –	Application Programming Interface
HTML –	HyperText Markup Language
HTTP –	HyperText Transfer Protocol
FHIR –	Fast Healthcare Interoperability Resources
HL7 –	Health Level Seven
REST –	Representational State Transfer
EHR –	Electronic health record
JSON –	JavaScript Object Notation

ЗМІСТ

ВСТУП

Актуальність дослідження. Епідемія Covid-19 негативно вплинула на всі країни світу без винятку. Можна з цілковитою впевненістю сказати, що остаточні наслідки пандемії будуть вивчатися і аналізуватися ще багато років.

Слід зазначити, що більшість систем охорони здоров'я, незважаючи на масштаби негативного впливу, намагаються впровадити ефективні рішення проблеми. Але Covid-19 приніс у звичний плин життя і людської діяльності нові, непередбачувані умови карантину. Ситуація в Україні – не виключення.

Відповідно до дослідження актуальної ситуації в Україні, результати котрого наведені в «Wiadomości Lekarskie, VOLUME LXXIV»[1], під назвою «HEALTHCARE IN UKRAINE DURING THE EPIDEMIC: DIFFICULTIES, CHALLENGES AND SOLUTIONS», однією з основних проблем системи охорони здоров’я в Україні є недосконаліть програмного забезпечення та повільні темпи адаптації системи під нові вимоги. Взагалі, проблема підтримки та забезпечення відповідності вимогам бізнеса програмного забезпечення є одним з основних завдань IT-інженера. Складність сфери охорони здоров’я, моделювання даних, зберігання медичних даних та користувальницька інтеграція із застарілими системами є факторами, які стимулюють тривалі цикли розробки та високі витрати на проекти для технологій охорони здоров’я.

Однак розвиток технологій не стоїть на місці, і з’являються нові шляхи розв’язку для актуальних проблем. В даній роботі планується розробити рішення, що буде забезпечувати високий рівень інтероперабельності завдяки стандартизації API. За основу дослідження була взятий позитивний досвід країн з високою якістю систем охорони здоров’я, як то США, Великобританія, Німеччина.

Мета дослідження полягає в розробці та дослідження ефективності впровадження веб-сервісу для валідації і зберігання медичних документів за стандартом FHIR v4.

Завдання дослідження. Для досягнення поставленої мети в роботі сформульовані та вирішені такі завдання:

1. Визначити існуючі стандарти в впровадженні програмного забезпечення в сфері охорони здоров’я;
2. Визначити архітектуру додатку;
3. Розробити серверний додаток з відкритим API, котрий реалізує стандарт FHIR;
4. Провести дослідження ефективності роботи даного додатку.

Об’єкт дослідження: процес розробки та впровадження FHIR сервісу для валідації, зберігання та REST взаємодії з медичними документами.

Предмет дослідження: методи обміну медичною інформацією і забезпечення інтероперабельності між медичними закладами.

Методи дослідження: методи дослідження базуються на основних принципах системного аналізу, функціонального аналізу, теорії баз даних. Також були використані методи емпіричного узагальнення на основі даних.

Наукова новизна: запропоновано методи покращення і спрощення ведення документообігу задля підвищення ефективності інтеграції між різними медичними закладами та забезпечення високого рівня інтероперабельності.

Практична цінність полягає в тому, що результат дослідження можна використовувати в медичних закладах задля об'єднання і обміну медичною інформацією та забезпечення високого рівня інтероперабельності.

Особистий внесок автора:

1. Наукові результати роботи отримані автором самостійно;
2. Вибір методів досліджень і технологій реалізацій;
3. Імплементація серверного додатку котрий реалізує стандарт FHIR;
4. Розробка теоретичної частини роботи, в якій досліджені і систематизовані знання про стандарти медичного документообігу;
5. Оцінка отриманих результатів.

Структура і обсяг роботи. Робота складається з вступу, чотирьох розділів, висновків, списку використаних джерел і трьох додатків. Загальний

обсяг роботи становить 87 сторінок, із них 61 сторінка основного тексту, в тому числі 28 рисунків, 3 таблиці і список використаних джерел із 39 найменувань на 2 сторінках.

РОЗДІЛ 1
РОЗВИТОК МЕДИЧНИХ СТАНДАРТІВ ТА ТИПОВІ ПРОБЛЕМИ ІНТЕРОПЕРАБЕЛЬНОСТІ СИСТЕМ ОХОРОНИ ЗДОРОВ'Я

1.1. Створення систем охорони здоров'я «сьомого» рівня

В 1978 році Міжнародний комітет зі стандартизації (ISO) розробив стандарт архітектури ISO 7498, для об'єднання різних мереж. У розробці брало участь 7 комітетів, кожному з них був відведений свій рівень. В 1980 році IEEE опублікував специфікацію 802, що детально описала механізми взаємодії фізичних пристроїв на канальному й фізичному рівнях моделі OSI. В 1984 році специфікацію моделі OSI переглянули й прийняли як міжнародний стандарт для мережних комунікацій.

Хоча це було зроблено не тільки для охорони здоров'я, модель OSI стала основоположною для розуміння розробки стандартів, оскільки розділивши комунікаційну систему на рівні абстракції, це відкрило двері для стандартів охорони здоров'я, створених на верхньому, сьомому, рівні.

1.2. Розробка стандарту HL7

Організація HL7 була заснована у 1987 році для розробки стандарту обміну даними з лікарняними інформаційними системами. Назва «Рівень охорони здоров'я – 7» це посилання на сьомий, так званий «прикладний» рівень еталонної моделі ISO OSI.

Стандарт HL7 версії 2 (також відомий як Pipehat) має на меті підтримувати робочі процеси в лікарні. Спочатку він був створений у 1989 році. HL7 версія 2 визначає серію електронних повідомлень для підтримки адміністративних, матеріально-технічних, фінансових, а також клінічних процесів. Стандарти v2.x мають зворотну сумісність (наприклад, повідомлення на основі версії 2.3 буде зрозуміло програмою, яка підтримує версію 2.6).

Повідомлення HL7 v2.x використовують синтаксис кодування, відмінного від XML, заснований на сегментах (рядках) і одно символьних роздільниках.

У 2000 році з'являються стандарти XML, HTTPS, SSL та TLS, котрі стають основою для HL7 v3.

Спроектований інформаційними архітекторами, а не потенційними імплементаторами стандарту, багато хто розглядає HL7 v3 як провал. Основні тези, чому так сталось:

- Компанії не бачили сенсу вкладати час і кошти на стандарт, який вже добре працював.
- Забезпечуючи інтероперабельність, V3 не вистачало гнучкості, необхідної для фактичних операцій та впровадження.

Хоча HTTPS був обраний як транспортний механізм, розробники також додали купу додаткових обгорток (SOAP і ebXML), які зробили і без того складну ситуацію некерованою.

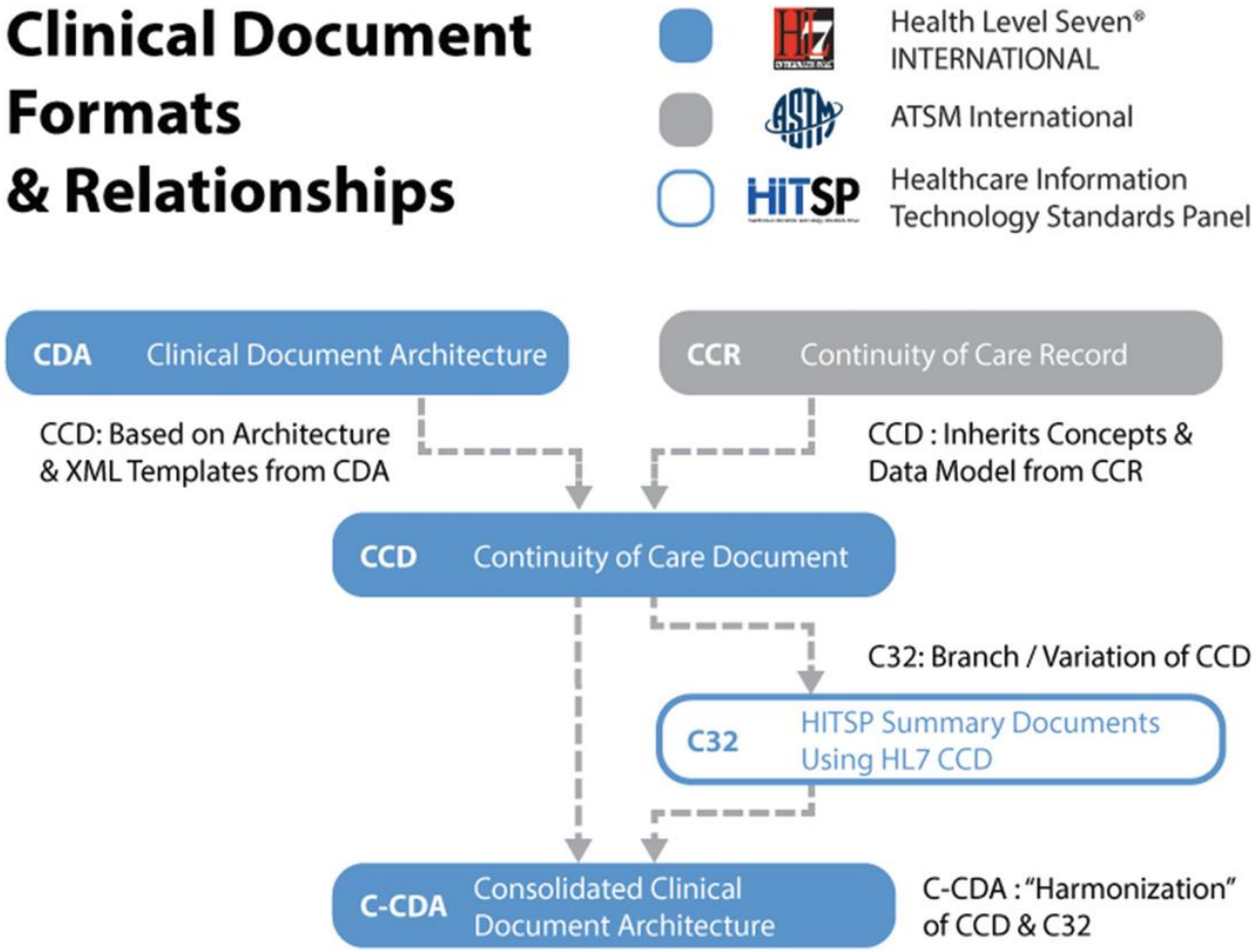


Рис. 1.1. Структура C-CDA

У листопаді 2000 року було опубліковано CDA (HL7 Clinical Document Architecture) – це стандарт розмітки на основі XML, призначений для визначення кодування, структури та семантики клінічних документів (рис 1.1).

В 2011 році було опубліковано C-CDA (HL7 Consolidated Clinical Document Architecture) – це стандарт який надає бібліотеку документів у форматі CDA. Клінічними документи, що використовують стандарти C-CDA, обмінюються мільярдами разів на рік тільки у Сполучених Штатах. Усі сертифіковані електронні медичні записи в Сполучених Штатах повинні експортувати медичні дані за стандартом C-CDA. Хоча стандарт був розроблений в першу чергу для Сполучених Штатів, він також використовується на міжнародному рівні. [2]

Також розповсюдженими стандартами є DICOM, EDIFACT, Cisco Medical Data Exchange Solution [3]. Порівняння функціональності деяких стандартів наведено в табл. 1.1

Таблиця 1.1

Порівняльна таблиця функціональності стандартів сфери охорони здоров'я

Назва функції	**DICOM**	**xDT**	**EDIFACT**	**HL7/CDA**
Інформаційна система лікарні	+	+	+	+
Інформація радіології	+	-	-	+
Система зберігання і обміну зображеннями	+	-	-	+
Головний індекс пацієнтів	-	+	-	+
Графічний діагноз	+		-	+
Архівування	+	+	-	+
Коментарі до діагнозу	-	+	-	+
Зображення в документації	+	-	-	+
Проміжні звіти	-	-	-	+
Відео в документації	-	-	-	+

Продовження таблиці 1.1

Назва функції	**DICOM**	**xDT**	**EDIFACT**	**HL7/CDA**
Реєстрація пацієнтів	-	+	-	+
Електронна історія хвороби	+	+	+	+
Створення рахунків	-	+	+	+
Рецепти ліків	-	-	-	+
Перевтілення данних	-	+	-	+
Відомості про надзвичайні випадки	-	-	-	+
Практика лікарів	-	+	+	+

1.3. Розробка стандарту FHIR

Fast Healthcare Interoperability Resources (FHIR) – це стандарт, що описує формати даних та елементи (відомі як «ресурси»), а також API для обміну електронними медичними записами. Стандарт був створений Міжнародною організацією зі стандартів охорони здоров’я 7 рівня (HL7) в 2014 році. [3]

FHIR базується на попередніх стандартах формату даних від HL7, як-от HL7 версії 2.x і HL7 версії 3.x. Але його легше реалізувати, оскільки він використовує сучасний веб-набір технологій API, включаючи REST на основі HTTP, а також JSON, XML або RDF для представлення даних. Однією з його цілей є сприяти взаємодії між застарілими системами охорони здоров’я, полегшити надання медичної інформації медичним працівникам, а також дозволити стороннім розробникам створювати медичні програми, які можна легко інтегрувати в існуючі системи. [4]

1.4. Інтероперабельність систем охорони здоров’я

Інтероперабельність охорони здоров’я описує здатність постачальників медичних послуг та різних систем обмінюватися інформацією про пацієнтів. Система EHR (Електронний медичний запис) одного постачальника повинна мати можливість передавати дані пацієнтів до системи іншого постачальника.

Щоб досягти справжньої сумісності, постачальники повинні підтримувати «ліквідність» даних. Це означає, що дані пацієнтів завжди доступні для зацікавлених сторін. [5]

Постачальник EHR, який надає інтерфейс з високим рівнем інтероперабельності, може легко стати лідером ринку.

Сумісність EHR також важлива для догляду за пацієнтами. Згідно з дослідженням, опублікованим у 2017 році в журналі «Studies in Health Technology and Informatics», медичні помилки є третьою за поширеністю причиною смерті в Сполучених Штатах Америки.

1.4.1 Типові проблеми інтероперабельності

У сфері медичного обслуговування чільне місце займають проблеми інтероперабельності. Основні з них наведені нижче:

1. Керування неузгодженою інформацією з декількох джерел. Для IT-компаній які обслуговують великі мережі охорони здоров’я, неконсистентна інформація є величезною проблемою. Провайдери розміщують різні фрагменти даних у кількох, часто розрізнених місцях, а IT-відділи охорони здоров’я витрачають незліченну кількість годин на їх пошук. Рішенням такої проблеми буде використання єдиної уніфікованої мережі та інтерфейсом.
2. Валідація електронних запитів на інформацію про пацієнтів. Без перебільшення можна сказати, що збереження конфіденційності та безпеки

медичних записів пацієнтів є одним з головних завдань системи охорони здоров’я.

3. Подолання опору обміну даними. Деякі суб’єкти в галузі охорони здоров’я зацікавлені в тому, щоб не ділитися даними з іншими постачальниками. Наприклад, лікарняні системи конкурують за пацієнтів з клініками невідкладної допомоги. Закон вимагає, щоб дані про стан здоров’я були доступними за межами організацій та для самих пацієнтів.
4. Висока вартість найму спеціалістів для управління інтероперабельністю. Досягнення інтероперабельності систем – це досить складна робота, котра потребує вузькопрофільних спеціалістів. У більшості закладів охорони здоров’я типовий співробітник не має відповідної кваліфікації.
5. Необхідна вимога – зробити данні доступними. В іншому випадку вас можуть охарактеризувати як блокувальника даних і оштрафувати. Найкращим рішенням буде експортувати дані в єдине місце в уніфікованому форматі, наприклад створити онлайн-портал. Таким чином, дані пацієнтів завжди доступні для тих, хто їх потребує.

1.5. Висновки до першого розділу

При порівнянні різних медичних стандартів та дослідженні можливостей запровадження інтероперабельності між сервісами та обігу медичних документів було виділено основні проблеми:

1. Керування неконсистентною інформацією;
2. Валідації інформації;
3. Зберігання інформації про пацієнта;
4. Надання публічного інтерфейсу для доступу до інформації.

Для вирішення цих проблем було обрано використання об’єктно-орієнтованої мови програмування Java, що має широку підтримку серед сучасних технологій для розробки сервісних додатків. Перевагами цієї мови

програмування є мультиплатформовість та величезне спільнота. Також, для побудови додатку було вирішено використовувати стек фреймворків Spring.

В якості архітектури додатку в цілому, була обрана клієнт-серверна архітектура.

Основою роботи сервісу було обрано FHIR v4, який фактично є стандартом у розробці в сфері охорони здоров’я. Це значно спростить розробку, а також забезпечить стабільність в роботі додатка в цілому, оскільки ця технологія заснована на перевірених стандартах галузі.

РОЗДІЛ 2
ТЕХНОЛОГІЯ РЕАЛІЗАЦІЇ ВЕБ-СЕРВІСА НА ОСНОВІ FHIR v4

2.1. Принцип роботи FHIR v4

У січні 2011 року правління HL7 International доручило експертній робочій групі відповісти на питання: «Якби HL7 почався з нуля сьогодні, як би виглядали стандарти інтероперабельності?» Розглядаючи це питання, робоча група зазначила, що:

- HL7 v2 була (і є) надзвичайно успішною, але технологія застаріла і погано підходить для нових реалій.
- HL7 v3, хоча і заснована на надійній моделі, не отримала широкого визнання і вважається як така, котру складно реалізувати.
- Архітектура клінічних документів (CDA) була надзвичайно успішною, але була розроблена як механізм обміну документами, і тому не підходить для всіх можливих сценаріїв.
- Існують нові варіанти використання, особливо мобільні пристрої, де поточні стандарти не підходять.
- В загальному, REST-підхід став домінуючим на ринку у царині обміну інформацією та спілкуванні між веб-сервісами.

Метою було створити стандарт, котрий:

- Його легко розробити, з невеликим вхідним періодом та мінімальними вимогами до інструментів.
- Легко реалізувати (або настільки просто, наскільки простою може бути інтероперабельність в галузі охорони здоров’я).
- Є «дружнім до впровадження» – себто використовує загально і добре відомі інструменти та технології.

- Артефакти повинні бути «human readable», тобто такими котра розуміє людина. Хоча вони не призначені для безпосереднього користування людиною, можливість їх зрозуміти буде корисною як для реалізаторів, так і для підтримуючого персоналу.
- Артефакти мають проходити валідацію в електронному форматі, без участі оператора настільки, наскільки це можливо.
- Має мати високий потенціал до інтеграції та використовувати добре знані технології комунікації (HTTP, XML, JSON тощо).

Аби досягти цього, команда FHIR створила:

- Специфікацію, розмішену в мережі та яка є повність гіпертекстовою, себто такою, де використовуються гіперпосилання.
- Легкий для читання формат всіх ресурсів, який включає визначення в вигляді «псевдо-XML», діаграми UML та посилання для формальних визначень. Формат такий, що фахівці в сфері охорони здоров'я можуть зрозуміти, що містить і що представляє ресурс (хоча цільова аудиторія все-ж таки є реалізатори стандарту).
- Кілька прикладів для кожного ресурсу, які показують, як кожен ресурс має використовуватися.
- Вільно доступні зразкові реалізації на Delphi, Java, .NET і Objective C, котрі можуть використовуватись як основа для власних розробок.
- Ряд загальнодоступних тестових серверів FHIR, таких як HAPPI, Firely та ін., які можуть використовуватися розробниками для тестування своїх програм.

2.1.1. Концепт ресурсу стандарту FHIR v4

Ресурс — це найменша одиниця обміну даних, яка «має сенс» в інтероперабельності, наприклад, Observation(спостереження), Patient(пацієнт)

або Condition(стан). Вони приблизно аналогічні сегменту в повідомленні v2 або СМЕТ у світі v3.

Намір полягає в тому, що існує невелика кількість фундаментальних ресурсів (100-150), які утворюють будівельні блоки всіх артефактів FHIR.

Ресурс має поняття «ідентичності» - щось, що ідентифікує його як логічну «одиницю» і має місце розташування (URI), де його можна знайти. URI включає як ідентифікатор, так і хост, де зберігається цей ресурс.

Ресурс складається з елементів, кожен з яких є певним типом даних (наприклад, String або CodeableConcept). Деякі елементи можуть бути представлені декількома типами даних. На рис 2.1 наведено приклад FHIR-ресурсу.


```
<Patient xmlns="http://hl7.org/fhir">
  <id value="glossy"/>
  <meta>
    <lastUpdated value="2014-11-13T11:41:00+11:00"/>
  </meta>
  <text>
    <status value="generated"/>
    <div xmlns="http://www.w3.org/1999/xhtml">
      <p>Henry Levin the 7th</p>
      <p>MRN: 123456. Male, 24-Sept 1932</p>
    </div>
  </text>
  <extension url="http://example.org/StructureDefinition/trials">
    <valueCode value="renal"/>
  </extension>
  <identifier>
    <use value="usual"/>
    <type>
      <coding>
        <system value="http://hl7.org/fhir/v2/0203"/>
        <code value="MR"/>
      </coding>
    </type>
    <system value="http://www.goodhealth.org/identifiers/mrn"/>
    <value value="123456"/>
  </identifier>
  <active value="true"/>
  <name>
    <family value="Levin"/>
    <given value="Henry"/>
    <suffix value="The 7th"/>
  </name>
  <gender value="male"/>
  <birthDate value="1932-09-24"/>
  <careProvider>
    <reference value="Organization/2"/>
    <display value="Good Health Clinic"/>
  </careProvider>
</Patient>
```

Рис. 2.1. Приклад ресурсу Patient в форматі fhir/xml

2.1.2. Ресурс і Доменний ресурс (Resource and DomainResource)

Resource – це об’єкт, який:

1. Має відомий ідентифікатор (URL), за допомогою якого до нього можна звертатися
2. Містить набір структурованих елементів даних, як описано визначенням типу ресурсу
3. Має визначену версію, яка змінюється, якщо змінюється вміст ресурсу

На рис. 2.2 наведено структуру ресурсу «Resource»

Structure

Name	Flags	Card.	Type	Description & Constraints
Resource	N		n/a	Base Resource
id	Σ	0..1	id	Logical id of this artifact
meta	Σ	0..1	Meta	Metadata about the resource
implicitRules	?! Σ	0..1	uri	A set of rules under which this content was created
language		0..1	code	Language of the resource content Common Languages (Preferred but limited to AllLanguages)

Рис 2.2. Структура ресурсу «Resource»

DomainResource – це об’єкт який наслідується від базового «Resource». Усі ресурси, крім «Bundle», «Parameters» і «Binary», розширюють цей ресурс. «DomainResource» характеризується наступними властивостями:

1. Має доступне для читання людиною XHTML-подання вмісту ресурсу.
2. Може мати всередині інші, не оголошені ресурси.
3. Може мати додаткові розширення.

Усі ресурси рівня 2 і вище успадковують усі елементи, визначені в «Resource» і «DomainResource». На рис. 2.3 наведено діаграму ієрархії наслідування ресурсів в FHIR.

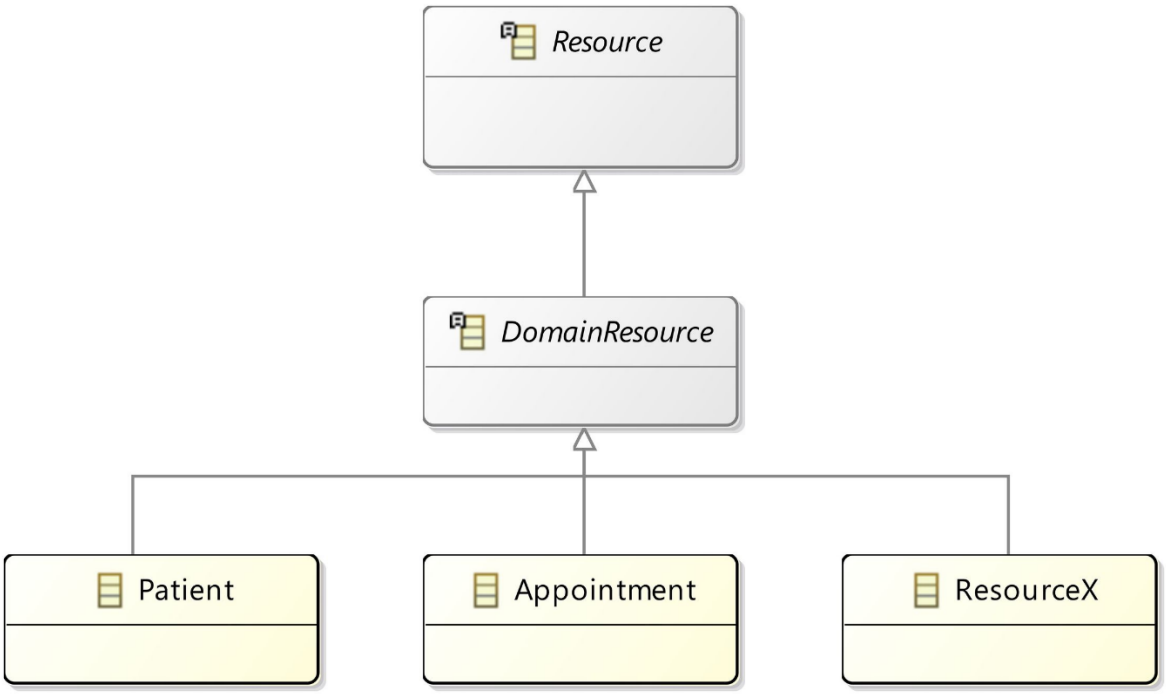


Рис. 2.3 Діаграма наслідування ресурсів

2.1.3. Розширення (Extensions)

Механізм розширення – це те, що відрізняє FHIR від HL7 версії 3 і, як не дивно, наближає його до версії 2. FHIR має філософію «80%» – конкретна властивість ресурсу включена лише в «основний» ресурс якщо на даний момент його використовують 80% існуючих систем.

Один із основних принципів FHIR полягає в тому, що специфікація включає лише ті структури, які фактично будуть використовуватися переважною більшістю реалізаторів. Це тому що сфера охорони здоров'я — це досить різноманітна галузь, в якій велика частина елементів будуть використовуватись лишень в вкрай специфічних випадках.

Слід мати на увазі кілька речей:

1. Немає офіційного статистичного аналізу. Робоча група, відповідальна за ресурс, робить приблизну оцінку на основі їх знань про бізнес-сферу. У

більшості випадків результат оцінки є очевидним – або дуже мало систем використовують цей елемент, або майже всі системи використовують його.

2. Оцінка базується на повному обсязі ресурсу, беручи до уваги реалізацію у всіх країнах, усі середовища реалізації, усі види догляду тощо. Можливо, що елемент з'явиться в більшості (або навіть у всіх) системах у певному середовищі, але зустрічається вкрай рідко в більшості середовищ.
3. Оцінка базується на тому, які системи підтримують. Наприклад, більшість систем охорони здоров'я підтримують відстеження дати смерті пацієнта, навіть якщо вона може не бути заповнена для більшості пацієнтів у цих системах.
4. У рідкісних випадках винятки з правила 80% можуть бути зроблені за погодженням групи управління FHIR для підтримки ключових випадків використання.

Використання цього підходу має ту перевагу, що самі ресурси залишаються керованого розміру, і їх набагато легше реалізувати, ніж якщо б кожна вимога з кожної сфери мала бути в кожному ресурсі.

2.1.4. Типи даних (Data types)

Кожен елемент у ресурсі має певний тип даних.

Специфікація FHIR визначає набір типів даних, які використовуються для елементів ресурсу. Є дві категорії типів даних: прості/примітивні типи, які є поодинокими елементами, і складні типи, які збіркою інших типів даних.

Типи даних FHIR є спрощеною версією типів даних HL7 v3 і також засновані на схемі W3C. На рис 2.4 наведено основні типи даних.

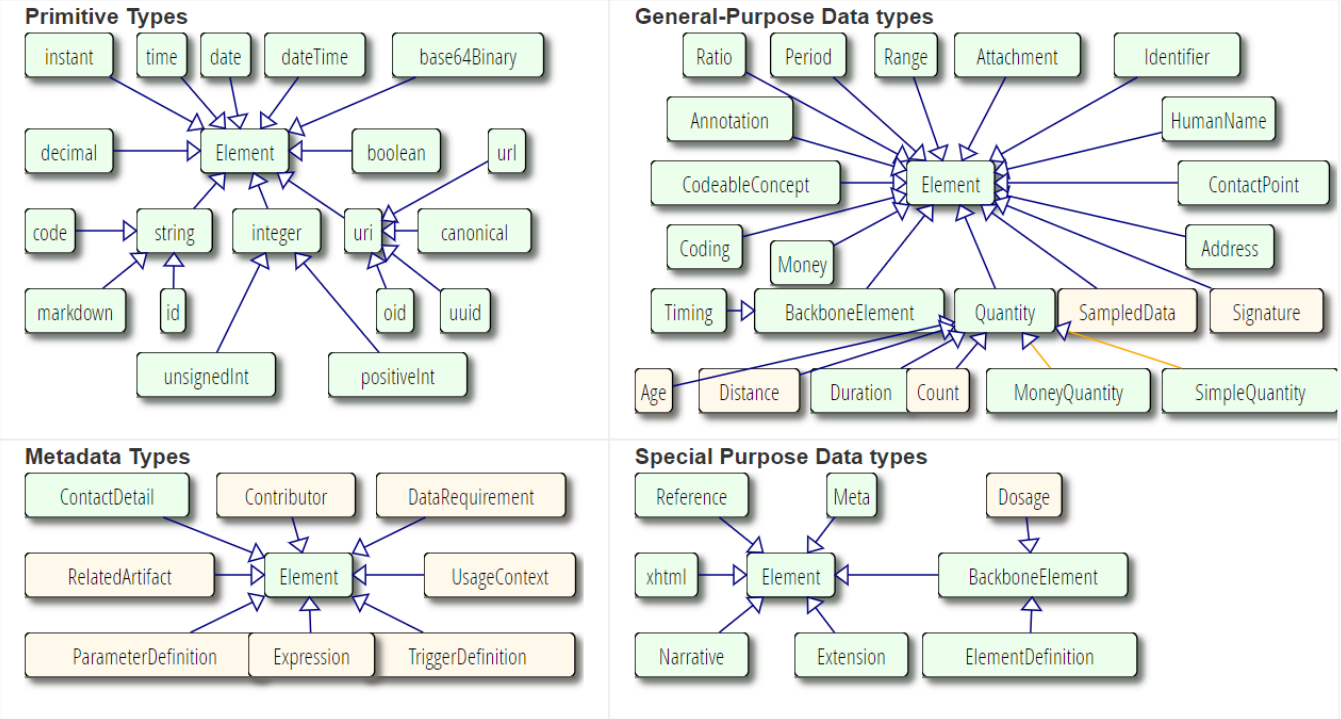


Рис. 2.4. Основні типи даних

2.1.5. Колекції (Bundles)

Існує багато ситуацій, коли потрібна коллекція ресурсів. До них належать:

1. Результати пошуку.
2. Колекція версій певного ресурсу.
3. Документ FHIR.
4. Повідомлення FHIR.
5. Колекція ресурсів котрі потрібно обробити.

Пакет містить певну інформацію в заголовку, а потім будь-яку кількість ресурсів. Заголовок містить:

1. Назву каналу.
2. Час (в форматі epoch), коли він був створений

3. Ідентифікатор для колекції. Ідентифікатор використовується при зберіганні колекції.
4. Ряд елементів посилання. Вони використовуються для опису програми, яка створила колекцію, і URL-адрес, які можна використовувати для пагінації великих колекцій (якщо сервер підтримує пагінацію).

Кожен запис містить:

1. Назву. Вона потрібна для підтримки атомарності і може бути використана, щоб надати короткий зміст запису.
2. Ідентифікатор. Це абсолютний URI, який вказує на логічний ідентифікатор ресурсу.
3. Час, коли ресурс у записі було останнє оновлено.

2.1.6. Профілі (Profiles)

Специфікація FHIR описує набір базових ресурсів, фреймворків та API, які використовуються в багатьох різних контекстах у сфері охорони здоров’я. Однак існує велика різниця між юрисдикціями в усій екосистемі охорони здоров’я щодо практик, вимог, правил. З цієї причини специфікація FHIR є «специфікацією платформи» – вона створює спільну платформу або основу, на якій реалізовано різноманітні рішення. Як наслідок, ця специфікація зазвичай вимагає подальшої адаптації до конкретних контекстів використання. Як правило, ці адаптації визначають:

1. Правила щодо того, які елементи ресурсу слід використовувати, і як додаються елементи, що не є частиною базової специфікації.
2. Правила про те, які функції API використовуються та як.
3. Опис того, як елементи ресурсу та функції API співвідносяться з локальними вимогами та/або реалізацією

2.2. RESTful в контексті FHIR. Базові CRUD операції

FHIR працює на багатьох різних парадигмах інтероперабельності, але REST є найкращим на даний момент – і, безумовно, найпростішим у використанні.

REST побудований на основі протоколу HTTP. HTTP (скорочення від HyperText Transfer Protocol) — це просто набір правил, які як сервер, так і клієнт (у даному випадку браузер) використовують для обміну даними через Інтернет. HTTP — це протокол «запит/відповідь» — ви робите запит (який може містити дані), сервер відповідає і все.

REST працює з ресурсами, кожен з яких має ідентичність, за допомогою якої ним можна маніпулювати. Це URI (Уніфікований ідентифікатор ресурсу) який однозначно визначає розташування ресурсу.

GET – запит дозволяє отримати певний ресурс на основі його URI (розташування сервера + ідентифікатор). FHIR використовує метод GET як для отримання одного ресурсу, так і для виконання пошуку, який може повернути кілька ресурсів.

POST – запит дозволяє створити новий ресурс у вказаній кінцевій точці. У REST сервер призначає ідентифікатор, а комбінація розташування сервера та ідентифікатора стає URI для цього ресурсу.

PUT – запит використовується для оновлення існуючого ресурсу. Клієнт, який виконує PUT, повинен знати URI ресурсу (що в FHIR означає, що він знає ідентифікатор ресурсу, а також сервер, на якому він зберігається). У FHIR це створить нову версію ресурсу.

DELETE – запит дозволяє видалити ресурс. Знову ж таки, URI має бути відомий. У FHIR рекомендується зберігати копію ресурсу та робити її доступною за допомогою зчитування з урахуванням версії та історії, але звичайний GET не повинен повертати видалений ресурс. (Однак він поверне код статусу, який повідомляє користувачеві, що ресурс було видалено.)

OPTIONS – це «спеціальний» метод у FHIR, який спрямовується до кореня сервера і повертає Conformance ресурс, який інформує клієнта про можливості сервера

Коли сервер відповідає на запит HTTP REST, він обробляє виклик відповідно до методу та внутрішньої бізнес-логіки, а потім повертає відповідь клієнту. Відповідь складається з кількох частин:

- Тіло відповіді – це те, що клієнт запитує від сервера – зазвичай у відповідь на запит GET. Більшість інших методів не повертають відповідь.
- Сервер завжди повертає код статусу, який повідомляє клієнту, яким був результат запиту – успішним чи невдалим – з деякими додатковими деталями щодо результату. Код статусу — це число, яке має чітко визначене значення в протоколі HTTP. FHIR визначає ресурс OperationOutcome, який сервер може використовувати, щоб надати клієнту детальну інформацію про будь-який збій.

І запит, і відповідь містять заголовки. Вони надають додаткову інформацію про запит або відповідь, щоб допомогти серверу або клієнту правильно обробити транзакцію.

2.2.1. Структура запита

VERB [base]/[type]/[id]{/_history/[vid]}{?_format=[mime-type]&parameter=value}

Перше слово – це дієслово HTTP (GET/PUT..)

Вміст, оточений символом [], є обов’язковим і буде замінено ідентифікатором. Можливі значення:

1. base: URL-адреса сервера
2. mime-type: тип медіа-типу

3. type: назва типу ресурсу (наприклад, «Patient»)
4. id: логічний ідентифікатор ресурсу
5. vid: ідентифікатор версії ресурсу
6. parameters: параметри URL-запиту

Медіа типи

Формальним типом MIME для ресурсів FHIR є application/fhir+xml або application/fhir+json. Клієнти та сервери МАЮТЬ використовувати правильний тип mime-type:

- XML: application/fhir+xml
- JSON: application/fhir+json
- RDF: application/fhir+turtle

2.2.2. Умовні запити

Стандарт FHIR підтримує операції умовного читання, створення, оновлення та видалення.

1. Умовне читання. Клієнти можуть використовувати HTTP-заголовок If-Modified-Since або If-None-Match у запиті на читання. Якщо так, вони ПОВИННІ прийняти або 304 Not Modified як дійсний код статусу у відповіді (що означає, що вміст ресурсу не було змінено), або повний вміст (або вміст змінився, або сервер не підтримує умовний запит).
2. Умовне створення. Взаємодія умовного створення дозволяє клієнту створити новий ресурс, лише якщо якийсь еквівалентний ресурс ще не існує на сервері.
3. Взаємодія умовного оновлення дозволяє клієнту оновлювати існуючий ресурс на основі деяких критеріїв ідентифікації, а не за логічним ідентифікатором.
4. Взаємодія умовного видалення дозволяє клієнту видалити наявний ресурс на основі деяких критеріїв вибору, а не за певним логічним ідентифікатором.

Таблиця 2.1

Ідентифікатори, котрі використовуються під час умовних запитів.

Ідентифікатор	Представлення в HTTP
Логічний ідентифікатор	Цей ідентифікатор явно представлений в URL-адресі
Ідентифікатор версії	Представлений у заголовку ETag
Останнє оновлення	Представлений у заголовку Last-Modified

2.3. Висновки до другого розділу

FHIR – це де-факто основний стандарт сьогодення, завдяки котрому можна реалізувати веб-сервіс для обігу, валідації і зберігання медичних документів. При складанні специфікації FHIR було враховано минулий досвід в створенні медичних стандартів, як то HL7 і CDA.

Набір правил FHIR стандарту забезпечує високий рівень інтероперабельності веб-сервісу.

Основним інструментом досягнення інтероперабельності в стандарті FHIR є чітке детермінування того, що є ресурс та використання добре-знаних технологій. Використання об’єктно-орієнтованого підходу для ресурсів дозволяє побудувати чітку ієрархію.

Розширюваність є особливістю, котра надає можливість розробки сервісу навіть для вкрай специфічних середовищ сфери охорони здоров’я.

REST-підхід є найдоступнішим та найлегшим способом організації API сервісу, котрий не вимагає спеціальних знань.

РОЗДІЛ 3
ІМПЛЕМЕНТАЦІЯ ВЕБ ДОДАТКУ НА ОСНОВІ СТАНДАРТУ FHIR v4.1

3.1. Принцип роботи додатку

В результаті дослідження в розділах 1 та 2 було вирішено використовувати серверну архітектуру для функціонування додатку. Додаток вирішено написати на мультиплатформовій мові Java.

Другим кроком буде написання профілю для тестового ресурсу. Зазвичай, за збором вимог у бізнеса то формулюванням їх перед розробниками відповідає бізнес-аналітик. Тому припустимо, що нам передали вимогу написати профіль для ресурса Patient, додавши до нього обов'язкові поля.

Після цього нам буле потрібно продумати та реалізувати принцип роботи додатку, його Public API.

3.1.1. Правила профілювання

Основна ідея профілювання в тому, що профілювання дає нам змогу реалізувати ті 20%, котрі не реалізовані впроваджувачами стандарту, себто розширити стандартний ресурс, додати кастомну валідацію, змінити кардинальність, навіть додати нові поля. Це дозволяє пристосовувати ресурс до реалій конкретної країни, або навіть конкретної системи охорони здоров'я (рис 3.1).

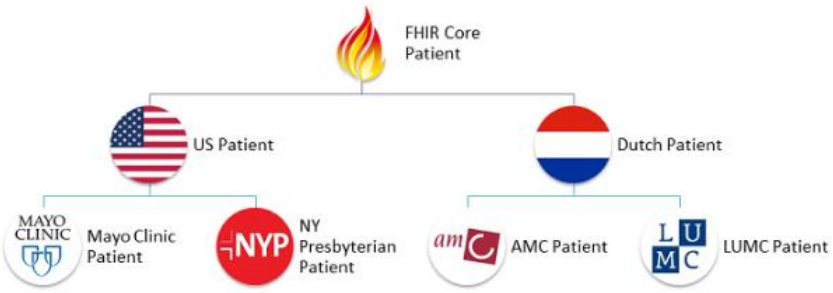


Рис. 3.1 Приклад можливості розробки профілю під конкретне середовище

Зазвичай прийнято профілювати основний ресурс, наприклад профіль для ресурсу «Patient». Можна використовувати шлях до елементу ресурсу і змінити його кардинальність. Наприклад, зробити поля «name» і «birthDate» обов’язковим, а поле «photo» - заборонити (рис 3.2).

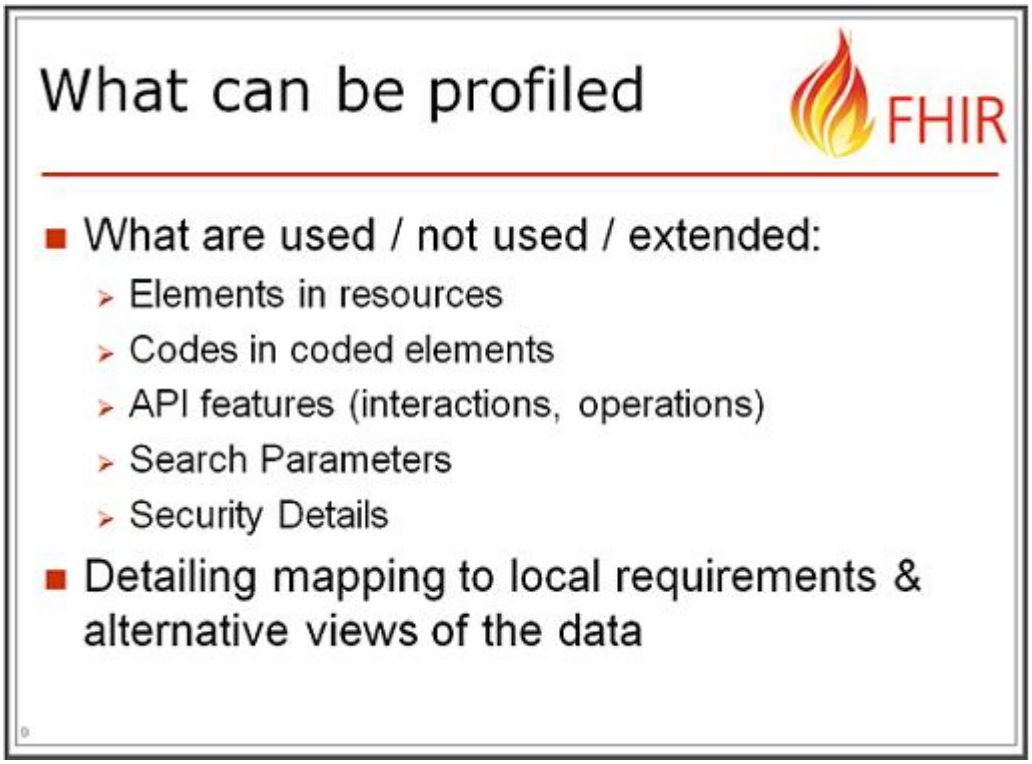


Рис 3.2 Що можна профілювати

Кардинальність елементу визначає можливу кількість допустимих сутностей цього елементу (табл. 3.1).

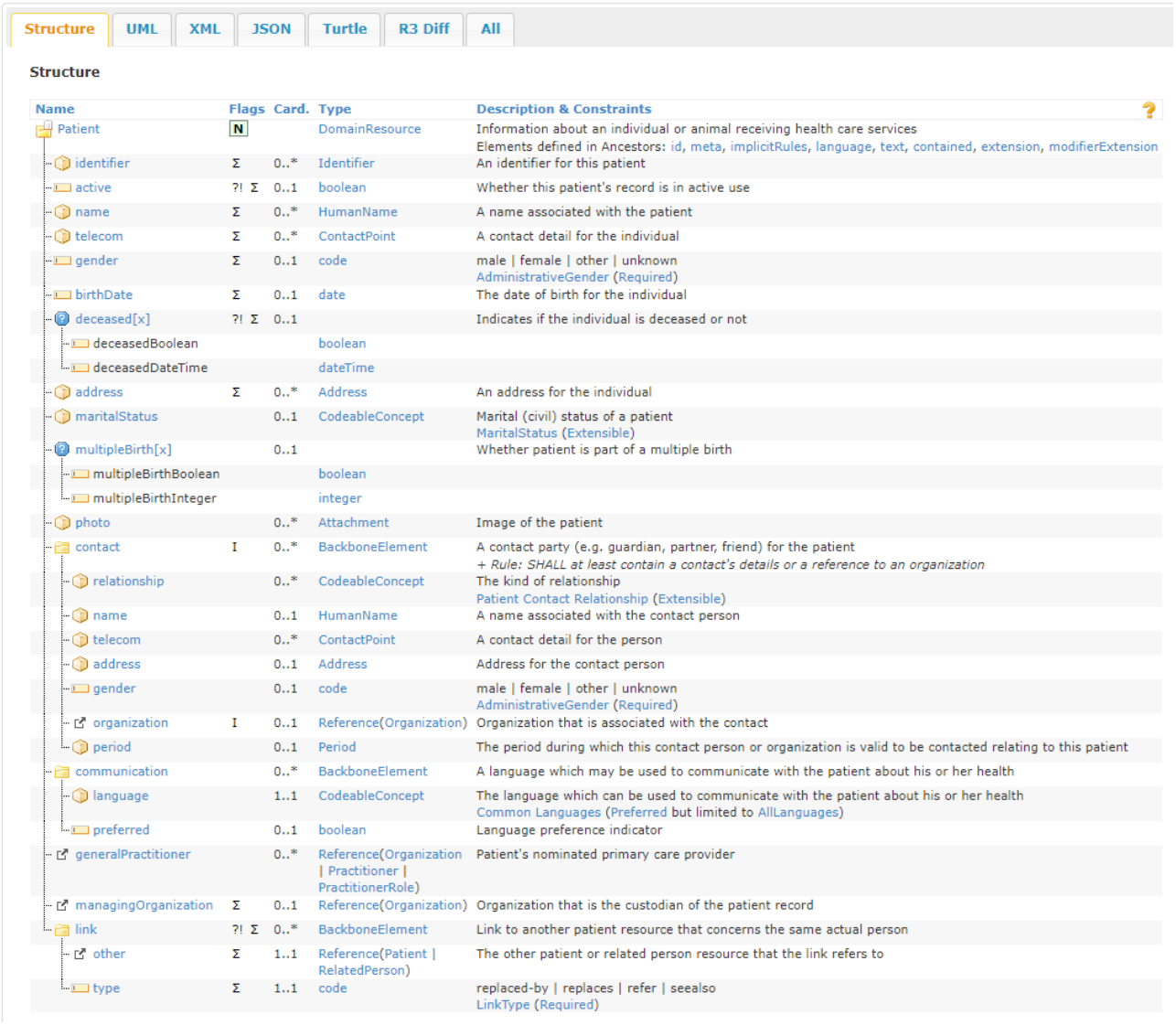
Таблиця 3.1

Найрозповсюдженіші значення кардинальності.

Кардинальність	Значення
0..0	Не використовується
0..1	Опціонально один елемент
0..*	Опціонально багато елементів
1..1	Один і тільки один
1..*	Хоча б один

3.2. Створення профілю для ресурса Patient

Стандартний ресурс Patient спроектовано таким чином, що в ньому немає жодних обов'язкових полів – тобто, можна створити повністю пустого Patient-а.



Structure | UML | XML | JSON | Turtle | R3 Diff | All

Structure

Name	Flags	Card.	Type	Description & Constraints
Patient	N		DomainResource	Information about an individual or animal receiving health care services Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtension
identifier	Σ	0..*	Identifier	An identifier for this patient
active	?! Σ	0..1	boolean	Whether this patient's record is in active use
name	Σ	0..*	HumanName	A name associated with the patient
telecom	Σ	0..*	ContactPoint	A contact detail for the individual
gender	Σ	0..1	code	male | female | other | unknown AdministrativeGender (Required)
birthDate	Σ	0..1	date	The date of birth for the individual
deceased[x]	?! Σ	0..1		Indicates if the individual is deceased or not
deceasedBoolean			boolean	
deceasedDateTime			dateTime	
address	Σ	0..*	Address	An address for the individual
maritalStatus		0..1	CodeableConcept	Marital (civil) status of a patient MaritalStatus (Extensible)
multipleBirth[x]		0..1		Whether patient is part of a multiple birth
multipleBirthBoolean			boolean	
multipleBirthInteger			integer	
photo		0..*	Attachment	Image of the patient
contact	I	0..*	BackboneElement	A contact party (e.g. guardian, partner, friend) for the patient *+ Rule: SHALL at least contain a contact's details or a reference to an organization*
relationship		0..*	CodeableConcept	The kind of relationship Patient Contact Relationship (Extensible)
name		0..1	HumanName	A name associated with the contact person
telecom		0..*	ContactPoint	A contact detail for the person
address		0..1	Address	Address for the contact person
gender		0..1	code	male | female | other | unknown AdministrativeGender (Required)
organization	I	0..1	Reference(Organization)	Organization that is associated with the contact
period		0..1	Period	The period during which this contact person or organization is valid to be contacted relating to this patient
communication		0..*	BackboneElement	A language which may be used to communicate with the patient about his or her health
language		1..1	CodeableConcept	The language which can be used to communicate with the patient about his or her health Common Languages (Preferred but limited to AllLanguages)
preferred		0..1	boolean	Language preference indicator
generalPractitioner		0..*	Reference(Organization | Practitioner | PractitionerRole)	Patient's nominated primary care provider
managingOrganization	Σ	0..1	Reference(Organization)	Organization that is the custodian of the patient record
link	?! Σ	0..*	BackboneElement	Link to another patient resource that concerns the same actual person
other	Σ	1..1	Reference(Patient | RelatedPerson)	The other patient or related person resource that the link refers to
type	Σ	1..1	code	replaced-by | replaces | refer | seealso LinkType (Required)

Рис 3.3 Контент стандартного ресурсу Patient

Отже, нам потрібен профіль для того щоб змінити контент стандартного профілю. Назвемо цей профіль SimplePatient. Всі профілі є сутностями спеціального ресурсу – StructureDefinition. Визначимо такі правила, що:

1. У Patient повинен бути присутнім хоча б один елемент name.
2. Довжина поля family елементу name повинна бути не більше 40 символів.
3. Довжина поля given елементу name повинна бути не більше 45 символів.

4. Повинно бути мінімум один і максимум три елемента telecom.
5. Повинен бути один і тільки один елемент birthDate.
6. Елементу photo не повинно бути взагалі.

Відповідно до вимог, котрі ми склали вище, розробимо StructureDefinition для створення профілю.

				Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtension
url	Σ	1..1	uri	Canonical identifier for this structure definition, represented as a URI (globally unique)
identifier	Σ	0..*	Identifier	Additional identifier for the structure definition
version	Σ	0..1	string	Business version of the structure definition
name	Σ I	1..1	string	Name for this structure definition (computer friendly)
title	Σ	0..1	string	Name for this structure definition (human friendly)
status	?! Σ	1..1	code	draft | active | retired | unknown PublicationStatus (Required)
experimental	Σ	0..1	boolean	For testing purposes, not real usage
date	Σ	0..1	dateTime	Date last changed
publisher	Σ	0..1	string	Name of the publisher (organization or individual)
contact	Σ	0..*	ContactDetail	Contact details for the publisher
description		0..1	markdown	Natural language description of the structure definition
useContext	Σ TU	0..*	UsageContext	The context that the content is intended to support
jurisdiction	Σ	0..*	CodeableConcept	Intended jurisdiction for structure definition (if applicable) Jurisdiction (Extensible)
purpose		0..1	markdown	Why this structure definition is defined
copyright		0..1	markdown	Use and/or publishing restrictions
keyword	Σ	0..*	Coding	Assist with indexing and finding Structure Definition Use Codes / Keywords (Extensible)
fhirVersion	Σ	0..1	code	FHIR Version this StructureDefinition targets FHIRVersion (Required)
mapping	I	0..*	BackboneElement	External specification that the content is mapped to *+ Rule: Must have at least a name or a uri (or both)*
identity		1..1	id	Internal id when this mapping is used
uri	I	0..1	uri	Identifies what this mapping refers to
name	I	0..1	string	Names what this mapping refers to
comment		0..1	string	Versions, Issues, Scope limitations etc.
kind	Σ	1..1	code	primitive-type | complex-type | resource | logical StructureDefinitionKind (Required)
abstract	Σ	1..1	boolean	Whether the structure is abstract
context	Σ I	0..*	BackboneElement	If an extension, where it can be used in instances
type	Σ	1..1	code	fhirpath | element | extension ExtensionContextType (Required)
expression	Σ	1..1	string	Where the extension can be used in instances
contextInvariant	Σ I	0..*	string	FHIRPath invariants - when the extension can be used
type	Σ I	1..1	uri	Type defined or constrained by this structure FHIRDefinedType (Extensible)
baseDefinition	Σ I	0..1	canonical(StructureDefinition)	Definition that this type is constrained/specialized from
derivation	Σ	0..1	code	specialization | constraint - How relates to base definition TypeDerivationRule (Required)
snapshot	I	0..1	BackboneElement	Snapshot view of the structure *+ Rule: Each element definition in a snapshot must have a formal definition and cardinalities* *+ Rule: All snapshot elements must start with the StructureDefinition's specified type for non-logical models, or with the same type name for logical models* *+ Rule: All snapshot elements must have a base definition*
element	I	1..*	ElementDefinition	Definition of elements in the resource (if no StructureDefinition) *+ Rule: provide either a binding reference or a description (or both)*
differential	I	0..1	BackboneElement	Differential view of the structure *+ Rule: No slicing on the root element* *+ Rule: In any differential, all the elements must start with the StructureDefinition's specified type for non-logical models, or with the same type name for logical models*
element		1..*	ElementDefinition	Definition of elements in the resource (if no StructureDefinition)

Рис 3.4 Контент ресурсу StructureDefinition.

У ресурса StructureDefinition обов’язково потрібні бути наступні поля:

1. url – це uri, за якою ми однозначно можемо ідентифікувати наш профіль.
2. name – це назва нашого профілю.

3. status – статус профілю. Може бути чотири можливих статуси: draft, active, retired, unknown.
4. kind – тип профілю. Може бути доступно чотири можливих типи: primitive-type | complex-type | resource | logical
5. abstract – це булінова змінна, котра означає, чи цей StructureDefinition абстрактний, чи його можна реалізувати.
6. differential або snapshot – одне із цих полів обов'язково повинне бути присутнім в профілі, аби показати різницю між ним і стандартним ресурсом. В differential потрібно описати тільки ті елементи, які ми змінюємо, а в snapshot – повністю ресурс. Реалізатору набагато простіше описувати differential, але веб-сервіс працює з snapshot.

Отже, за поставленими вимогами було розроблено наступний профіль:


```
{
  «resourceType»: «StructureDefinition»,
  «id»: «SimplePatient»,
  «url»: «http://somewhere.org/fhir/uv/myig/StructureDefinition/SimplePatient»,
  «version»: «1»,
  «name»: «SimplePatient»,
  «title»: «My Simple Patient Profile»,
  «status»: «draft»,
  «fhirVersion»: «4.0.1»,
  «kind»: «resource»,
  «abstract»: false,
  «type»: «Patient»,
  «baseDefinition»: «http://hl7.org/fhir/StructureDefinition/Patient»,
  «derivation»: «constraint»,
  «differential»: {
    «element»: [
      {
        «id»: «Patient.name»,
        «path»: «Patient.name»,
        «short»: «Baidak master diploma SimplePatient»,
        «definition»: «Baidak master diploma SimplePatient»,
        «min»: 1
      },
      {
        «id»: «Patient.name.family»,
        «path»: «Patient.name.family»,
        «maxLength»: 40
      },
      {
        «id»: «Patient.name.given»,
        «path»: «Patient.name.given»,
        «maxLength»: 45
      },
      {
        «id»: «Patient.telecom»,
        «path»: «Patient.telecom»,
```

```
        «min»: 1,
        «max»: «3»
      },
      {
        «id»: «Patient.birthDate»,
        «path»: «Patient.birthDate»,
        «min»: 1,
        «max»: «1»
      },
      {
        «id»: «Patient.photo»,
        «path»: «Patient.photo»,
        «max»: «0»
      }
    ]
  }
}
```


Наступним кроком буде створення Snapshot(«Знімок», профіль, котрий включає наші зміни разом з стандартним ресурсом) з Differential(«Різниця», профіль суто з нашими змінами). Для цього нам потрібно з наданих ресурсів створити особливий документ, який називається Implementation Guide.

Implementation Guide – це документ, опублікований доменом, установою або постачальником, котрий описує, як FHIR адаптований до певного, специфічного варіанту використання. Тобто, це такий документ, який описує різницю між стандартним ресурсами і наданими постачальниками. Implementation Guide об’єднує набір ресурсів котрі підтримуються розробником. Implementation Guide — це набір правил про те, як ресурси FHIR використовуються (або повинні використовуватися) для вирішення певної проблеми, з пов’язаною документацією для підтримки та пояснення використання(рис 3.4).

Основні можливості використання Implementation Guide:

1. Інформація про FHIR: канонічна URL-адреса, версія стандарту
2. Інформація про пакет FHIR: ім’я, ідентифікатор, NPM, ліцензія, залежність від інших посібників із впровадження тощо.
3. Визначення FHIR: усі ресурси (включаючи StructureDefinitions), приклади, параметри, сторінки, які будуть включені.
4. Збірка пакету FHIR: інформація про зібраний посібник із впровадження.

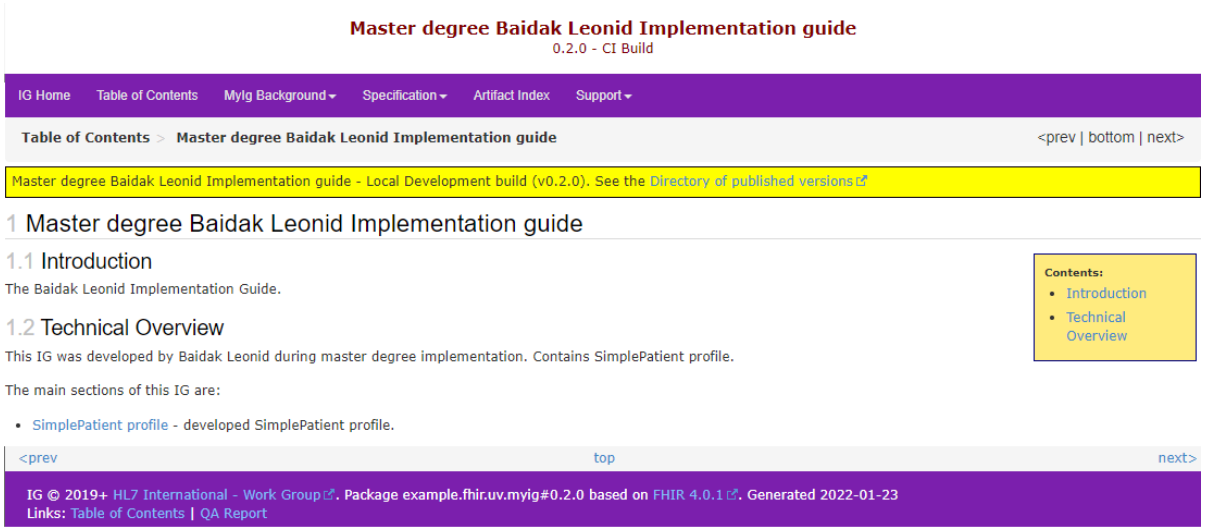


Рис 3.4 Індекс сторінка розробленого Implementation Guide.

Перейшовши по посиланню на головній сторінці, ми можемо знайти документацію по розробленому профілю (рис 3.5).

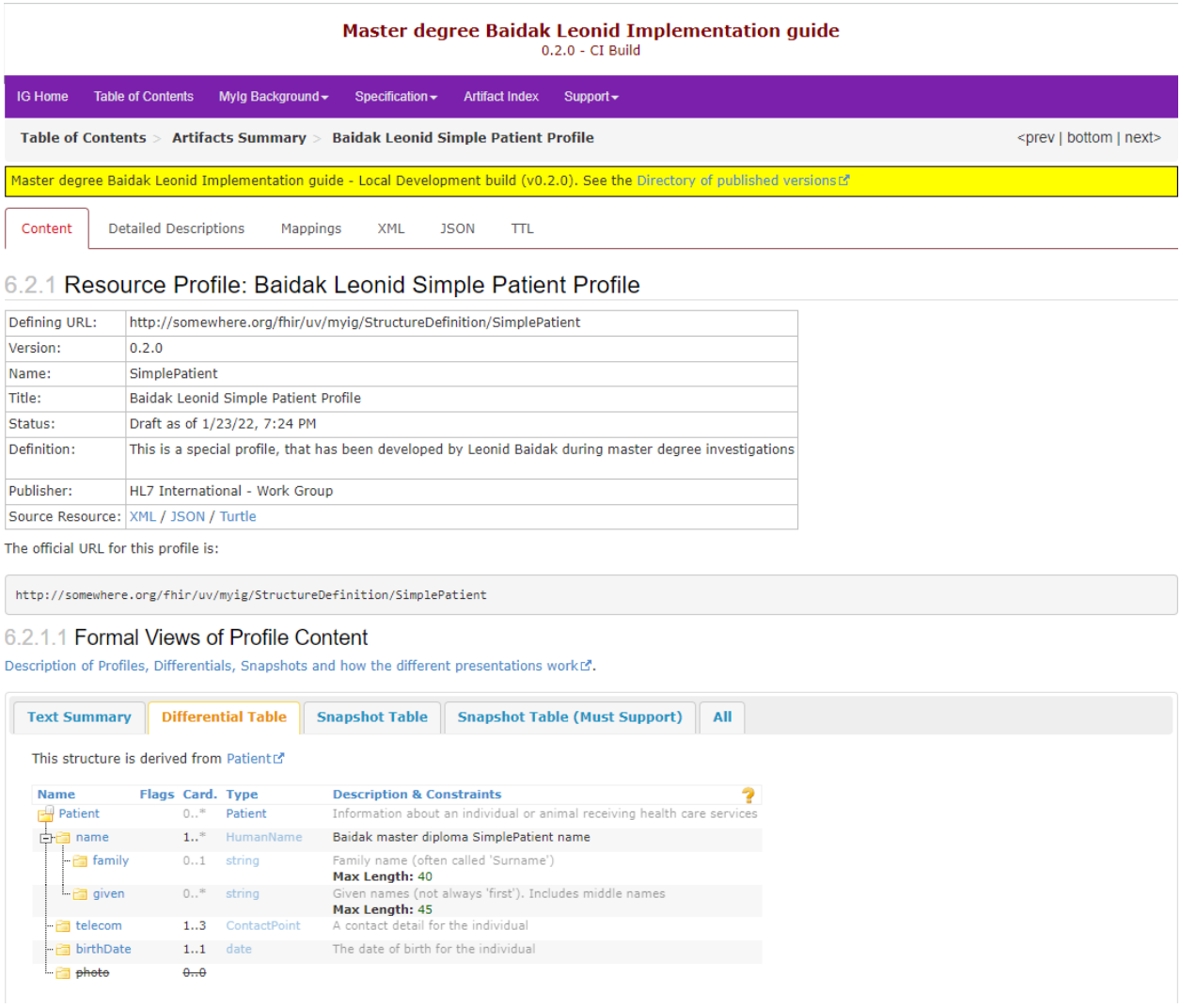


Рис 3.5 Документація по розробленому профілю SimplePatient

3.3. Розробка веб-сервісу

Концепція веб-сервісу полягає в тому, що розроблений Implementation Guide буде використовуватись як бібліотека, а отже користувачі сервісу відразу зможуть переглядати необхідну документацію.

Сервіс передбачає 4 можливих операції для ресурсу Patient – Create, Read, Update і Delete. Операція Create буде виконуватись за допомогою http POST запиту, Read – GET запиту, Update – PUT запиту і Delete – DELETE запиту. Діаграма використання наступна:

1. Create. При створенні буде валідуватися тіло запиту. Якщо там будуть порушені накладені обмеження – користувач не зможе створити сутність і отримає 400 BAD_REQUEST. Якщо така сутність вже існує в базі даних, користувач отримає 409 CONFLICT статус у якості відповіді. Якщо користувач буде намагатись зробити запит на неіснуючу URL, він отримає 404 NOT_FOUND. У якості успіху користувач отримає 200 OK статус і створену сутність в форматі application/fhir+json разом з присвоєним їй ідентифікатром. Use-case запиту на створення наведено на рис. 3.6.

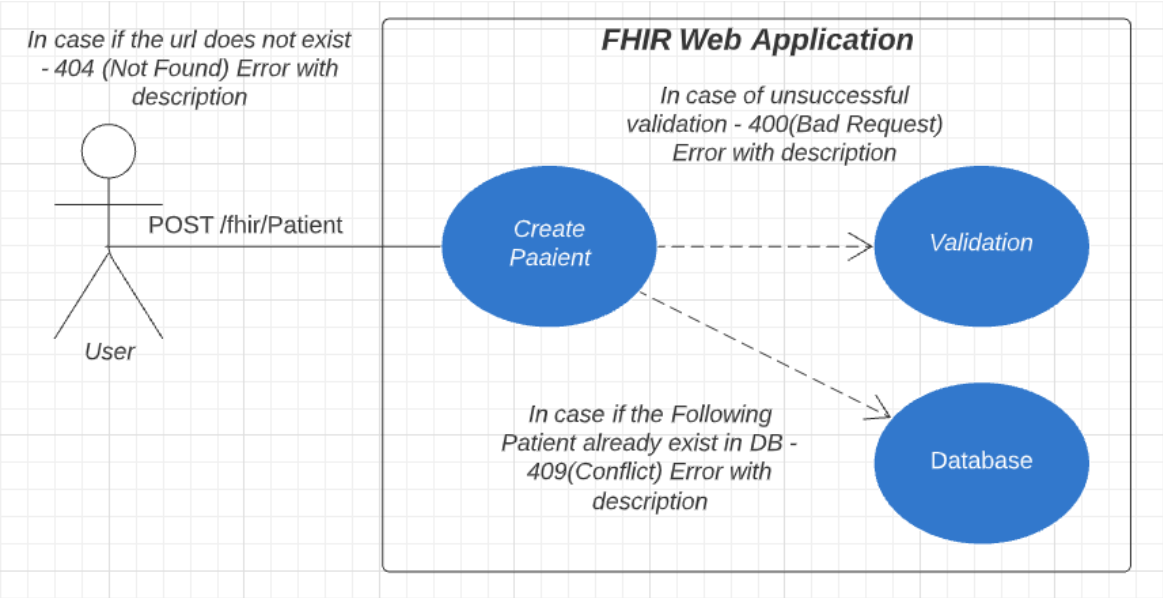


Рис 3.6 Use-case POST запиту

2. Read. Якщо сутності не існує в БД, користувач отримає 404 NOT_FOUND статус у якості відповіді. Якщо користувач буде намагатись зробити запит на неіснуючу URL, він також отримає 404 NOT_FOUND. У якості успіху користувач отримає 200 OK статус і сутність в форматі application/fhir+json. Use-case запиту на читання наведено на рис. 3.7.

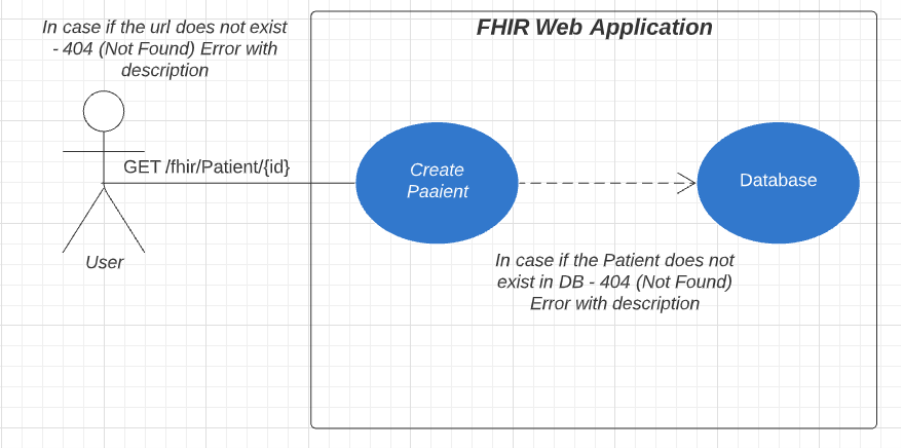


Рис 3.7 Use-case GET запиту

3. Update. При оновленні буде валідуватися тіло запиту. Якщо там будуть порушені накладені обмеження – користувач не зможе оновити сутність і отримає 400 BAD_REQUEST. Якщо така сутність вже існує в базі даних, користувач отримає 409 CONFLICT статус у якості відповіді. Якщо користувач буде намагатись зробити запит на неіснуючу URL, він отримає 404 NOT_FOUND. У якості успіху користувач отримає 200 OK статус і створену сутність в форматі application/fhir+json. Use-case запиту на створення наведено на рис. 3.8.

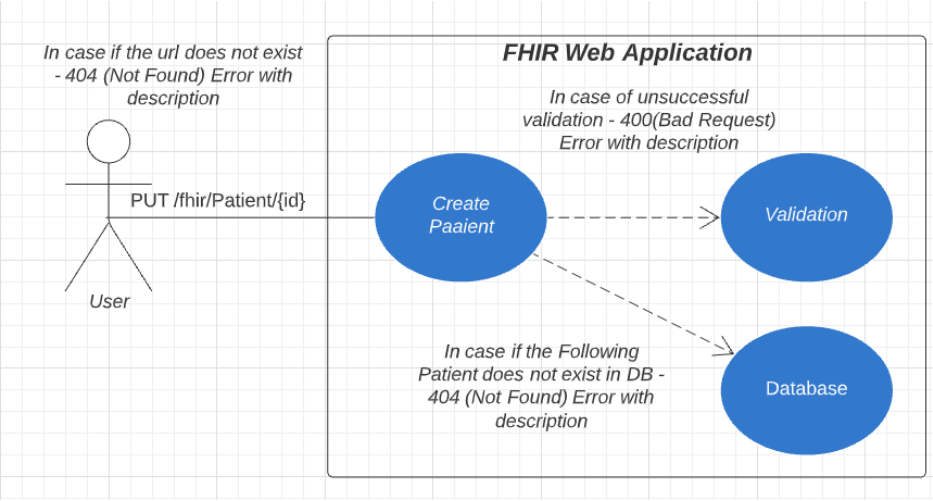


Рис 3.8 Use-case PUT запиту

4. Delete. Якщо сутності не існує в БД, користувач отримає 404 NOT_FOUND статус у якості відповіді. Якщо сутність вже була позначена як видалена, користувач отримає 404 NOT_FOUND. У якості успіху користувач отримає 200 OK статус і сутність в форматі application/fhir+json, а сутність буде позначену як видалену, а отже взаємодія з нею буде неможливою. Use-case запиту на читання наведено на рис. 3.8.

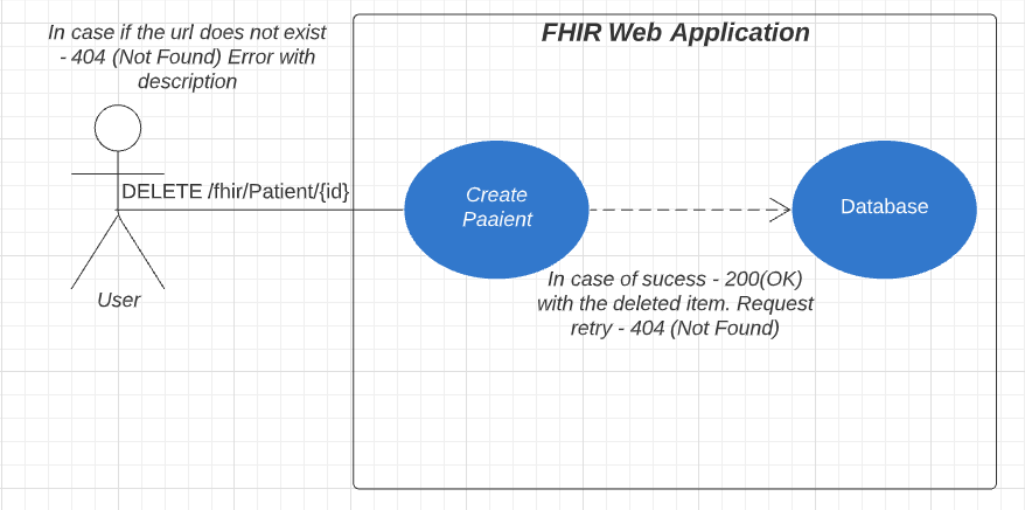


Рис 3.8 Use-case DELETE запиту

3.3.1. Проектування бази даних

У якості бази даних буде використовуватись реляційна база даних Microsoft SQL. Основна мова запитів до цієї бази даних – це модифікована версія звичайного SQL, котра називається Transact-SQL.

База даних буде мати всього одну значущю таблицю Patient, та дві таблиці з метаінформацією – DATABASECHANGELOGLOCK і DATABASECHANGELOG. В таблиці Patient будуть наступні атрибути:

1. PatientID, де буде зберігатись унікальний ідентифікатор пацієнта. Тип даних – BIGINT. Атрибут підтримує автоінкремент. Атрибут не може бути пустим.
2. Data, де буде зберігатись сутність в текстовому форматі. Тип даних – VARCHAR(2048). Атрибут не може бути пустим.

3. VersionTimestamp, де буде зберігатись час останньої модифікації ресурсу. Тип даних – BIGINT. Атрибут не може бути пустим.
4. Deleted, де зберігається булінове значення, яке означає, чи було сутність видалено. Тип даних – BIT. Атрибут не може бути пустим.
5. Version, де зберігається значення про версії сутності. Тип даних – INT. Атрибут не може бути пустим.
6. Created(DATETIME),CreatedBy(VARCHAR(50)), CreatedByHost(VARCHAR(50)),LastModified(DATETIME), LastModifiedBy(VARCHAR(50)), LastModifiedByHost(VARCHAR(50)) є службовими атрибутами, котрі несуть мента інформацію про те, коли і ким була створена або змінена сутність. Атрибути будуть заповнюватись за допомогою спеціального тригера,

Обмеження унікальності буде накладене на атрибут Data, тобто у нас не може бути двох сутностей з повність однаковим атрибутом Data.

Первинним ключем буде виступати PatientID атрибут.

База даних буде створена за допомогою 41iquibase – це незалежна від вендора бази даних бібліотека з відкритим кодом для відслідковування, керування і застосування змін схеми бази даних. Інструкції в бібліотеці 41iquibase описуються за допомогою xml, yaml, json і sql. Отже, структура бази даних наведена на рис 3.9.

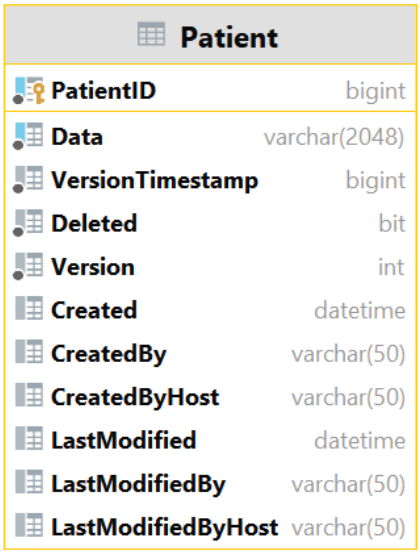


Рис 3.9 Структура бази даних

3.3.2. Архітектура додатку

Додаток має серверну архітектуру. Додаток реалізовано на основі мови Java та стеку фреймворків Spring. Public API створено завдяки популярному підходу REST. Також на сервері знаходиться база даних. На сервері додано публічну документацію, описану за допомогою фреймворку Swagger. Також, щоб бути повноцінним FHIR-додатком, сервіс повинен реалізовувати CapabilityStatement – це спеціальний FHIR-ресурс, котрий дозволяє отримати мета інформацію про сервіс, можливі взаємодії з ним.

3.3.3. Реалізація серверної частини

Серверна частина базується на технології сервлетів. Java Servlet API — стандартизований API для створення динамічного контенту до веб-сервера, використовуючи платформу Java. Сервлети — аналог технологій PHP, CGI і ASP.NET. Сервлет може зберігати інформацію між багатьма транзакціями, використовуючи HTTP, сесії або через редагування URL.

Servlet API, що міститься в пакеті javax.servlet, описує взаємодію веб-контейнера і сервлета. Веб-контейнер — це компонент веб-сервера, що створений для взаємодії з сервлетами. Він відповідає за управління життєвим циклом сервлетів, перетворення URL у певний сервлет та забезпечення того, щоб клієнт, який зробив URL запит, мав відповідні права доступу.

Сервлети хостяться на Apache Tomcat. Apache Tomcat — контейнер сервлетів, розроблений Apache Software Foundation. Повністю написаний мовою програмування Java та реалізує специфікацію сервлетів і Java Server Pages від Sun Microsystems, що є стандартами для розробки веб-застосунків на Java.

Для роботи з базою даних використовується фреймворк Hibernate, котрий реалізує специфікацію Java Persistence API — стандартизований інтерфейс для Java ORM фреймворків. ORM (Object-relational mapping) — технологія

програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних».

Вирішення проблеми зберігання даних існує — це реляційні системи управління базами даних. Використання реляційної бази даних для зберігання об'єктно-орієнтованих даних приводить до семантичного провалу, примушуючи програмістів писати програмне забезпечення, яке повинне вміти як обробляти дані в об'єктно-орієнтованому вигляді, так і вміти зберегти ці дані в реляційній формі. Ця постійна необхідність в перетворенні між двома різними формами даних не тільки сильно знижує продуктивність, але і створює труднощі для програмістів, оскільки обидві форми даних накладають обмеження одна на одну.

Реляційні бази даних використовують набір таблиць, що представляють прості дані. Додаткова або зв'язана інформація зберігається в інших таблицях. Часто для зберігання одного об'єкта в реляційній базі даних використовують кілька таблиць; це, у свою чергу, вимагає застосування операції JOIN для отримання всієї інформації, що стосується об'єкта, для її обробки.

Оскільки системи управління реляційними базами даних зазвичай не реалізують реляційного представлення фізичного рівня зв'язків, виконання кількох послідовних запитів (що стосуються однієї «об'єктно-орієнтованої» структури даних) може бути дуже витратним. Зокрема, один запит вигляду «знайти такого-то користувача і всі його телефони і всі його адреси і повернути їх у такому форматі», швидше за все, буде виконаний швидше за серію запитів вигляду «Знайти користувача. Знайти його адреси. Знайти його телефони». Це відбувається завдяки роботі оптимізатора і витратам на синтаксичний аналіз запиту.

Деякі реалізації ORM автоматично синхронізують завантажені в пам'ять об'єкти з базою даних. Для того, щоб це було можливим, після створення SQL-запиту, що перетворює об'єкт в SQL, отримані дані копіюються в поля об'єкта, як у всіх інших реалізаціях ORM. Після цього об'єкт повинен стежити за змінами цих значень і записувати їх у базу даних.

Системи управління реляційними базами даних показують хорошу продуктивність на глобальних запитах, які зачіпають велику ділянку бази даних, але об'єктно-орієнтований доступ ефективніший при роботі з малими обсягами даних, бо це дозволяє скоротити семантичний провал між об'єктною і реляційною формами даних.

При одночасному існуванні цих двох різних світів збільшується складність об'єктного коду для роботи з реляційними базами даних, і він стає схильнішим до помилок.

Розроблено безліч пакетів, що знімають необхідність у перетворенні об'єктів для зберігання в реляційних базах даних.

Деякі пакети вирішують цю проблему, надаючи бібліотеки класів, здатних виконувати такі перетворення автоматично. Маючи список таблиць в базі даних і об'єктів в програмі, вони автоматично перетворять запити з одного вигляду в іншій

З погляду програміста система повинна виглядати як постійне сховище об'єктів. Він може просто створювати об'єкти і працювати з ними як завжди, а вони автоматично зберігатимуться в реляційній базі даних.

На практиці все не так просто і очевидно. Всі системи ORM зазвичай проявляють себе в тому або іншому вигляді, зменшуючи в деякому роді можливість ігнорування бази даних. Більш того, шар транзакцій може бути повільним і неефективним (особливо в термінах згенерованого SQL). Все це може привести до того, що програми працюватимуть повільніше і використовуватимуть більше пам'яті, ніж програми, написані «вручну».

Але ORM позбавляє програміста від написання великої кількості коду, часто одноманітного і схильного до помилок, тим самим значно підвищуючи швидкість розробки. Крім того, більшість сучасних реалізацій ORM дозволяє програмістові при необхідності жорстко задати код SQL-запитів, який використовуватиметься при тих чи інших діях (збереження в базу даних, завантаження, пошук тощо) з постійним об'єктом.

Hibernate — засіб відображення між об'єктами та реляційними структурами для платформи Java. Hibernate є вільним програмним забезпеченням Hibernate надає легкий для використання каркас для відображення між об'єктно-орієнтованою моделлю даних і традиційною реляційною базою даних.

Hibernate забезпечує прозоре збереження POJO (Plain Old Java Objects — простих старих об'єктів Java). Єдина сувора вимога для персистентного класу — конструктор без аргументів, не обов'язково публічний. Для правильної поведінки деяких програм також потрібна особлива увага до методів equals() і hashCode().

Колекції об'єктів даних, як правило, зберігаються у вигляді колекцій Java-об'єктів, таких як набір (Set) і список (List). Підтримуються узагальнені класи (Generics), введені в Java 5. Hibernate може бути налаштований на «ледачі» (відкладені) завантаження колекцій. Відкладені завантаження є варіантом за умовчанням, починаючи з Hibernate 3.

Зв'язані об'єкти можуть бути налаштовані на каскадні операції. Наприклад, батьківський клас, Album (музичний альбом), може бути налаштований на каскадне збереження і/або видалення свого нащадка Track. Це може скоротити час розробки і забезпечити цілісність. Функція перевірки зміни даних (dirty checking) дозволяє уникнути непотрібного запису дій в базу даних, виконуючи SQL оновлення тільки при зміні полів персистентних об'єктів.

Для валідації ресурсів використовується відкрита бібліотека IBM FHIR – це модульна реалізація Java версії 4 специфікації HL7 FHIR з акцентом на продуктивність і можливість налаштування.

Для документування API використовується фреймворк Swagger – це мова опису інтерфейсу для опису RESTful API, виражених за допомогою JSON. Swagger використовується разом із набором програмних інструментів з відкритим кодом для проектування, створення, документування та використання веб-сервісів RESTful. Swagger включає в себе автоматизовану документацію, генерацію коду (на багатьох мовах програмування) і генерацію тестових

прикладів. Також, в додатку використовується стек Spring – Spring Web, Spring Data, Spring Actuator.

Spring Framework — це програмний фреймворк з відкритим кодом та контейнери з підтримкою інверсії управління для платформи Java.

Основні особливості Spring Framework можуть бути використані будь-яким додатком Java, але є розширення для створення веб-додатків на платформі Java EE. Незважаючи на це, Spring Framework не нав'язує якоїсь конкретної моделі програмування, Spring Framework став популярним в спільноті Java як альтернатива, або навіть доповнення моделі Enterprise JavaBean (EJB).

3.4 Демонстрація роботи додатку та взаємодії користувача з системою

Розглянемо CRUD операції для взаємодії з ресурсом Patient, котрі доступні користувачеві. Для демонстрації використовується Postman – це свого роду швейцарський ніж, який дозволяє створювати і виконувати запити, документувати й моніторити Ваші сервіси в одному місці.

3.4.1 Створення Patient

Приклад коректного запита на створення наведено на рис 3.10.

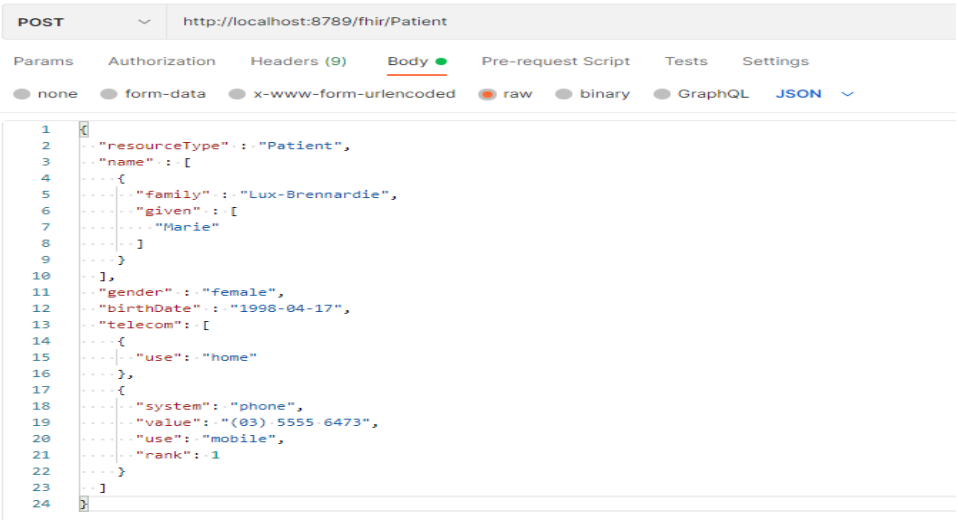


Рис 3.10 POST-запит

Відповідь на коректний запит на створення наведено на рис 3.11. Як ми можемо побачити, сутності був присвоєний унікальний ідентифікатор.

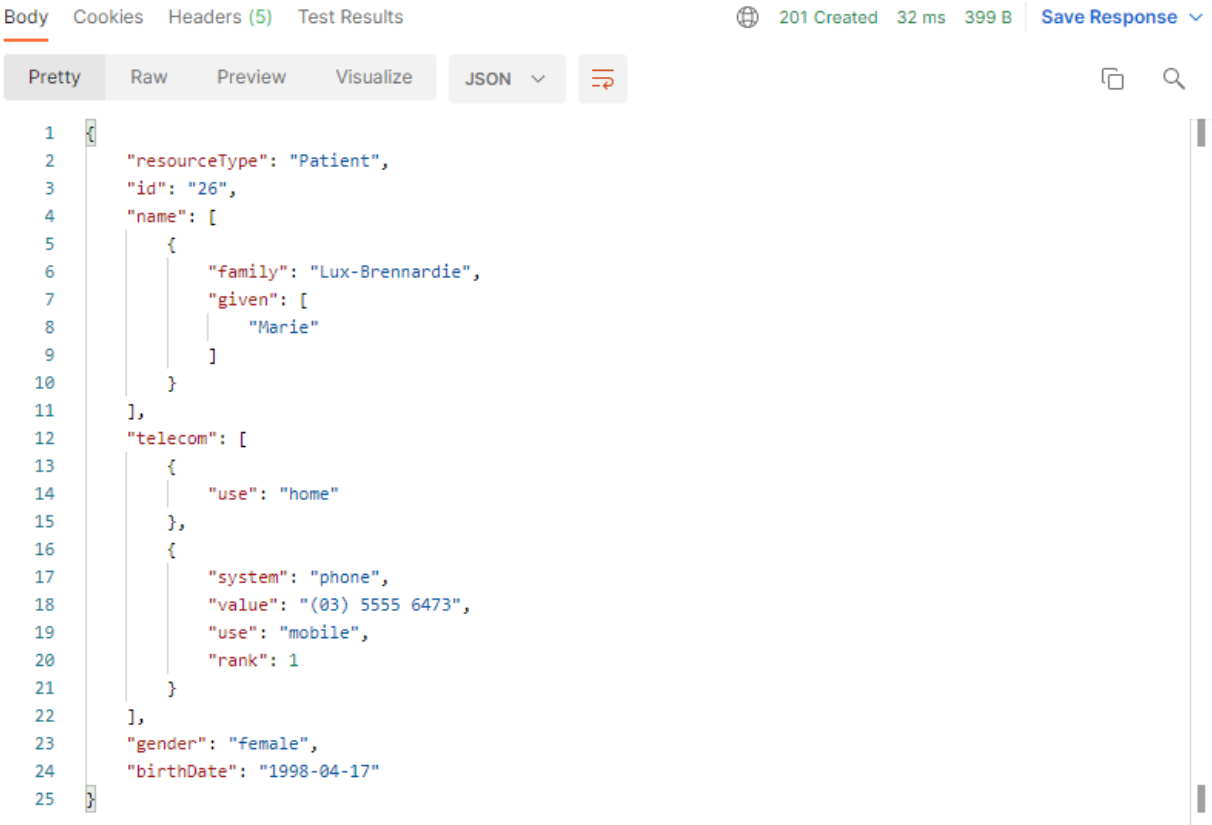


Рис 3.11 Відповідь на валідний POST-запит

Якщо ми будемо намагатись створити Patient з таким самим тілом запиту, ми отримаємо наступну відповідь (рис 3.12).

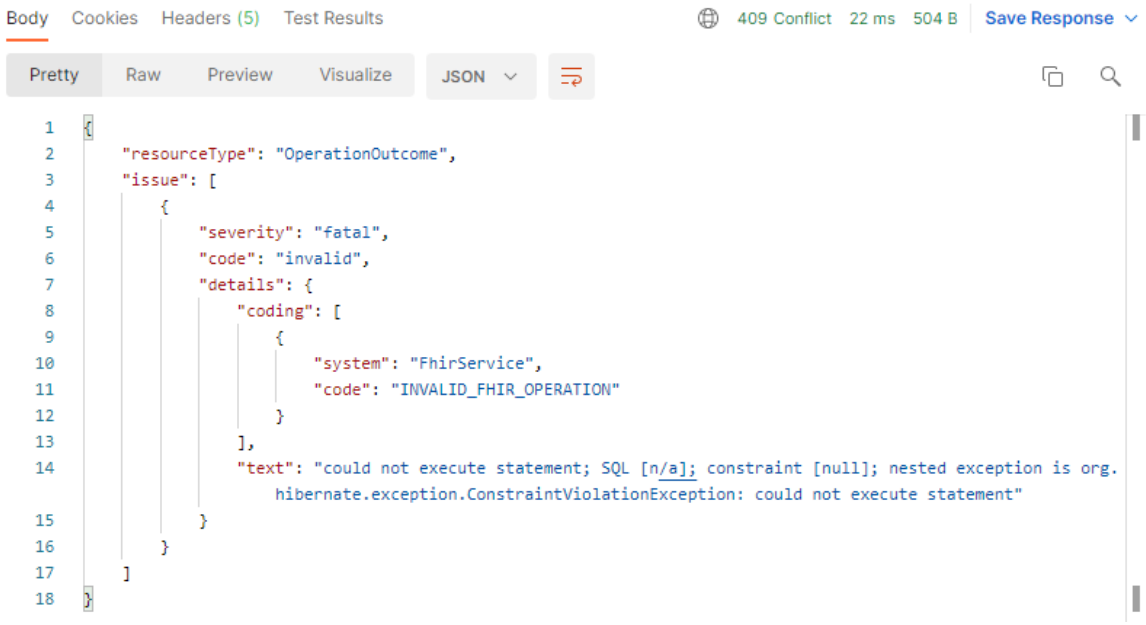


Рис 3.12 Відповідь на створення вже існуючої сутності

Якщо ми будемо намагатися створити Patient без атрибутів name, telecom і birthdate, ми отримаємо наступну відповідь (рис 3.13)

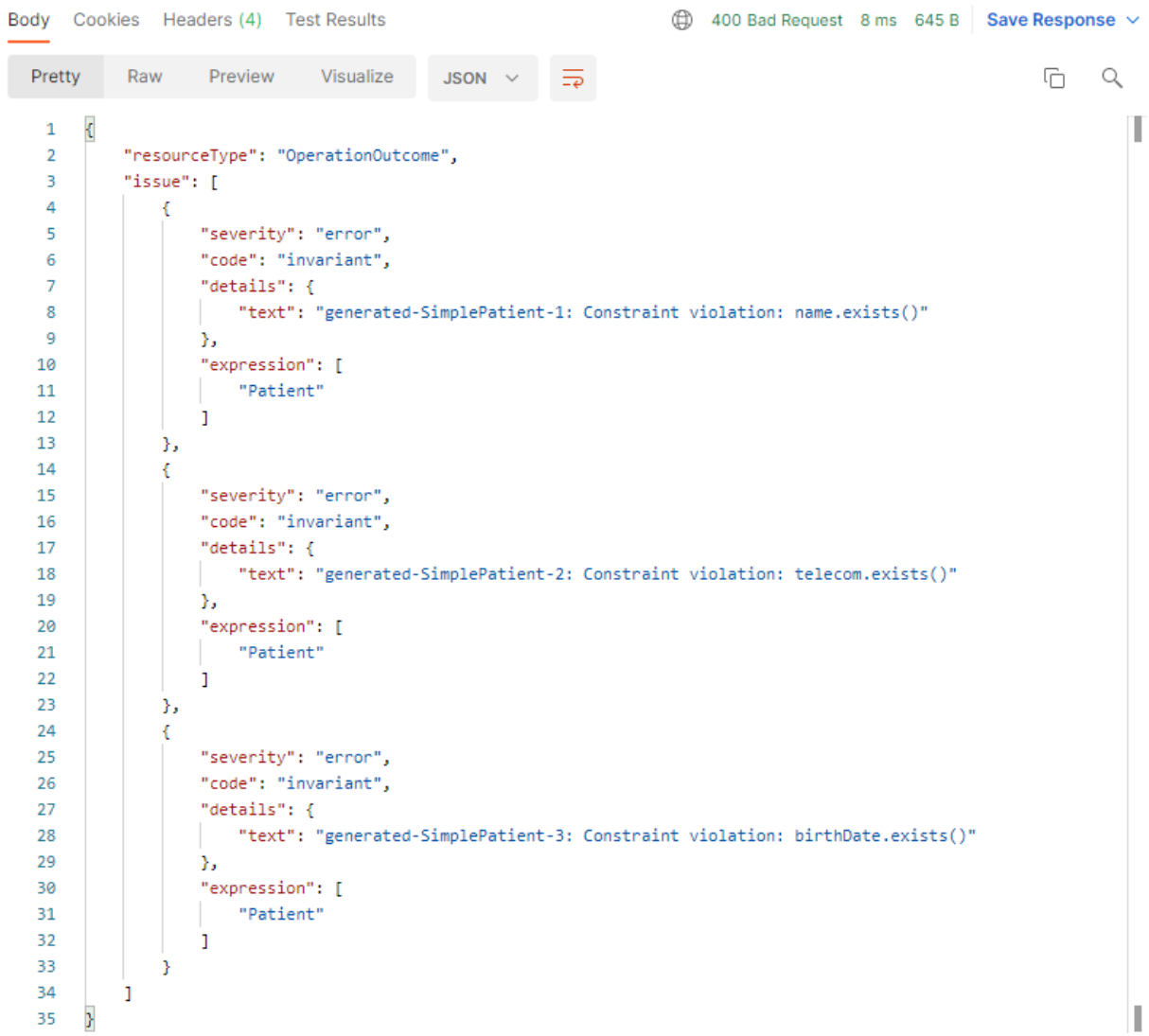


Рис 3.13 Відповідь на невалідний POST-запит

Якщо ми будемо намагатися створити Patient за неіснуючою url, то ми отримаємо наступну відповідь (рис 3.14)

Рис 3.14 Відповідь на POST-запит за невалідною url

3.4.2. Читання Patient

Запит на читання сутності і відповідь наведено на 3.15.

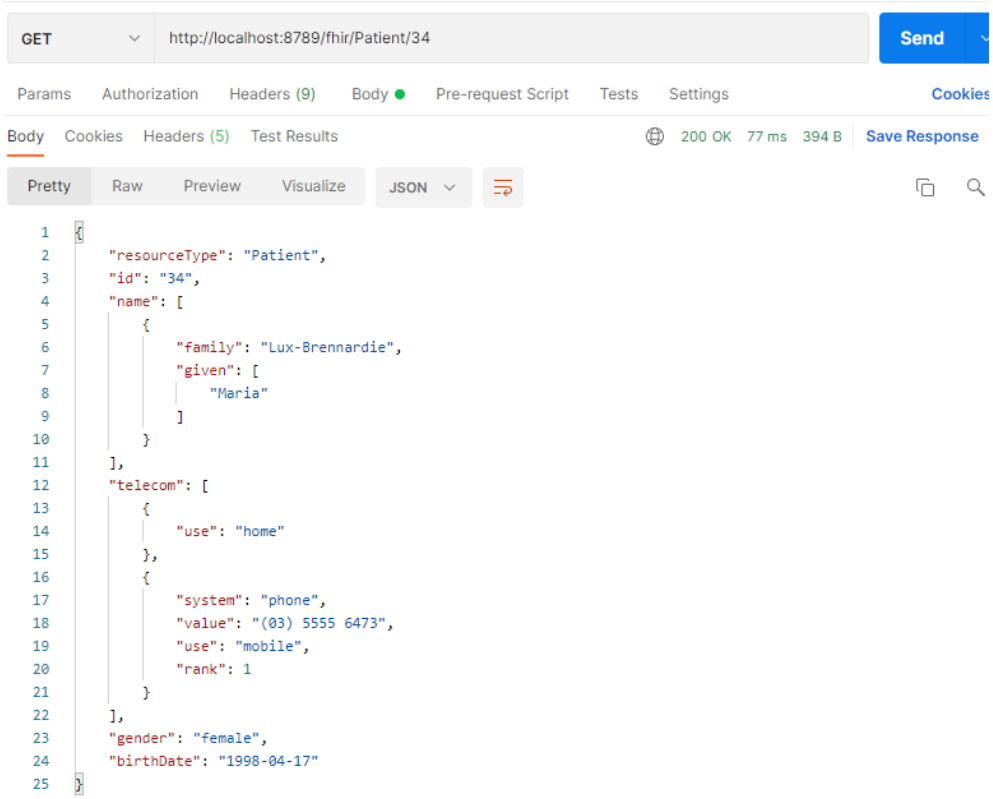


Рис 3.15 Відповідь на валідний GET-запит

Якщо ми будемо намагатися прочитати Patient за неіснуючим ідентифікатором, ми отримаємо наступну відповідь(рис 3.16)

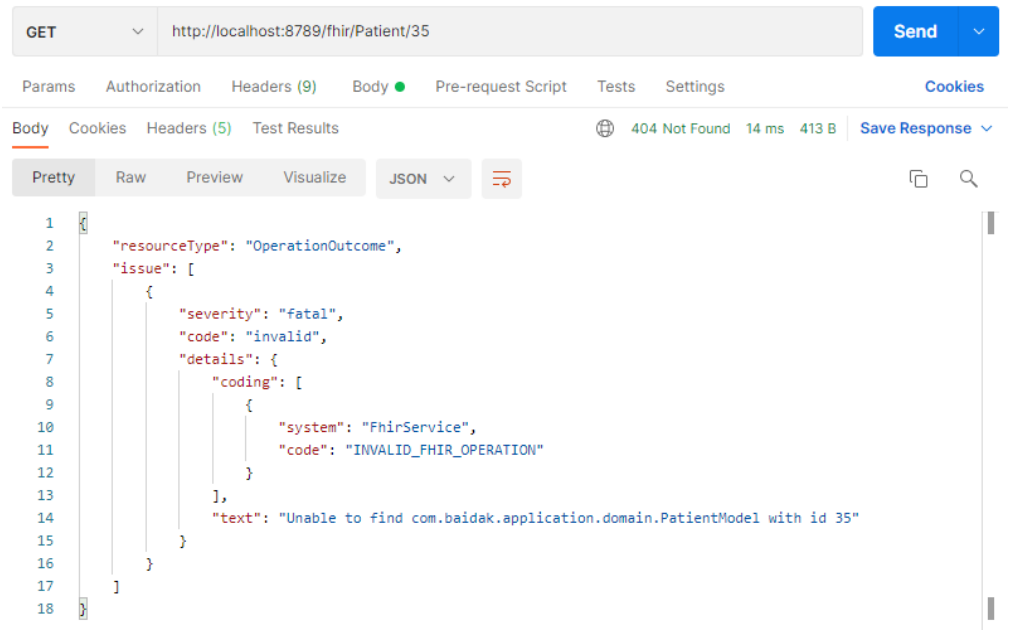


Рис 3.16 Відповідь на GET-запит за неіснуючим ідентифікатором

3.4.3. Оновлення Patient

Запит на оновлення сутності наведено на рис 3.17.

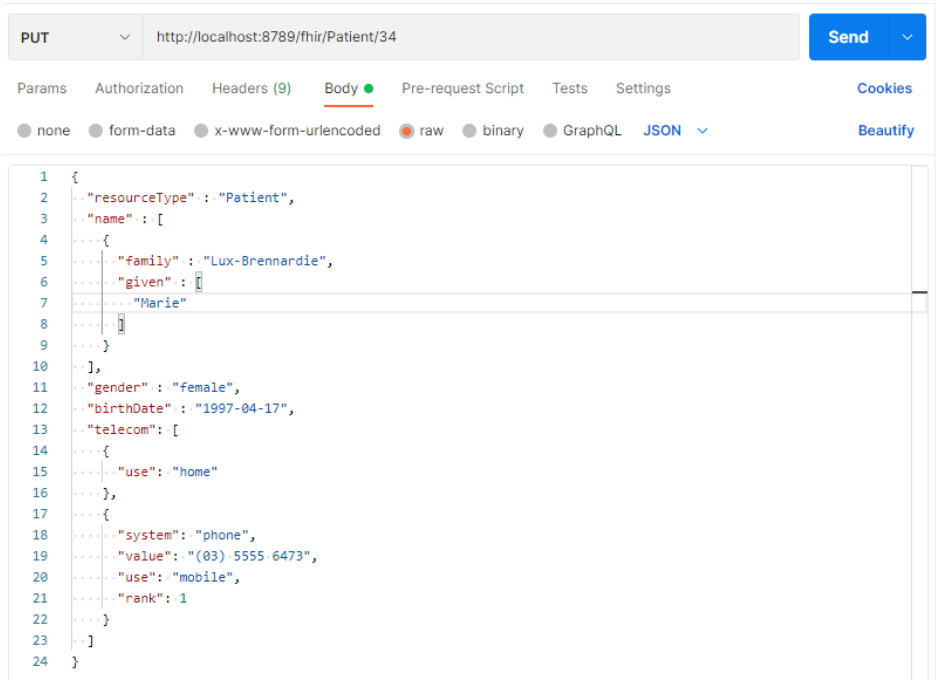


Рис 3.17 PUT-запит

Відповідь на коректний запит на оновлення наведено на рис 3.18.

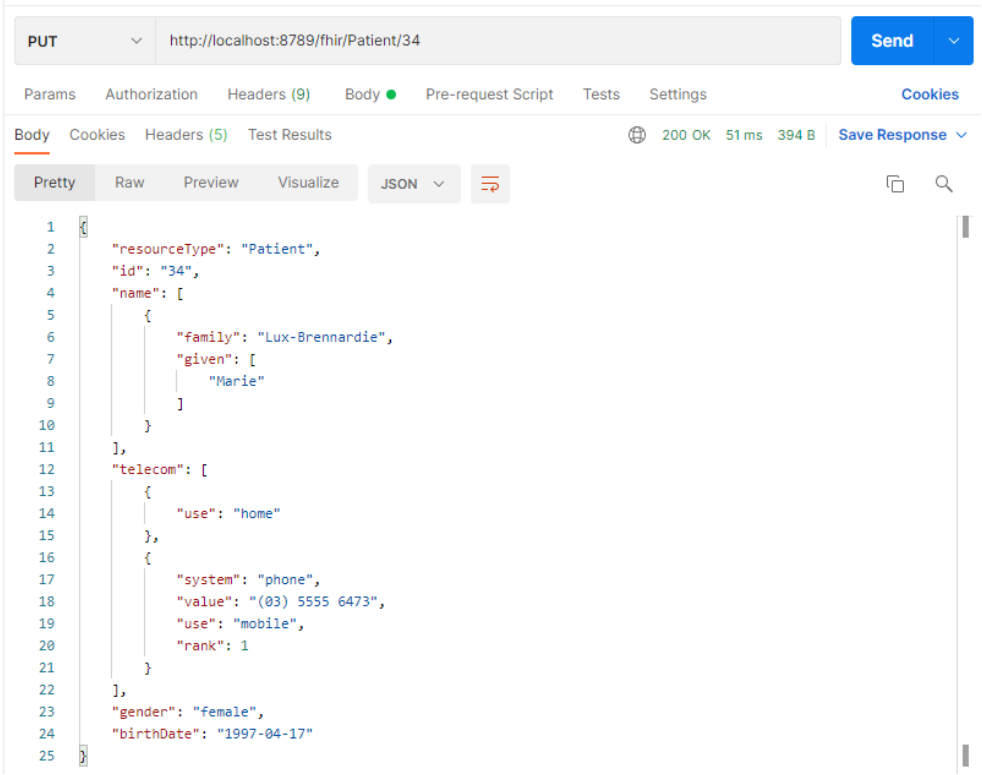


Рис 3.18 Відповідь на валідний PUT-запит

Якщо ми будемо намагатися оновити Patient за неіснуючим ідентифікатором, ми отримаємо наступну відповідь(рис 3.19)

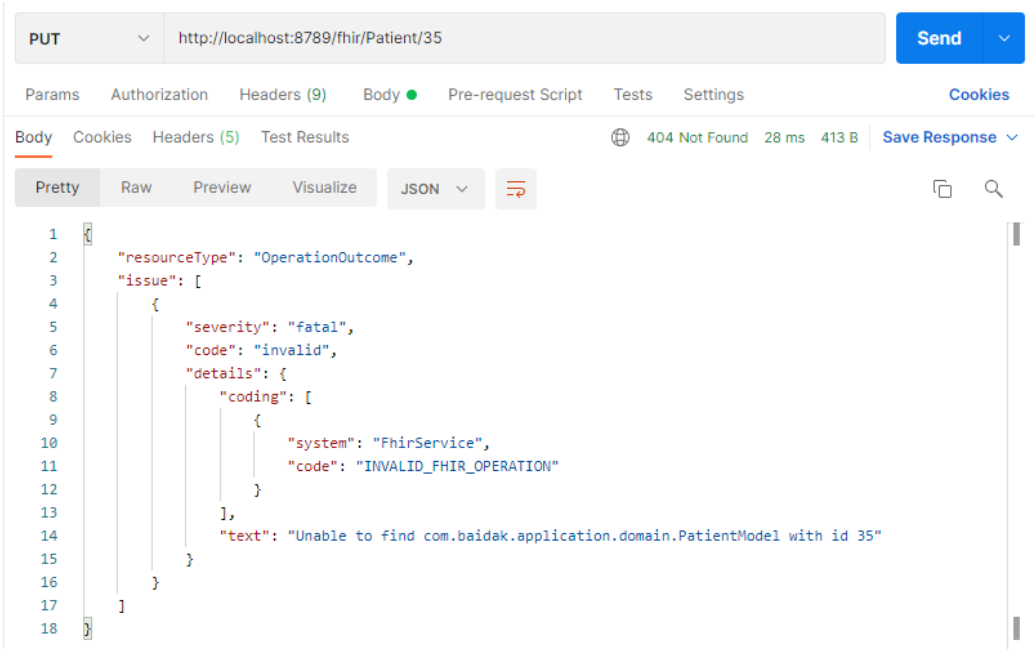


Рис 3.19 Відповідь PUT-запит за неіснуючим ідентифікатором

3.4.4. Видалення Patient

Коректний запит на видалення і відповідь на нього наведено на рис 3.20.

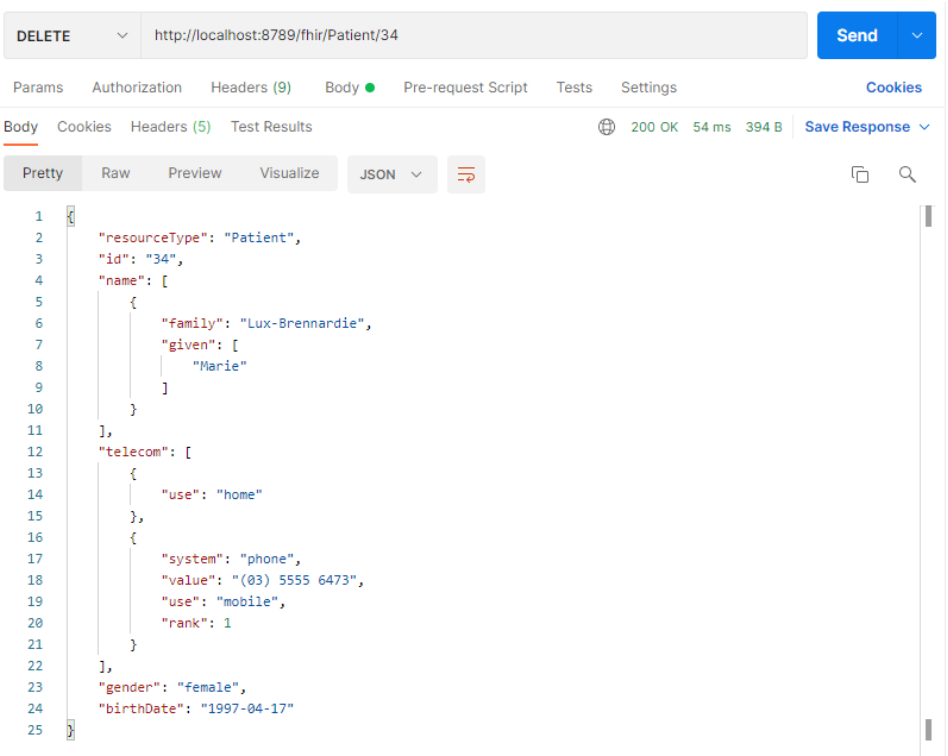


Рис 3.20 Відповідь на валідний DELETE-запит

Тепер, якщо ми будемо намагатися взаємодіяти з цим ресурсом, то завжди отримаємо наступну відповідь, хоча сам ресурс в базі даних не видалено, а лише позначено як видалений(рис. 3.21)

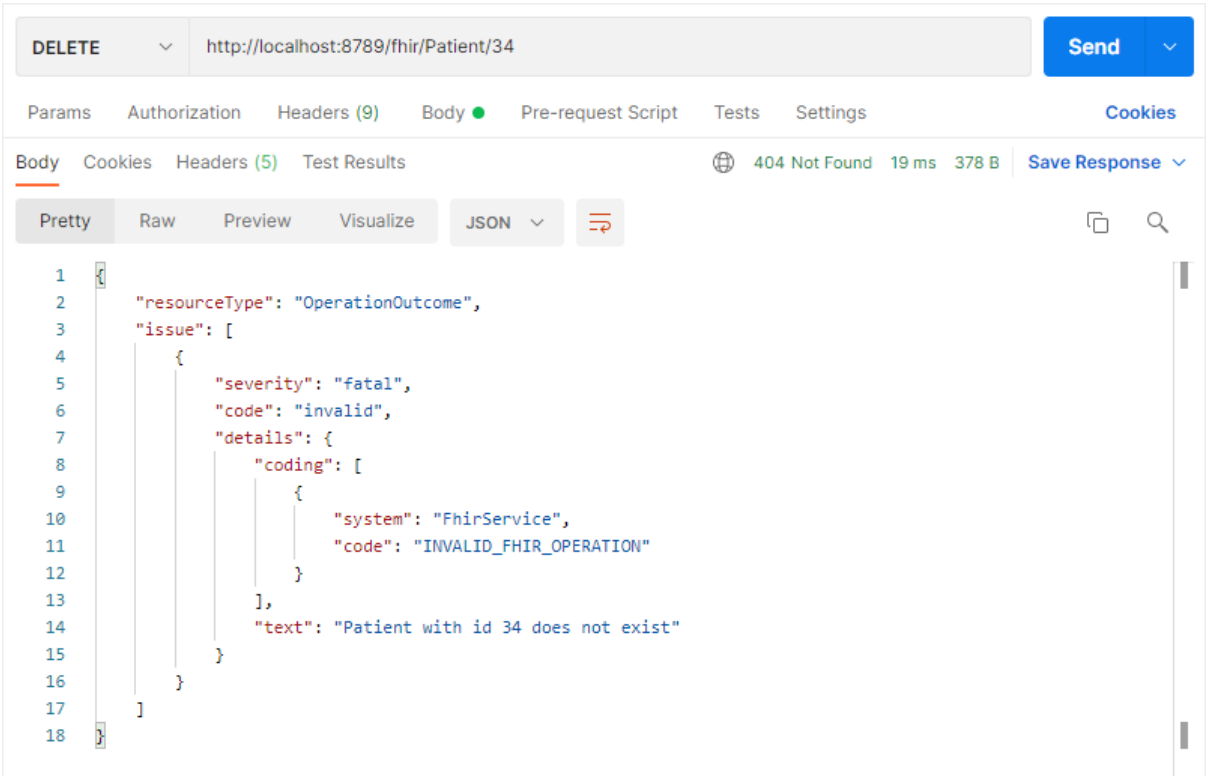


Рис 3.21 Відповідь на DELETE-запит вже видаленої сутності

Наявність ресурсу в базі даних можна побачити на рис 3.22

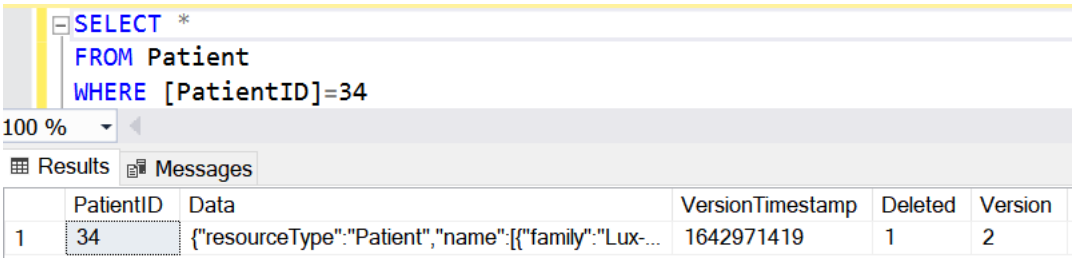


Рис 3.22 Представлення видаленого ресурсу в базі даних

Це наслідок ідемпотентності операції – властивості, яка проявляється в тому, що повторна її дія над будь-яким об'єктом уже не змінює результату. Тобто повторне виконання операцій з об'єктом не змінює результату, досягнутого при першому виконанні. І хоча ми отримали іншу відповідь при повторному DELETE запиті, сам стан ресурсу залишився незмінним.

3.4.4. Swagger документація

Всі операції, наведені вище, можна виконати за допомогою динамічної документації Swagger. Вигляд документації наведено нижче, на рис 3.23.

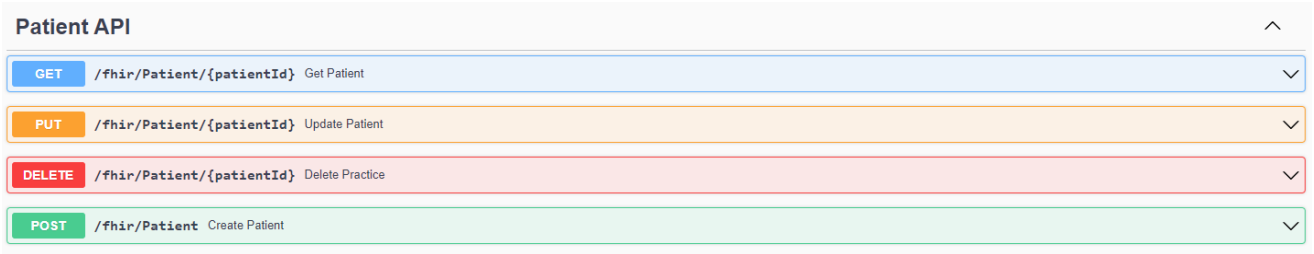


Рис 3.23 Загальний вигляд Swagger документації

3.5 CapabilityStatemet

Повноцінний FHIR-service зобов’язаний реалізовувати так званий CapabilityStatement. Власне, якщо сервіс реалізує тільки CapabilityStatement, він може вважатись FHIR-based. Це спеціальний документ, котрий дозволяє переглядати мета-інформацію сервіса та всі можливі взаємодії з усіма можливими ресурсами. В специфікації визначено, що CapabilityStatement ресурс повинен повертатись як відповідь на OPTIONS-запит на root, себто корінь веб-сервісу. Приклад цього наведено на рис 3.24.

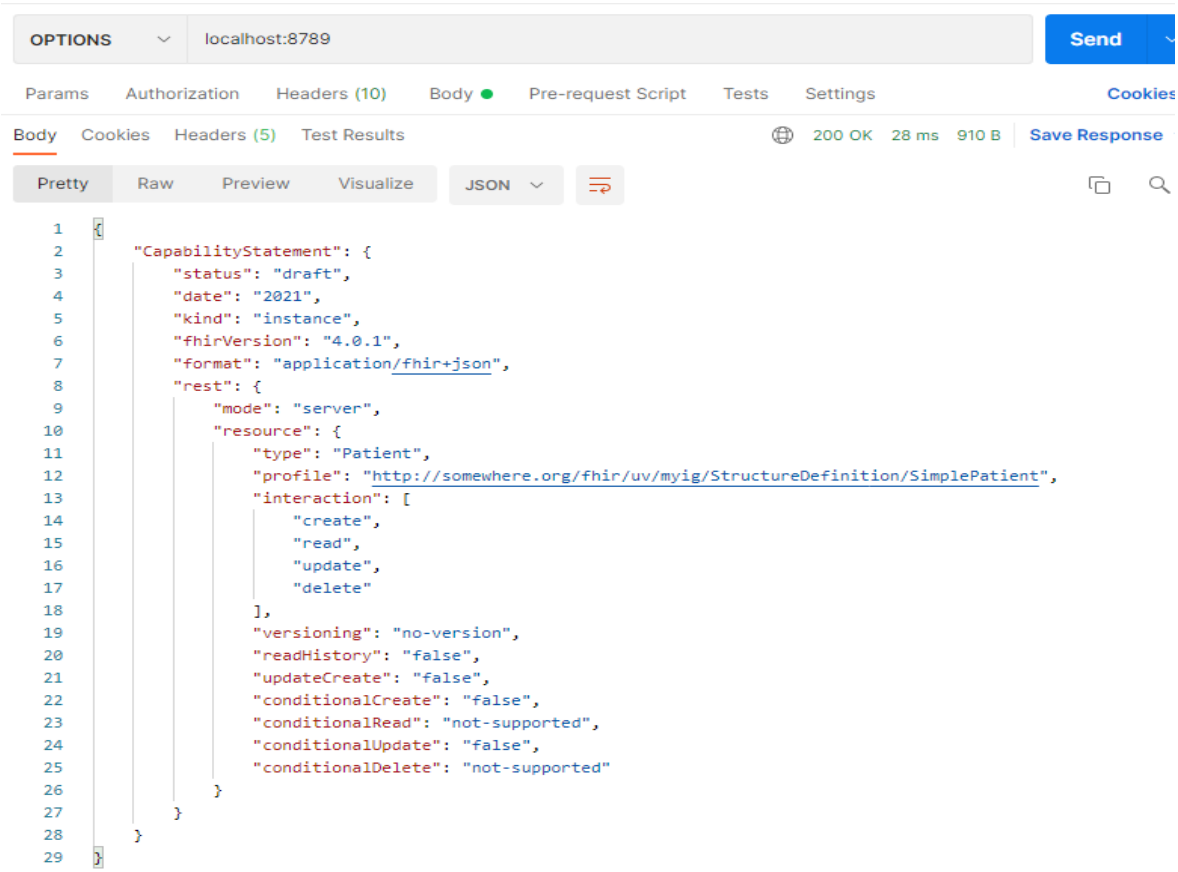


Рис 3.24 CapabilityStatement у відповідь на валідний OPTIONS-запит

3.6. Висновки до третього розділу

Метою даного розділу було розробити повноцінний веб-сервіс, котрий заснований на стандарті FHIR.

Для прикладу було обрано створення профілю, котрий би розширював стандартний ресурс Patient. Додаток написано на мові програмування Java, котра забезпечує незалежність від платформи. Спочатку було створено сам профіль, потім інтегровано його в Implementation Guide, котрий є гіперлінковим документом, який дозволяє переглядати не тільки документацію вендора, але і переходити на офіційну FHIR документацію. Сервіс має чотири ендпоінти, котрі реалізують CRUD операції над ресурсом Patient. Також було реалізовано OPTIONS ендпоінт, котрий повертає мета інформацію у форматі CapabilityStatement документу. Всі ендпоінти описано в динамічній документації Swagger.

РОЗДІЛ 4
ЕКОНОМІКА

4.1. Визначення трудомісткості розробки програмного забезпечення

Ключовим етапом при розробці програмного забезпечення є процес дослідження ринку, визначення трудомісткості та розрахунок кількості витрат на створення програмного забезпечення. В даному розділі буде проведено розрахунок витрат та економічної ефективності розробки сервісу на основі стандарту FHIR.

Вхідні дані:

- плата за годину роботи інженера-програміста, грн / год — 150;
- вартість машино-години ЕОМ, грн / год — 50.

Розрахування витрат на розробку програмного забезпеченням вкрай ускладняється динамічністю бізнеса, котрий може генерувати нові вимоги навіть щодня, тому трудомісткість розробки ПЗ може бути розрахована на основі системних моделей з різною точністю оцінки.

Трудомісткість розраховується за формулою:

$$t = t_u + t_a + t_n + t_{\text{відл}} + t_{\text{д}},$$

де t_o – витрати праці на підготовку і опис поставленого завдання (50 год.);

t_u – витрати праці на дослідження алгоритму вирішення задач;

t_a – витрати праці на розробку блок-схем алгоритму;

t_n – витрати праці на програмування за готовим блок-схемою;

$t_{відл}$ – витрати праці на відладку програм на ПЕОМ;

$t_д$ – витрати праці на підготовку документації.

Загальна трудомісткість також залежить від кількості співробітників проекту. Почнемо з розрахунку витрат праці, котрі очікуються:

$$Q = q * C * (1 + p)$$, людино-годин,

де q – передбачуване число операторів;

C – коефіцієнт складності програми;

p – коефіцієнт корекції програми в ході її розробки.

В нашому випадку, q буде дорівнювати 1100, C – 1,25 , p – 0,15.

$$Q = 1100 * 1.25 * (1 + 0.15) = 1581$$, людино-години

Витрати праці на вивчення опису завдання визначаються з урахуванням уточнення опису і кваліфікації програміста за формулою:

$$t_u = \frac{QB}{(75 \dots 85)K}$$

де B – коефіцієнт збільшення витрат праці (внаслідок неповного опису завдання, B = 1,2…1,5);

K – Коефіцієнт кваліфікації програміста, який визначається в залежності від стажу роботи за фахом (якщо стаж роботи менший 2 років, то K = 1,2).

Витрати праці на дослідження алгоритму рішення задачі:

$$t_u = 1581 * 1.5 \,/\, (82 * 1.2) \; = 24.1$$, людино-години

Витрати на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20 \dots 25)K}$$

$$t_a = 1581 / \; (22 * 1.2) \; = 60$$, людино-години

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20 \dots 25)K}$$

$$t_n = 1581 / \; (24 * 1.2) \; = 55$$, людино-години

Витрати праці на налагодження програми на ЕОМ:

$$t_{\text{відл}} = \frac{Q}{(4 \dots 5)K}$$

$$t_{\text{відл}} = 1581/\ (5 * 1.2)\ = 263.5, \text{людино-години}$$

Витрати праці на підготовку документації:

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}}, \text{людино-годин}$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису.

$$t_{\text{др}} = \frac{Q}{(15 \dots 20)K}$$

$$t_{\text{др}} = 1581/\ (15 * 1.2)\ = 87.8, \text{людино-години}$$

$t_{до}$ – трудомісткість редагування, друкування й оформлення документації

$$t_{\text{до}} = 0.75 * t_{\text{др}}, \text{людино-годин}$$
$$t_{\text{до}} = 0.75 * 87.8 = 65.9, \text{людино-годин}$$
$$t_{\text{д}} = 87.8 + 65.9 = 153.65, \text{людино-годин}$$

Отримуємо трудомісткість розробки програмного забезпечення:

$$t = 24.1 + 60 + 55 + 78.2 + 263.5 + 153.65 = 634.45, \text{людино-години}$$

4.2. Розрахунок витрат на створення програмного забезпечення

Витрати на створення програмного забезпечення $К_{пз}$ включають витрати на заробітну плату розробників програми ($З_{зп}$) і витрати машинного часу, необхідного для налагодження програми на ЕОМ ($З_{мв}$).

$$K_{\text{по}} = З_{\text{зп}} + З_{\text{мв}}$$

$$З_{зп} = t * С_{пр}$$

де *t* – загальна трудомісткість розробки ПЗ;

$C_{пр}$ – середня годинна заробітна плата програміста

$$З_{зп} = 634.45 * 150 = 95\ 168 \text{ грн.}$$

Витрати машинного часу, необхідного для налагодження програми на ЕОМ ($З_{мв}$) визначаються за формулою:

$$З_{мв} = t_{відл} * С_{мч}$$

де $t_{відл}$ – трудомісткість налагодження програми на ЕОМ;

$C_{мч}$ – вартість машино-години ЕОМ.

$$З_{мв} = 263.5 * 50 = 13175 \text{ грн.}$$

Таким чином витрати на створення програмного забезпечення, складуть:

$$К_{по} = 95168 + 13175 = 108343 \text{ грн.}$$

Очікувана тривалість розробки ПО:

$$T = \frac{t}{B_k * F_p}$$

де $B_к$ – число розробників;

$F_р$– місячний фонд робочого часу (при 40-ка годинному робочому тижні $F_р$ = 176 годин).

$$t_{др} = 634.45 / 1 * 176 = 3.6 \text{ місяців}$$

4.3. Маркетингові дослідження

Основною цільовою аудиторією додатку є привітні клініки та державні установи охорони здоров’я. Додаток, написаний за стандартом FHIR, має значно меншу вартість розробки завдяки інтероперабельності і стандартизації, що має зменшити кількість годин, витрачених на підтримку в майбутньому.

Оскільки додаток написано на мові програмування Java, він є мультиплатформним, а отже не буде вимагати від покупця спеціальної операційної системи на сервері.

4.4. Економічна ефективність

Економічний ефект від впровадження програмного забезпечення для обміну та валідацією медичними документами за стандартом FHIR очікується позитивним, адже потребує значно меншої кваліфікації від розробника, ніж інші стандарти. Це зумовлено тим, що стандарт легко вивчати, адже він має об’ємну документацію. Також, такий сервіс буде легко підтримувати в майбутньому, що є вкрай важливим аспектом при вираховуванні економічної ефективності.

ВИСНОВКИ

В процесі дослідження були отримані наступні результати. При вивчені принципів побудови додатку для валідації та обміну медичними документами, було прийнято рішення розділити проектування сервісу на декілька частин, а саме вирішення проблеми з підтримкою різних платформ, розробка архітектури додатку, його прикладного програмного інтерфейсу, розробка та документація правил валідації документів, та власне кажучи, розробка самого додатку.

Для вирішень проблем з підтримкою багатьох операційних систем було розроблено додаток з використанням багатоплатформової мови програмування Java. Такий підхід дозволяє додатку бути скомпільованим для будь-якої операційної системи, а отже позбутися платформної залежності. І як наслідок, реалізатору не потрібно розробляти окрему версію для кожної потенційної операційної системи.

Додаток має серверну архітектуру, а для спілкування з іншими системами надає публічний API. Прикладний програмний інтерфейс розроблено на основі REST-підходу, тому взаємодіяти з додатком можна за допомогою стандартних HTTP-запитів.

Додаток засновано на одному з найпопулярніших стандартів в царині програмного забезпечення для медичного обслуговування – FHIR.

FHIR є найновішим стандартом, розробленим HL7 – одним із головних новаторів та впроваджувачем трендів в сфері медичного обслуговування в США. Також, FHIR є міжнаціональним стандартом, а отже його можна буде використовувати для запитів щодо пацієнта в інші країни, якщо він потребує медичної допомоги не у себе на батьківщині. Плюсами цього стандарту є те, що він є абсолютно безкоштовним та забезпечує швидкий і динамічний процес розробки.

Задля демонстрації роботи такого додатку було розроблено профіль для розширення ресурсу Patient.

Перш за все, було розроблено сам профіль, котрий накладає деякі обмеження та зобов’язання на стандартний ресурс.

Після дослідження доступних технологій та розробки архітектури додатку, було створено сам додаток. В якості фреймворку для мови Java було використано Spring. Цей фреймворк де-факто є стандартом в розробці програмного забезпечення, написаного на Java, для великого бізнесу, ентерпрайзу. Сам додаток побудовано на технології сервлетів, котрі хостяться всередині контейнера сервлетів Apache Tomcat. У якості бази даних було використано реляційну базу даних, Microsoft SQL Server.

Протягом останніх трьох років, ми можемо спостерігати надзвичайно важкий стан системи охорони здоров’я, причому не тільки Української, але й загалом світової. Криза, спричинена Covid-19 продовжує набирати оберти, і далеко не всі заклади охорони здоров’я витримують ці випробовування. Доступ до інформації користувача для самого користувача обмежено, а часті перебої та іноді навіть неможливість отримати сертифікат про вакцинацію свідчать про застарілість та недосконаліть програмного забезпечення в царині охорони здоров’я.

В Україні 2021 рік пройшов під егідою «інформаційного майбутнього», і можна зробити висновок, що покращення програмного забезпечення в сфері охорони здоров’я, а відтак і сфери охорони здоров’я в цілому є необхідним і невідкладним кроком для досягнення цього «інформаційного майбутнього».

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wiadomości Lekarskie, VOLUME LXXIV, ISSUE 5, MAY 2021, HEALTHCARE IN UKRAINE DURING THE EPIDEMIC: DIFFICULTIES, CHALLENGES AND SOLUTIONS
2. Health Informatics on FHIR: How HL7's New API is Transforming Healthcare, 2018. – 263 p.
3. https://habr.com/ru/post/139904/
4. https://hl7.org.ua/standards/
5. https://www.hl7.org/fhir/comparison.html
6. https://www.healthjump.com/blog/5-challenges-with-healthcare-interoperability
7. Albers V., Still B. (Eds.) Usability of Complex Information Systems: Evaluation of User Interaction. – CRC Press, 2011. – 392 p.
8. Cay S. Horstmann Core Java Volume I – Fundamentals
9. Joshua Bloch, Effective Java, 3 edition
10. Cay S. Horstmann and Gary Cornell , Core Java
11. Dr. Heinz Kabutz, Extreme Java - Concurrency Performance for Java 8
12. Paul J. Deitel and Harvey M. Deitel, Java: How to Program: Early Objects
13. Alan Mycroft and Mario Fusco, Java 8 in Action
14. David Flanagan , Java in a Nutshell: A Desktop Quick Reference
15. Jamie Chan, Learn Java in One Day and Learn It Well
16. Bill Joy, Gilad Bracha, Guy L. Steele Jr., and James Gosling, The Java Language Specification
17. Allen B. Downey, Think Java: How to think like a Computer Scientist
18. Bruce Eckel, Thinking in Java
19. Craig Walls, Spring in Action
20. Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship

21. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, third edition
22. Robert C. Martin, The Clean Coder: A Code of Conduct for Professional Programmers
23. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Grady Booch (Foreword), Design Patterns: Elements of Reusable Object-Oriented Software
24. https://www.hl7.org/fhir/
25. https://auc.org.ua/novyna/amu-problemy-v-ohoroni-zdorovya-potrebuyut-pershochergovogo-vyrishennya
26. https://www.oecd-ilibrary.org/sites/85e4b6a1-en/index.html?itemId=%2Fcontent%2Fcomponent%2F85e4b6a1-en&fbclid=IwAR2TvB37WrRUi5pZnllLmpc5P44fOjUTqTOjd1j-t1HBUttLdRIQyr6WjsI#section-d1e7825
27. http://neweurope.org.ua/media-post/koronavirus-i-yednist-u-yes-vchytys-na-pomylkah-derzhav-maye-i-ukrayina/
28. https://www.frontiersin.org/articles/10.3389/fpubh.2021.647315/full
29. https://www.health-samurai.io/articles/fhir-what-is-great-what-isnt-so-good-and-what-it-is-not#:~:text=FHIR%20solves%20these%20problems%20by,need%20to%20build%20applications%20fast.
30. https://digitalhealth.folio3.com/blog/hl7-vs-fhir/
31. https://blog.cloudticity.com/hl7-vs-fhir-key-differences-healthcare-data-exchange
32. https://prolifics.com/history-of-fhir/
33. https://en.wikipedia.org/wiki/Fast_Healthcare_Interoperability_Resources
34. https://www.netsolutions.com/insights/5-healthcare-problems-which-digital-technologies-can-solve-for-a-fit-and-healthy-world/
35. https://habr.com/ru/post/425995/

36. https://help.salesforce.com/s/articleView?language=ru&type=5&id=sf.admin_clinical_data_model_fhir.htm

37. https://docs.oracle.com/en/java/

38. https://docs.spring.io/spring-framework/docs/current/reference/html/

39. https://idratherbewriting.com/learnapidoc/#:~:text=%E2%80%9CREST%E2%80%9D%20stands%20for%20Representational%20State%20Transfer.&text=In%20your%20REST%20API%20documentation,sample%20responses%20from%20the%20endpoints.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

Liquibase код для створення таблиці Patient

src/main/resources/db-changelog/changelog-master.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog
        xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
        http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.8.xsd">

    <include file="changesets/Patient-001.xml" relativeToChangelogFile="true"/>

</databaseChangeLog>
```

src/main/resources/db-changelog/changesets/Patient-001.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog
        xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
        http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.8.xsd">

    <changeSet id="1" author="Leonid.Baidak">
        <createTable tableName="Patient">
            <column name="PatientID" type="BIGINT" autoIncrement="true">
                <constraints nullable="false"/>
            </column>
            <column name="Data" type="VARCHAR(2048)">
                <constraints nullable="false"/>
            </column>
            <column name="VersionTimestamp" type="BIGINT">
                <constraints nullable="false"/>
            </column>
            <column name="Deleted" type="BIT">
                <constraints nullable="false"/>
            </column>
            <column name="Version" type="INT">
                <constraints nullable="false"/>
            </column>
            <column name="Created" type="DATETIME">
                <constraints nullable="true"/>
            </column>
            <column name="CreatedBy" type="VARCHAR(50)">
                <constraints nullable="true"/>
```

```
        </column>
        <column name="CreatedByHost" type="VARCHAR(50)">
            <constraints nullable="true"/>
        </column>
        <column name="LastModified" type="DATETIME">
            <constraints nullable="true"/>
        </column>
        <column name="LastModifiedBy" type="VARCHAR(50)">
            <constraints nullable="true"/>
        </column>
        <column name="LastModifiedByHost" type="VARCHAR(50)">
            <constraints nullable="true"/>
        </column>
    </createTable>
</changeSet>

<changeSet id="2" author="Leonid.Baidak">
    <addPrimaryKey
            tableName="Patient"
            columnNames="PatientID"
            constraintName="PK_Patient"
            clustered="true"/>
</changeSet>

<changeSet id="3" author="Leonid.Baidak">
    <sql dbms="mssql">
        <comment>Audit trigger for Patient</comment>
        CREATE TRIGGER dbo.Patient_Audit ON dbo.Patient
            AFTER INSERT, UPDATE AS
        BEGIN
            SET NOCOUNT ON
                IF EXISTS (SELECT * FROM Deleted)
        UPDATE Patient
        SET Created = Deleted.Created,
            CreatedBy = Deleted.CreatedBy,
            CreatedByHost = Deleted.CreatedByHost,
            LastModified = SYSUTCDATETIME(),
            LastModifiedBy = SYSTEM_USER,
            LastModifiedByHost = HOST_NAME()
            FROM dbo.Patient
                        JOIN Deleted
        ON (Patient.PatientID=Deleted.PatientID)
            ELSE
        UPDATE Patient
        SET Created = SYSUTCDATETIME(),
            CreatedBy = SYSTEM_USER,
            CreatedByHost = HOST_NAME(),
            LastModified = SYSUTCDATETIME(),
            LastModifiedBy = SYSTEM_USER,
            LastModifiedByHost = HOST_NAME()
            FROM dbo.Patient
                    JOIN Inserted
        ON (Patient.PatientID=Inserted.PatientID)
```

```
                SET NOCOUNT OFF
            END
        </sql>
        <rollback>
            <sql>
                DROP TRIGGER dbo.Patient_Audit
            </sql>
        </rollback>
    </changeSet>

    <changeSet id="4" author="Leonid.Baidak">
        <addUniqueConstraint
                tableName="Patient"
                columnNames="Data"
                constraintName="UQ_Patient_Data"
        />
    </changeSet>
</databaseChangeLog>
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.4.5</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.baidak</groupId>
    <artifactId>application</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
        <revision>-local</revision>
        <buildCounter>local</buildCounter>
        <buildTimestamp>local</buildTimestamp>
        <commitHash>local</commitHash>
        <branch>local</branch>
        <jacoco.version>0.8.7</jacoco.version>
        <properties.maven.plugin.version>1.0.0</properties.maven.plugin.version>
        <mockito.version>3.11.2</mockito.version>
        <fhir.version>4.0.1</fhir.version>
        <fhir.ibm.version>4.5.5</fhir.ibm.version>
        <org.hl7.fhir.publisher.version>1.1.34</org.hl7.fhir.publisher.version>
```

```
    <liquibase.maven.plugin.version>3.8.0</liquibase.maven.plugin.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-configuration-processor</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
        <groupId>com.microsoft.sqlserver</groupId>
        <artifactId>mssql-jdbc</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>log4j-over-slf4j</artifactId>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>jcl-over-slf4j</artifactId>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>jul-to-slf4j</artifactId>
    </dependency>
    <dependency>
```

```
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-ui</artifactId>
    <version>1.6.4</version>
</dependency>
<dependency>
    <groupId>org.dom4j</groupId>
    <artifactId>dom4j</artifactId>
    <version>2.1.3</version>
</dependency>
<dependency>
    <groupId>jakarta.xml.bind</groupId>
    <artifactId>jakarta.xml.bind-api</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
        <exclusion>
            <groupId>org.junit.vintage</groupId>
            <artifactId>junit-vintage-engine</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
</dependency>
<dependency>
    <groupId>ch.qos.logback.contrib</groupId>
    <artifactId>logback-json-classic</artifactId>
    <version>0.1.5</version>
</dependency>
<dependency>
    <groupId>ch.qos.logback.contrib</groupId>
    <artifactId>logback-jackson</artifactId>
    <version>0.1.5</version>
</dependency>
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-inline</artifactId>
    <version>${mockito.version}</version>
    <scope>test</scope>
</dependency>

<!--FHIR Dependencies-->
<dependency>
    <groupId>com.ibm.fhir</groupId>
    <artifactId>fhir-validation</artifactId>
```

```
            <version>${fhir.ibm.version}</version>
        </dependency>
        <dependency>
            <groupId>org.hl7.fhir.publisher</groupId>
            <artifactId>org.hl7.fhir.publisher.cli</artifactId>
            <version>${org.hl7.fhir.publisher.version}</version>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>ca.uhn.hapi.fhir</groupId>
            <artifactId>org.hl7.fhir.validation.cli</artifactId>
            <version>5.0.0</version>
            <optional>true</optional>
            <exclusions>
                <exclusion>
                    <groupId>ca.uhn.hapi.fhir</groupId>
                    <artifactId>org.hl7.fhir.r5</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>ca.uhn.hapi.fhir</groupId>
                    <artifactId>org.hl7.fhir.dstu2</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>ca.uhn.hapi.fhir</groupId>
                    <artifactId>org.hl7.fhir.dstu2016may</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>ca.uhn.hapi.fhir</groupId>
                    <artifactId>org.hl7.fhir.dstu3</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>

    <build>
        <resources>
            <resource>
                <directory>src/main/resources</directory>
                <filtering>true</filtering>
            </resource>
        </resources>

        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <configuration>
                    <failOnMissingWebXml>false</failOnMissingWebXml>
                    <archive>
                        <manifest>
                            <addDefaultImplementationEntries>true</addDefaultImplementationEntries>
                        </manifest>
                        <manifestEntries>
```

```
                            <buildCounter>${buildCounter}</buildCounter>
                            <buildTimestamp>${buildTimestamp}</buildTimestamp>
                            <commitHash>${commitHash}</commitHash>
                            <branch>${branch}</branch>

<generatedVersionRevisionSuffix>${revision}</generatedVersionRevisionSuffix>
                        </manifestEntries>
                    </archive>
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.liquibase</groupId>
                <artifactId>liquibase-maven-plugin</artifactId>
                <version>${liquibase.maven.plugin.version}</version>
                <configuration>
                    <changeLogFile>src/main/resources/db-changelog/changelog-
master.xml</changeLogFile>
                    <driver>com.microsoft.sqlserver.jdbc.SQLServerDriver</driver>
                    <propertyFileWillOverride>false</propertyFileWillOverride>
                    <promptOnNonLocalDatabase>false</promptOnNonLocalDatabase>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

src/main/resources/fhir/CapabilityStatement.json

```
{
  "CapabilityStatement":{
    "status":"draft",
    "date":"2021",
    "kind":"instance",
    "fhirVersion":"4.0.1",
    "format":"application/fhir+json",
    "rest":{
      "mode":"server",
      "resource":{
        "type":"Patient",
        "profile":"http://somewhere.org/fhir/uv/myig/StructureDefinition/SimplePatient",
        "interaction":[
          "create",
          "read",
          "update",
          "delete"
        ],
        "versioning":"no-version",
        "readHistory":"false",
        "updateCreate":"false",
        "conditionalCreate":"false",
        "conditionalRead":"not-supported",
        "conditionalUpdate":"false",
        "conditionalDelete":"not-supported"
      }
```

```
    }
  }
}
```

src/main/resources/application-local.properties

```
server.port=8789

management.endpoints.enabled-by-default=false
management.endpoints.web.exposure.include=prometheus, metrics, health
management.endpoint.health.enabled=true
management.endpoint.metrics.enabled=true
management.endpoint.prometheus.enabled=true

spring.datasource.driver-class-name=com.microsoft.sqlserver.jdbc.SQLServerDriver
spring.jpa.hibernate.naming.physical-
strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl

spring.datasource.url=jdbc:sqlserver://localhost\\BAIDAKMSSQL;databaseName=FhirDatabase;sendStringPa
rametersAsUnicode=false;
spring.datasource.username=sa
spring.datasource.password=123456aD$

springdoc.swagger-ui.default-models-expand-depth=-1
springdoc.override-with-generic-response=false
springdoc.cache.disabled=true
springdoc.swagger-ui.display-request-duration=true
springdoc.swagger-ui.path=/swagger-client.html
```

logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration scan="false">
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} %highlight(%5p) %cyan(%c:%L)
- %green(%mdc) %msg%n</pattern>
        </encoder>
    </appender>

    <root>
        <level value="INFO"/>
        <appender-ref ref="CONSOLE"/>
    </root>

    <logger name="com.baidak" level="DEBUG"/>
    <logger name="org.hibernate.engine.internal.StatisticalLoggingSessionEventListener"
level="WARN"/>
</configuration>
```

src/main/resources/fhir/structure_definition/package/StructureDefinition-SimplePatient.json

```
{
  "resourceType": "StructureDefinition",
```

```
  "id": "SimplePatient",
  "url": "http://somewhere.org/fhir/uv/myig/StructureDefinition/SimplePatient",
  "version": "1",
  "name": "SimplePatient",
  "title": "Baidak Leonid Simple Patient Profile",
  "status": "draft",
  "fhirVersion": "4.0.1",
  "kind": "resource",
  "abstract": false,
  "type": "Patient",
  "baseDefinition": "http://hl7.org/fhir/StructureDefinition/Patient",
  "derivation": "constraint",
  "description": "This is a special profile, that has been developed by Leonid Baidak during master
degree investigations",
  "differential": {
    "element": [
      {
        "id": "Patient.name",
        "path": "Patient.name",
        "short": "Baidak master diploma SimplePatient name",
        "definition": "Baidak master diploma SimplePatient name",
        "min": 1
      },
      {
        "id": "Patient.name.family",
        "path": "Patient.name.family",
        "maxLength": 40
      },
      {
        "id": "Patient.name.given",
        "path": "Patient.name.given",
        "maxLength": 45
      },
      {
        "id": "Patient.telecom",
        "path": "Patient.telecom",
        "min": 1,
        "max": "3"
      },
      {
        "id": "Patient.birthDate",
        "path": "Patient.birthDate",
        "min": 1,
        "max": "1"
      },
      {
        "id": "Patient.photo",
        "path": "Patient.photo",
        "max": "0"
      }
    ]
  }
}
```


src/main/java/com/baidak/application/ServletInitializer.java

```
package com.baidak.application;

import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

public class ServletInitializer extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(Application.class);
    }
}
```

src/main/java/com/baidak/application/Application.java

```
package com.baidak.application;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

src/main/java/com/baidak/application/domain/PatientModel.java

```
package com.baidak.application.domain;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.NoArgsConstructor;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Version;

import static javax.persistence.GenerationType.IDENTITY;

@Data
@Table(name = "Patient")
@Entity
@Builder
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode(exclude = {"version"})
public class PatientModel {

    @Id
    @GeneratedValue(strategy = IDENTITY)
    private Long patientId;

    @Version
    private Integer version;

    private String data;
    private Boolean deleted;
    private Long versionTimestamp;
}
```

src/main/java/com/baidak/application/dto/PatientCreateRequest.java

```
package com.baidak.application.dto;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
```

```
import lombok.NoArgsConstructor;


@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class PatientCreateRequest {


    private String data;
}
```

src/main/java/com/baidak/application/dto/PatientUpdateRequest.java

```
package com.baidak.application.dto;


import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class PatientUpdateRequest {

    private Long patientId;
    private String data;
}
```

src/main/java/com/baidak/application/repository/PatientRepository.java

```
package com.baidak.application.repository;

import com.baidak.application.domain.PatientModel;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface PatientRepository extends JpaRepository<PatientModel, Long> {
}
```

src/main/java/com/baidak/application/service/PatientService.java

```
package com.baidak.application.service;

import com.baidak.application.domain.PatientModel;
import com.baidak.application.repository.PatientRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import javax.persistence.EntityNotFoundException;
import java.time.Instant;

@Service
@RequiredArgsConstructor
public class PatientService {

    private final PatientRepository patientRepository;

    @Transactional
```

```
    public PatientModel createPatient(PatientModel patientToCreate) {
        return patientRepository.saveAndFlush(patientToCreate);
    }

    @Transactional(readOnly = true)
    public PatientModel getPatient(Long patientId) {
        PatientModel patientModel = patientRepository.getOne(patientId);
        if (Boolean.FALSE.equals(patientModel.getDeleted())) {
            return patientModel;
        } else {
            throw new EntityNotFoundException("Patient with id " + patientId + " does not exist");
        }
    }

    @Transactional(readOnly = true)
    public PatientModel updatePatient(PatientModel patientToUpdate) {
        PatientModel patientModel = patientRepository.getOne(patientToUpdate.getPatientId());
        if (Boolean.FALSE.equals(patientModel.getDeleted())) {
            patientModel.setData(patientToUpdate.getData());
            patientModel.setVersionTimestamp(Instant.now().getEpochSecond());
            return patientRepository.saveAndFlush(patientModel);
        } else {
            throw new EntityNotFoundException("Patient with id " + patientToUpdate.getPatientId() +
" does not exist");
        }
    }

    @Transactional(readOnly = true)
    public PatientModel deletePatient(Long patientId) {
        PatientModel patientModel = patientRepository.getOne(patientId);
        if (Boolean.FALSE.equals(patientModel.getDeleted())) {
            patientModel.setDeleted(true);
            patientModel.setVersionTimestamp(Instant.now().getEpochSecond());
            return patientRepository.saveAndFlush(patientModel);
        } else {
            throw new EntityNotFoundException("Patient with id " + patientId + " does not exist");
        }
    }
}
```

src/main/java/com/baidak/application/service/PatientManagementService.java

```
package com.baidak.application.service;

import com.baidak.application.domain.PatientModel;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
@RequiredArgsConstructor
public class PatientManagementService {

    private final PatientService patientService;

    @Transactional
    public PatientModel createPatient(PatientModel patientToCreate) {
        return patientService.createPatient(patientToCreate);
    }

    @Transactional(readOnly = true)
    public PatientModel getPatient(Long patientId) {
        return patientService.getPatient(patientId);
    }

    @Transactional
    public PatientModel updatePatient(PatientModel patientToUpdate) {
        return patientService.updatePatient(patientToUpdate);
    }

    @Transactional
    public PatientModel deletePatient(Long patientId) {
        return patientService.deletePatient(patientId);
    }
}
```

src/main/java/com/baidak/application/fhir/validation/CustomFHIRRegistryResourceProvider.java

```
package com.baidak.application.fhir.validation;


import com.ibm.fhir.registry.util.PackageRegistryResourceProvider;
import org.springframework.stereotype.Component;

@Component
public class CustomFHIRRegistryResourceProvider extends PackageRegistryResourceProvider {

    @Override
    public String getPackageId() {
        return "fhir.structure_definition";
    }
}
```

src/main/java/com/baidak/application/fhir/Profile.java

```
package com.baidak.application.fhir;

import lombok.Getter;
import lombok.RequiredArgsConstructor;

@Getter
@RequiredArgsConstructor
public enum Profile {

    SIMPLE_PATIENT("SimplePatient");

    private final String name;
}
```

src/main/java/com/baidak/application/exception/InvalidPatientRuntimeException.java

```
package com.baidak.application.exception;

import com.ibm.fhir.model.resource.OperationOutcome;
import lombok.AllArgsConstructor;
import lombok.Getter;

import java.util.List;

@Getter
@AllArgsConstructor
public class InvalidPatientRuntimeException extends RuntimeException {

    private final List<OperationOutcome.Issue> issueList;

    @Override
    public String getMessage() {
        StringBuilder message = new StringBuilder();
        for (OperationOutcome.Issue issue : issueList) {
            message.append(" Details: ").append(issue.getDetails()).append(" Expression:
").append(issue.getExpression());
        }
        return message.toString();
    }
}
```

src/main/java/com/baidak/application/converter/PatientUpdateRequestToPatientModelConverter.java

```
package com.baidak.application.converter;
```

```
import com.baidak.application.domain.PatientModel;
import com.baidak.application.dto.PatientUpdateRequest;
import org.springframework.core.convert.converter.Converter;
import org.springframework.stereotype.Component;

@Component
public class PatientUpdateRequestToPatientModelConverter implements Converter<PatientUpdateRequest,
PatientModel> {

    @Override
    public PatientModel convert(PatientUpdateRequest request) {
        return PatientModel.builder()
                .patientId(request.getPatientId())
                .data(request.getData())
                .build();
    }
}
```

src/main/java/com/baidak/application/converter/PatientCreateRequestToPatientModelConverter.java

```
package com.baidak.application.converter;

import com.baidak.application.domain.PatientModel;
import com.baidak.application.dto.PatientCreateRequest;
import org.springframework.core.convert.converter.Converter;
import org.springframework.stereotype.Component;

import java.time.Instant;

@Component
public class PatientCreateRequestToPatientModelConverter implements Converter<PatientCreateRequest,
PatientModel> {

    @Override
    public PatientModel convert(PatientCreateRequest request) {
        return PatientModel.builder()
                .data(request.getData())
                .deleted(false)
                .versionTimestamp(Instant.now().getEpochSecond())
                .build();
    }
}
```

src/main/java/com/baidak/application/controller/PatientController.java

```
package com.baidak.application.controller;

import com.ibm.fhir.model.parser.exception.FHIRParserException;
import com.ibm.fhir.validation.exception.FHIRValidationException;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.tags.Tag;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;

import java.io.IOException;

import static org.springframework.http.HttpStatus.CREATED;
import static org.springframework.http.HttpStatus.OK;

@Validated
@Tag(name = "Patient API")
@RequestMapping("/fhir/Patient")
@ApiResponse(responseCode = "405", description = "Method Not Allowed")
@ApiResponse(responseCode = "500", description = "Internal Server Error ")
public interface PatientController {

    String APPLICATION_FHIR_JSON = "application/fhir+json";
```

```
    @ResponseStatus(CREATED)
    @Operation(summary = "Create Patient", responses = {
            @ApiResponse(responseCode = "201", description = "Created",
                    content = {@Content(mediaType = "application/json")}),
            @ApiResponse(responseCode = "400", description = "Bad request"),
            @ApiResponse(responseCode = "404", description = "Not Found"),
            @ApiResponse(responseCode = "406", description = "Not Acceptable", content = {
                    @Content(mediaType = "text/plain")}),
            @ApiResponse(responseCode = "409", description = "Conflict"),
            @ApiResponse(responseCode = "415", description = "Unsupported media type")},
            description = "Create Patient endpoint")
    @PostMapping
    String createPatient(@RequestBody final String proposedPatient)
            throws IOException, FHIRParserException, FHIRValidationException;

    @ResponseStatus(OK)
    @Operation(summary = "Get Patient", responses = {
            @ApiResponse(responseCode = "200", description = "OK"),
            @ApiResponse(responseCode = "400", description = "Bad request"),
            @ApiResponse(responseCode = "404", description = "Not Found"),
            @ApiResponse(responseCode = "406", description = "Not Acceptable", content = {
                    @Content(mediaType = "text/plain")})},
            description = "Get Patient endpoint")
    @GetMapping(path = "/{patientId}")
    String getPatient(@PathVariable final Long patientId) throws IOException, FHIRParserException;

    @ResponseStatus(OK)
    @Operation(summary = "Update Patient", responses = {
            @ApiResponse(responseCode = "200", description = "OK",
                    content = {@Content(mediaType = "application/json")}),
            @ApiResponse(responseCode = "400", description = "Bad request"),
            @ApiResponse(responseCode = "404", description = "Not Found"),
            @ApiResponse(responseCode = "406", description = "Not Acceptable", content = {
                    @Content(mediaType = "text/plain")}),
            @ApiResponse(responseCode = "415", description = "Unsupported media type")},
            description = "Create Patient endpoint")
    @PutMapping(path = "/{patientId}")
    String updatePatient(@PathVariable final Long patientId, @RequestBody final String
proposedPatient)
            throws IOException, FHIRParserException, FHIRValidationException;

    @ResponseStatus(OK)
    @Operation(summary = "Delete Practice", responses = {
            @ApiResponse(responseCode = "200", description = "OK",
                    content = {@Content(mediaType = "application/json")}),
            @ApiResponse(responseCode = "400", description = "Bad request"),
            @ApiResponse(responseCode = "404", description = "Not Found")},
            description = "Create Patient endpoint")
    @DeleteMapping(path = "/{patientId}")
    String deletePatient(@PathVariable final Long patientId) throws IOException,
FHIRParserException;
}
```

src/main/java/com/baidak/application/controller/DefaultControllerAdvice.java

```
package com.baidak.application.controller;

import com.baidak.application.exception.InvalidPatientRuntimeException;
import com.ibm.fhir.model.format.Format;
import com.ibm.fhir.model.generator.FHIRGenerator;
import com.ibm.fhir.model.generator.exception.FHIRGeneratorException;
import com.ibm.fhir.model.resource.OperationOutcome;
import com.ibm.fhir.model.type.Code;
import com.ibm.fhir.model.type.CodeableConcept;
import com.ibm.fhir.model.type.Coding;
import com.ibm.fhir.model.type.String;
import com.ibm.fhir.model.type.code.IssueSeverity;
import com.ibm.fhir.model.type.code.IssueType;
import lombok.extern.slf4j.Slf4j;
import org.springframework.core.annotation.Order;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.dao.InvalidDataAccessApiUsageException;
import org.springframework.dao.InvalidDataAccessResourceUsageException;
import org.springframework.data.mapping.PropertyReferenceException;
import org.springframework.http.converter.HttpMessageNotReadableException;
```

```
import org.springframework.validation.BindException;
import org.springframework.web.HttpMediaTypeNotAcceptableException;
import org.springframework.web.HttpMediaTypeNotSupportedException;
import org.springframework.web.HttpRequestMethodNotSupportedException;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.method.annotation.MethodArgumentTypeMismatchException;

import javax.persistence.EntityNotFoundException;
import javax.servlet.http.HttpServletRequest;
import javax.validation.ConstraintViolationException;
import java.io.IOException;
import java.io.StringWriter;
import java.io.Writer;
import java.util.Collections;

import static org.springframework.http.HttpStatus.BAD_REQUEST;
import static org.springframework.http.HttpStatus.CONFLICT;
import static org.springframework.http.HttpStatus.INTERNAL_SERVER_ERROR;
import static org.springframework.http.HttpStatus.METHOD_NOT_ALLOWED;
import static org.springframework.http.HttpStatus.NOT_ACCEPTABLE;
import static org.springframework.http.HttpStatus.NOT_FOUND;
import static org.springframework.http.HttpStatus.UNSUPPORTED_MEDIA_TYPE;

@Slf4j
@Order
@ControllerAdvice
@ResponseBody
public class DefaultControllerAdvice {

    @ResponseStatus(BAD_REQUEST)
    @ExceptionHandler(InvalidPatientRuntimeException.class)
    @ResponseBody
    public java.lang.String invalidAppointmentRuntimeException(InvalidPatientRuntimeException
exception) {
        log.warn("Invalid patient", exception);
        try (Writer writer = new StringWriter()) {
            OperationOutcome operationOutcome = OperationOutcome.builder()
                    .issue(exception.getIssueList())
                    .build();
            FHIRGenerator.generator(Format.JSON).generate(operationOutcome, writer);
            return writer.toString();
        } catch (IOException | FHIRGeneratorException e) {
            throw new RuntimeException(e);
        }
    }

    @ResponseStatus(NOT_ACCEPTABLE)
    @ExceptionHandler(HttpMediaTypeNotAcceptableException.class)
    public java.lang.String notAcceptable(HttpServletRequest request,
                                          HttpMediaTypeNotAcceptableException exception) {

        return buildResponse(request, exception);
    }

    @ResponseStatus(NOT_FOUND)
    @ExceptionHandler(EntityNotFoundException.class)
    public java.lang.String entityNotFoundException(
            HttpServletRequest request, EntityNotFoundException exception) {
        return buildResponse(request, exception);
    }

    @ResponseStatus(BAD_REQUEST)
    @ExceptionHandler(HttpMessageNotReadableException.class)
    @ResponseBody
    public java.lang.String httpMessageNotReadableException(
            HttpServletRequest request, HttpMessageNotReadableException exception) {
        return buildResponse(request, exception);
    }

    @ResponseStatus(BAD_REQUEST)
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public java.lang.String validationException(
            HttpServletRequest request, MethodArgumentNotValidException exception) {
        return buildResponse(request, exception);
    }
```

```
    @ResponseStatus(BAD_REQUEST)
    @ExceptionHandler(ConstraintViolationException.class)
    public java.lang.String handleConstraintViolationException(
            HttpServletRequest request, ConstraintViolationException exception) {
        return buildResponse(request, exception);
    }

    @ResponseStatus(METHOD_NOT_ALLOWED)
    @ExceptionHandler(HttpRequestMethodNotSupportedException.class)
    public java.lang.String httpMethodNotSupportedException(
            HttpServletRequest request, HttpRequestMethodNotSupportedException exception) {
        return buildResponse(request, exception);
    }

    @ResponseStatus(UNSUPPORTED_MEDIA_TYPE)
    @ExceptionHandler(HttpMediaTypeNotSupportedException.class)
    public Object httpMediaTypeNotSupported(
            HttpServletRequest request, HttpMediaTypeNotSupportedException exception) {
        return buildResponse(request, exception);
    }

    @ResponseStatus(BAD_REQUEST)
    @ExceptionHandler(MethodArgumentTypeMismatchException.class)
    public java.lang.String methodArgumentTypeMismatchException(
            HttpServletRequest request, MethodArgumentTypeMismatchException exception) {
        return buildResponse(request, exception);
    }

    @ResponseStatus(BAD_REQUEST)
    @ExceptionHandler(BindException.class)
    public java.lang.String bindException(
            HttpServletRequest request, BindException exception) {
        return buildResponse(request, exception);
    }

    @ResponseStatus(BAD_REQUEST)
    @ExceptionHandler(PropertyReferenceException.class)
    public java.lang.String propertyReferenceException(
            HttpServletRequest request, PropertyReferenceException exception) {
        return buildResponse(request, exception);
    }

    @ResponseStatus(BAD_REQUEST)
    @ExceptionHandler(InvalidDataAccessResourceUsageException.class)
    public java.lang.String invalidDataAccessResourceUsageException(
            HttpServletRequest request, InvalidDataAccessResourceUsageException exception) {
        return buildResponse(request, exception);
    }

    @ResponseStatus(BAD_REQUEST)
    @ExceptionHandler(InvalidDataAccessApiUsageException.class)
    public java.lang.String invalidDataAccessApiUsageException(
            HttpServletRequest request, InvalidDataAccessApiUsageException exception) {
        return buildResponse(request, exception);
    }

    @ResponseStatus(CONFLICT)
    @ExceptionHandler(DataIntegrityViolationException.class)
    public java.lang.String conflict(
            HttpServletRequest request, DataIntegrityViolationException exception) {
        return buildResponse(request, exception);
    }

    @ExceptionHandler(Throwable.class)
    @ResponseStatus(INTERNAL_SERVER_ERROR)
    public java.lang.String generalThrowable(
            HttpServletRequest request, Throwable throwable) {
        return buildResponse(request, throwable);
    }

    private java.lang.String buildResponse(HttpServletRequest request, Throwable throwable) {
        log.warn("Exception occurred, path=" + request.getRequestURI() + ", method=" +
request.getMethod(), throwable);
        OperationOutcome operationOutcome = OperationOutcome.builder()
                .issue(Collections.singletonList(
                        OperationOutcome.Issue.builder()
                                .code(IssueType.INVALID)
                                .severity(IssueSeverity.FATAL)
                                .details(CodeableConcept.builder()
                                        .text(String.of(throwable.getMessage()))
```

```
                                        .coding(Collections.singletonList(
                                                Coding.builder()
                                                        .system(com.ibm.fhir.model.type.Uri.of(
                                                                "FhirService"))
                                                        .code(Code.code("INVALID_FHIR_OPERATION"))
                                                        .build()))
                                        .build())
                                .build()))
                .build();
        try (Writer writer = new StringWriter()) {
            FHIRGenerator.generator(Format.JSON).generate(operationOutcome, writer);
            return writer.toString();
        } catch (IOException | FHIRGeneratorException e) {
            throw new RuntimeException(e);
        }
    }
}
```

src/main/java/com/baidak/application/controller/CapabilityStatementController.java

```
package com.baidak.application.controller;

import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.tags.Tag;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import java.io.IOException;

@Validated
@Tag(name = "Patient API")
@RequestMapping
@ApiResponse(responseCode = "200", description = "OK")
@ApiResponse(responseCode = "405", description = "Method Not Allowed")
@ApiResponse(responseCode = "500", description = "Internal Server Error ")
public interface CapabilityStatementController {

    @RequestMapping(method = RequestMethod.OPTIONS)
    String capabilityStatement() throws IOException;
}
```

src/main/java/com/baidak/application/controller/impl/PatientControllerImpl.java

```
package com.baidak.application.controller.impl;

import com.baidak.application.controller.PatientController;
import com.baidak.application.domain.PatientModel;
import com.baidak.application.dto.PatientCreateRequest;
import com.baidak.application.dto.PatientUpdateRequest;
import com.baidak.application.exception.InvalidPatientRuntimeException;
import com.baidak.application.service.PatientManagementService;
import com.ibm.fhir.model.format.Format;
import com.ibm.fhir.model.generator.FHIRGenerator;
import com.ibm.fhir.model.generator.exception.FHIRGeneratorException;
import com.ibm.fhir.model.parser.FHIRParser;
import com.ibm.fhir.model.parser.exception.FHIRParserException;
import com.ibm.fhir.model.resource.OperationOutcome;
import com.ibm.fhir.model.resource.Patient;
import com.ibm.fhir.model.type.code.IssueSeverity;
import com.ibm.fhir.validation.FHIRValidator;
import com.ibm.fhir.validation.exception.FHIRValidationException;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.core.convert.ConversionService;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.StringWriter;
import java.io.Writer;
import java.util.List;
```

```
import java.util.stream.Collectors;

import static com.baidak.application.fhir.Profile.SIMPLE_PATIENT;


@Slf4j
@Controller
@ResponseBody
@RequiredArgsConstructor
public class PatientControllerImpl implements PatientController {

    private final ConversionService conversionService;
    private final PatientManagementService patientManagementService;

    @Override
    public String createPatient(@RequestBody final String proposedPatient)
            throws IOException, FHIRParserException, FHIRValidationException {
        log.debug("Received request to create new Patient, proposedPatient:" + proposedPatient);
        Patient receivedPatient = parsePatient(proposedPatient).toBuilder()
                .id(null)
                .build();
        validatePatient(receivedPatient);
        String proposedPatientWithoutId = buildJson(receivedPatient);
        PatientCreateRequest patientCreateRequest = PatientCreateRequest.builder()
                .data(proposedPatientWithoutId)
                .build();
        PatientModel patientToCreate = conversionService.convert(patientCreateRequest,
PatientModel.class);
        PatientModel createdPatient = patientManagementService.createPatient(patientToCreate);
        Patient patientWithId = parsePatient(createdPatient.getData()).toBuilder()
                .id(createdPatient
                        .getPatientId()
                        .toString())
                .build();
        return buildJson(patientWithId);
    }

    @Override
    public String getPatient(@PathVariable final Long patientId) throws IOException,
FHIRParserException {

        log.debug("Received request to get a Patient, patientId:" + patientId);
        PatientModel patient = patientManagementService.getPatient(patientId);
        Patient patientWithId = parsePatient(patient.getData()).toBuilder()
                .id(patient
                        .getPatientId()
                        .toString())
                .build();
        return buildJson(patientWithId);
    }

    @Override
    public String updatePatient(@PathVariable final Long patientId, @RequestBody final String
proposedPatient)
            throws IOException, FHIRParserException, FHIRValidationException {
        log.debug("Received request to update existing Patient, proposedPatient:" +
proposedPatient);
        Patient receivedPatient = parsePatient(proposedPatient).toBuilder()
                .id(null)
                .build();
        validatePatient(receivedPatient);
        String proposedPatientWithoutId = buildJson(receivedPatient);

        PatientUpdateRequest patientUpdateRequest = PatientUpdateRequest.builder()
                .patientId(patientId)
                .data(proposedPatientWithoutId)
                .build();

        PatientModel patientToUpdate = conversionService.convert(patientUpdateRequest,
PatientModel.class);
        PatientModel patient = patientManagementService.updatePatient(patientToUpdate);
        Patient patientWithId = parsePatient(patient.getData()).toBuilder()
                .id(patient
                        .getPatientId()
                        .toString())
                .build();
        return buildJson(patientWithId);
    }

    @Override
```

```
    public String deletePatient(@PathVariable final Long patientId) throws IOException,
FHIRParserException {
        log.debug("Received request to delete Patient, patientId:" + patientId);
        PatientModel patient = patientManagementService.deletePatient(patientId);
        Patient patientWithId = parsePatient(patient.getData()).toBuilder()
                .id(patient
                        .getPatientId()
                        .toString())
                .build();
        return buildJson(patientWithId);
    }

    private Patient parsePatient(String requestBody) throws FHIRParserException, IOException {
        Patient patient;

        try (InputStream in = new ByteArrayInputStream(requestBody.getBytes())) {
            patient = FHIRParser.parser(Format.JSON).parse(in);
        }

        return patient;
    }

    private void validatePatient(Patient patient) throws FHIRValidationException {
        FHIRValidator fhirValidator = FHIRValidator.validator();
        List<OperationOutcome.Issue> issueList = fhirValidator
                .validate(patient, SIMPLE_PATIENT.getName())
                .stream()
                .filter(issue -> issue.getSeverity().equals(IssueSeverity.FATAL)
                        || issue.getSeverity().equals(IssueSeverity.ERROR))
                .collect(Collectors.toList());

        if (!issueList.isEmpty()) {
            throw new InvalidPatientRuntimeException(issueList);
        }
    }

    private String buildJson(Patient patient) {
        try (Writer writer = new StringWriter()) {
            FHIRGenerator.generator(Format.JSON).generate(patient, writer);
            return writer.toString();
        } catch (IOException | FHIRGeneratorException e) {
            throw new RuntimeException(e);
        }
    }
}
```

src/main/java/com/baidak/application/controller/impl/CapabilityStatementControllerImpl.java

```
package com.baidak.application.controller.impl;

import com.baidak.application.controller.CapabilityStatementController;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.io.FileUtils;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ResponseBody;

import java.io.File;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

@Slf4j
@Controller
@ResponseBody
@RequiredArgsConstructor
public class CapabilityStatementControllerImpl implements CapabilityStatementController {

    @Override
    public String capabilityStatement() throws IOException {
        return FileUtils.readFileToString(new
File("src/main/resources/fhir/CapabilityStatement.json"),
                StandardCharsets.UTF_8);
    }
}
```

src/main/java/com/baidak/application/configuration/SwaggerConfiguration.java

```
package com.baidak.application.configuration;

import io.swagger.v3.oas.annotations.OpenAPIDefinition;
import io.swagger.v3.oas.annotations.servers.Server;
import org.springframework.context.annotation.Configuration;

@Configuration
@OpenAPIDefinition(servers = {@Server(url = "/")})
public class SwaggerConfiguration {
}
```

src/main/java/com/baidak/application/configuration/FHIRValidatorConfiguration.jav a

```
package com.baidak.application.configuration;

import com.baidak.application.fhir.Profile;
import com.baidak.application.fhir.validation.CustomFHIRRegistryResourceProvider;
import com.ibm.fhir.model.resource.OperationOutcome;
import com.ibm.fhir.model.resource.StructureDefinition;
import com.ibm.fhir.model.type.code.IssueSeverity;
import com.ibm.fhir.profile.ProfileSupport;
import com.ibm.fhir.validation.FHIRValidator;
import lombok.extern.slf4j.Slf4j;
import org.springframework.context.annotation.Configuration;

import javax.annotation.PostConstruct;
import java.util.List;
import java.util.stream.Collectors;

@Slf4j
@Configuration
class FHIRValidatorConfiguration {
    @PostConstruct
    void configureValidator() {
        CustomFHIRRegistryResourceProvider provider = new CustomFHIRRegistryResourceProvider();
        provider.getRegistryResources();
        validateStructureDefinitions(FHIRValidator.validator());
    }

    private void validateStructureDefinitions(FHIRValidator fhirValidator) {
        for (Profile profile : Profile.values()) {
            try {
                StructureDefinition structureDefinition =
ProfileSupport.getProfile(profile.getName());
                List<OperationOutcome.Issue> issueList =
fhirValidator.validate(structureDefinition).stream()
                        .filter(issue -> issue.getSeverity().equals(IssueSeverity.FATAL)
                                || issue.getSeverity().equals(IssueSeverity.ERROR))
                        .collect(Collectors.toList());
                if (!issueList.isEmpty()) {
                    log.warn("Found validation issue with {} profile : {} ", profile.getName(),
issueList.toArray());
                }
            } catch (Exception e) {
                log.warn("Cannot load structure definition for profile '{}'", profile.getName(), e);
            }
        }
    }
}
```

ДОДАТОК Б

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

Факультет інформаційних технологій
Кафедра програмного забезпечення комп'ютерних систем

ВІДГУК

Керівника
економічної
частини Касьяненко Л.В., к.е.н., доцента каф. ПЕП та ПУ

(прізвище, ім'я, по батькові, вчене звання, посада, місце роботи)

На кваліфікаційну роботу

студента Байдака Леоніда Васильовича

(прізвище, ім'я, по батькові)

курсу II групи 122м-20-1
спеціальності Комп'ютерні науки
на тему Розробка та дослідження ефективності впровадження веб-сервісу для валідації і берігання медичних документів за стандартом FHIR v4.

ДОДАТОК В

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файла	Опис
Пояснювальні документи	
Baidak_122m_20_1_diploma.docx	Пояснювальна записка до магістерської роботи. Документ Word.
Baidak_122m_20_1_diploma.pdf	Пояснювальна записка до магістерської роботи. Документ PDF.
Програма	
app.rar	Архів. Містить код програми.
Презентація	
Baidak_122m_20_1_presentation.pptx	Презентація до магістерської роботи