КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

122 «Комп'ютерні науки»
(шифр і назва спеціальності)

«Прикладне програмування»
(назва освітньої програми)

Кваліфікаційна робота бакалавра

на тему: «Інформаційна технологія діагностування хвороб»

Виконав _____
(Підпис)
Щавінський Ярослав-Петро Мирославович
(прізвище, ім'я, по батькові)

Керівник Краснощок Віктор Миколайович
(прізвище, ім'я, по батькові)

_____
(Резолюція «До захисту»)

Попередній захист:

_____
(Висновок: "До захисту в екзаменаційній комісії ")
Завідувач кафедри _____ Плескач В.Л.
(Підпис) (Прізвище, ініціали) (Дата)

Київ – 2021

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Номер	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	26.10.2020	
2.	Видача завдання кваліфікаційної роботи бакалавра	23.11.2020	заява
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	01.12.2020	
4.	Затвердження плану кваліфікаційної роботи бакалавра	18.02.2021	
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	25.02.2021	
6.	Підготовка і подання науковому керівнику першого варіанту I розділу роботи	05.03.2021	
7.	Підготовка і подання науковому керівнику першого варіанту II розділу роботи	09.04.2021	
8.	Підготовка і подання науковому керівнику першого варіанту III розділу роботи	07.05.2021	
9.	Подання роботи у першому варіанті	11.05.2021	
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	12.05.2021	
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	**24.05.2021**	
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	28.05.2021	
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	11.06.2021	
14.	Захист кваліфікаційної роботи бакалавра	23.06.2021	

Здобувач вищої освіти _____
(підпис)

Керівник _____
(підпис)

ВІДОМІСТЬ ДИПЛОМНОЇ РОБОТИ

Зміст пояснювальної записки (перелік питань під час дослідження)

Складові частини дипломної роботи	Обсяг, арк. 65
Титульний аркуш	1 ст.
Календарний план виконання роботи	1 ст.
Відомість дипломної роботи	1 ст.
Анотація	1 ст.
Анотація (іноземною мовою-англійською)	1 ст.
Зміст	1 ст.
Вступ	2 ст.
1. теоретичні основи побудови програмної системи «інформаційна технологія діагностування хвороб»	4 ст.
2. проектні та технічні рішення. забезпечення проектованої системи.	26 ст.
3. Опис роботи системи	15 ст.
Висновки	2 ст.
Перелік посилань	2 ст.
Додатки	11 ст.

				ДП XXXX 00.000.00		
	ПІБ	Підп.	Дата			
Розробн.				Відомість дипломної роботи	Лист	Листів
Керівн.						
Н/контр.	Макаренко С.А.					
Зав.каф.	Плескач В.Л.					

ЗМІСТ

Анотація (реферат)

Обсяг роботи: 55 сторінок, 26 ілюстрацій, 25 джерел посилань.

До бакалаврської дипломної роботи Щавінського Ярослава-Петра Мирославовича на тему «Інформаційна технологія діагностування хвороб»

Метою моєї дипломної роботи є побудова мікросервісної веб-системи для визначення діагнозів за симптомами хвороби. Створена система має надавати можливість автоматичного визначення діагнозу шляхом використання нейронної мережі та машинного навчання. Також система включає в себе можливість діагностування хвороби за допомогою спілкування з живим спеціалістом, та запис отриманих результатів в свою базу даних.

Об’єктом дослідження бакалаврської роботи є створення системи, яка буде здатна з певною точністю визначати діагноз по симптомам, та давати змогу визначати діагнози з участю живих спеціалістів. Питання обліку даних які виникають в процесі роботи цієї системи.

Предметом дослідження бакалаврської роботи роботи є програмна система «Інформаційна технологія діагностування хвороб». Її принципи та підходи для організації основного функціоналу, створення нейронної мережі яка буде визначати діагнози за симптомами, автентифікація користувачів та захист даних.

Результат роботи. Було досліджено уже існуючі веб-сайти для визначення діагнозів за симптомами та обліку медичних даних, розроблено веб додаток для передбачення хвороби за симптомами, визначення діагнозу за допомогою лікаря та обліку пацієнтів і їх діагнозів.

Ключові слова: визначення хвороб, найронні мережі, java, AWS, Tensorflow.

Annotation (abstract)

Thesis: 55 pages, 26 figures, 25 sources.

To the bachelor's work of Shchavinskyi Yaroslav-Petero Myroslavovich on the topic "Information system for disease diagnosis"

The purpose of my thesis is to build a microservice web system for diagnosing diseases by symptoms. Created system should provide the possibility of automatic diagnosis by using a neural network and machine learning. The system also includes the ability to diagnose the disease through communication with a specialist, and record the results of diagnosis in its database.

The object of research of the bachelor's work is to create a system that will be able to determine diagnoses by symptoms with a certain accuracy, and to allow to determine diagnoses with the participation of living specialists. Issues of accounting data that arise during the operation of this system.

The subject of research of the bachelor's work is the software system "Information system for disease diagnosis". Principles of it’s work, implementation of main functionality, creation of the neural network that will determine diagnoses by symptoms, user authentication and data protection.

The result of work. Research of existing systems for diagnosis prediction. Created working website, that able to predict diagnosis by given symptoms, and gives possibility to ask help in actual doctor. Service saves user and medical records, and able to use them in further improvement if inner neural network.

Key words: disease prediction, neural networks, java, AWS, TensorFlow, Spring.

ВСТУП

Умови пандемії та розвиток інформаційних технологій спричиняє перехід все більшої кількості галузей на віддалену роботу. Медична сфера не є виключенням. Все більше людей хочуть мати змогу отримувати консультації від лікаря онлайн. Використовуючи сучасні технології машинного навчання такі консультації можна ще більше спростити використовуючи систему замість спеціаліста для спрощення діагностування та лікування різноманітних хвороб

Лікування пацієнтів в лікарні створює велику кількість документів та продукує багато інформації. Через це виникає сильна потреба в її зберіганні та організації. На даний момент в більшості лікарень нашої країни використовуються тільки паперові документи, використання електронної системи документообігу дозволило б сильно спростити зберігання та пошук інформації та облегшити доступ до потрібної інформації пацієнтами.

Актуальність теми цієї бакалаврської роботи зумовлено важливістю створення можливостей для віддаленої роботи частини функцій медичних закладів. Також така система зможе зменшити навантаження на спеціалістів в цій галузі, перекладаючи частину їх обов’язків на комп’ютер. Також реалізація такої системи також може значно спростити та зробити прозорішими такі процеси як облік пацієнтів та їхніх діагнозів. Зберігання інформації в електронному вигляді дозволить отримувати доступ до неї з любого місця та сильно спростить її пошук.

Об’єктом дослідження бакалаврської роботи є створення системи, яка буде здатна з певною точністю визначати діагноз по симптомам, та давати змогу визначати діагнози з участю живих спеціалістів. Питання обліку даних які виникають в процесі роботи цієї системи. Принципи, концептуальні підходи до побудови програмної системи «Інформаційна технологія діагностування хвороб».

Предметом дослідження бакалаврської роботи роботи є програмна система «Інформаційна технологія діагностування хвороб». Її принципи та підходи для організації основного функціоналу, створення нейронної мережі яка буде

визначати діагнози за симптомами, автентифікація користувачів та захист данних

У процесі виконання бакалаврської роботи було використано: платформу хмарних обчислень AWS, об'єктно-орієнтовану мову програмування Phyton, бібліотеку машинного навчання TensorFlow, технології хмарних обчилень AWS Sagemaker, AWS DynamoDB, AWS Lambda, AWS API Gateway, AWS S3, об'єктно-орієнтовану мову програмування java, та її фреймворк Spring а саме його компоненти: Spring Boot, Spring Core, Spring Web, Spring Data JPA, Spring Security, Spring Testing. Java бібліотеки: Lombok, Mockito, log4j, коннектор JDBC для MySql, javax servlet api. Як СУБД використовується MySQL. Для front-end частини сайту використовується html, css, javaScript та бібліотека jQuery. Під час розробки сайту використувувалось інтегроване середовище розробки IntelliJ IDEA, інструмент для візуального проектування баз даних MySQL Workbench та інструменти відладки Google Chrome.

Структура роботи: бакалаврська робота складається із вступу, трьох розділів, розподілених на підрозділи, висновку, додатків і списку використаних джерел.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ ПРОГРАМНОЇ СИСТЕМИ «ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ДІАГНОСТУВАННЯ ХВОРОБ»

1.1 Аналіз задачі яку має виконувати проект «Інформаційна технологія діагностування хвороб, та підстави для його реалізації

В зв’язку з глобальною пандемією та загальним збільшенням навантаження на медичні заклади все більше людей хоче мати змогу отримувати медичну консультацію онлайн, без потреби відвідувати лікаря особисто. В теорії, не тяжкі хвороби можуть бути автоматично визначені за допомогою інформаційної технології, яка проаналізує симптоми людини, та дасть діагноз з певною ймовірністю, на основі попередніх запитів. Якщо ж пацієнту не достатньо такої відповіді, він повинен мати змогу створити запит на допомогу від кваліфікованого лікаря. Такий підхід, має зменшити навантаження на медичні установи, надати пацієнтам зручний сервіс для діагностування не виходячи з дому, а також, дати змогу людям, які не хворіють в даний момент, але хочуть перевірити який діагноз найбільш вірогідний при певних симптомах, змогу це зробити.

Для реалізації роботи системи автоматичного визначення діагнозу потрібно щоб вона аналізувала реальні данні, та могла знаходити закономірності в них. Для збільшення точності такого визначення система може використовувати діагнози, які були поставлені живими лікарями в процесі експулатації цеї технології як вхідні данні, та будувати всі припущення на їх основі. Таким чином система буде самонавчатися, та збільшувати точність передбачення в процесі її використання.

Найбільшої ефективності можна досягти при інтеграції цієї технології з системою докуметообігу поліклініки або лікарні. Проходження лікування в лікарні є необхідністю для багатьох людей у всіх країнах світу. Теоретична логіка реалізації системи для створення документації цього процесу може виглядати як сайт, на якому реєструються працівники лікарні та пацієнти, на цьому сайті зберігаються історії хвороб та приписані пацієнтам препарати,

процедури та операції. Користувач сайту який зареєстрований як лікар може переглядати данні пацієнтів, ставити їм діагнози та прописувати різноманітні види лікування. Користувачі сайту які мають меншу спеціалізацію, такі як медсестри можуть тільки приписувати медикаменти, проте не можуть ставити нові діагнози. Пацієнт може переглядати свою історію хвороб, та бачити всі медикаменти та процедури які йому приписані. Оскільки данні про лікування є приватними потрібно приділити більше уваги до захисту системи, заборонити пацієнтам переглядати данні інших користувачів. При інтеграції такої системи з технологією передбачення діагнозів, можна використовувати спільну базу даних для користувачів, та діагнозів. В такому випадку, сайт також повинен мати сторінку в якій можна вибрати список симптомів, та отримати результат передбачення діагнозу системою по ним, після цього авторизовані користувачі будуть мати змогу створити запит на отримання допомоги в діагностуванні від лікаря. Пацієнт може переглядати запити які він створив та коментувати їх, а лікар може перглядати запити всі запити на допомогу в системі, коментувати їх, та виставляти остаточний діагноз. Після виставлення діагнозу, він записується в загальну базу даних системи, та в базу узагальнених даних про хвороби, для подальшого навчання технології передбачення з їх використанням. База узагальнених даних про хвороби не містить ніякої особистої інформації пацієнтів крім симптомів та остаточного діагнозу, тому не повинна створювати колізій з такими специфікаціями зберігання даних як GDPR та HIPPA.

1.2 Існуючі системи передбачення діагнозів за симптомами

1. apimedic [23]

Apidaemic надає REST сервіс для передбачення діагнозів за симптомами. Цей сервіс пропонує багато можливостей для визначення різних хвороб, проте не надає графічного інтерфейсу для користувачів, та є платним. Також через обмежену інформацію на сайті компанії, не зрозуміло алгоритм роботи системи, та наскільки актуальними даними вона оперує.

2. symptomate [24]

Symptomate веб сайт, створений на основі infermedica, для визначення діагнозів по симптомам та інших данних методами машинного навчання, дозволяє безкоштовно пройти опитування, після цього видає передбачення діагнозу та його ймовірність. Цей сервіс немає інтеграції з медичним закладом, для отримання швидкої консультації за наданими данними. Також, опитування займає багато часу та містить питання, які повторюються.

3. symptoms.webmd [25]

Symptoms.webmd веб сервіс для визначення діагнозу на основі віку, статі та симптомів, після короткого опитування вертає список діагнозів, які підходять по заданим параметрам, та їх короткий опис. Цей сайт містить мало інформації про алгоритм роботи системи, тому тяжко судити про її точність. Інтеграція з медичною установою, та можливість отримати допомогу від спеціаліста відсутня.

1.3 Існуючі системи забезпечення електронного документообігу в поліклініках та стаціонарах

На даний момент в Україні більша чистина документації та обліку процесу проводиться в паперовому вигляді, це зумовлює багато незручностей та мішає ефективності цього процесу. В світі є багато прикладів схожих систем які працюють як на державному рівні так і в приватних стаціонарах. Для проведення аналізу вимог для системи було взято наступні інформаційні системи які використовуються в Україні.

Doktor eleks

Emcimed

Helsi

Функціонал цих систем було вивчено та проаналізовано. Серверна частина всіх систем написана за допомогою .net фреймворку. Всі вони включають в себе менеджмент не тільки лікарень а й поліклінік та облік продажу ліків. Оскільки ці системи надають дуже обширний функціонал для створення нового сайту потрібно зосередитеся тільки на певних основних аспектах обліку процесу

лікування в лікарні. Основними факторами що впливають на ефективність подібних систем являються:

легкість введення системи в заклад де такої не було;

простота інтерфейсу, наявність детальних посібників по користуванню;

доступність сайту на різних пристроях;

стабільність роботи та малі системні вимоги до серверу;

захищеність даних.

Висновки

Проаналізувавши існуючі системи, та ситуацію з потребою таких сервісів в світі, було вибрано основні задачі для системи, яка розробляється. Система має існуючі аналоги для кожної частини функціоналу, але існуючої системи яка б поєднувала в собі передбачення хвороб методами машинного навчання і можливість допомоги від живих лікарів не вдалось знайти. Загалом, всі існуючі системи передбачення хвороб мають суттєві недоліки, тому є можливість створити систему, яка буде перевершувати їх у певних аспектах.

РОЗДІЛ 2. ПРОЕКТНІ ТА ТЕХНІЧНІ РІШЕННЯ. ЗАБЕЗПЕЧЕННЯ ПРОЕКТОВАНОЇ СИСТЕМИ.

2.1 Інформаційне забезпечення проектованої системи

Для реалізації інтерфейсу проектної системи, було використано технології створення веб−сайтів: HTML, CSS, Bootstrap, javaScript, jQuery java, JSP, JSTL, Spring Boot, Spring Core, Spring Web, Spring Data JPA, Spring Security, Spring Testing, Junit, Mockito, Lombok, log4j, коннектор JDBC для MySql, javax servlet api, MySQL. Такі технології дозволяють виконати всі вимоги технічного завдання та значно полегшити розробку.

Для створення системи передбачення діагнозу за симптомами було створено модель нейронної мережі та розвернуто його на платформі хмарних обчислень AWS. Для цього було використано такі технології як: Tensorflow, Phyton, Amazon SageMaker, AWS Lambda, AWS API Gateway, AWS DynamoDB, AWS IAM, Amazon S3.

Далі будуть наведені основні відомості про технології та для чого вони були використані.

HTML (Hyper Text Markup Language) – Мова розмітки, описує структуру сайту. Використовує структуру тегів подібну до XML та використовується на більшості сайтів. Має багато різноманітних тегів для позначення різних частин сайту та його функціональних елементів. Підтримується в всіх сучасних браузерах та дозволяє переглянути базову версію сайту.

Для виконання стилізування HTML документів використовуються каскадні таблиці стилів (**CSS**) − це технологія описання форми вигляду та властивостей елементів сайту, які позначені в HTML. CSS дозволяє специфікувати кольори елементів сайту, разташування їх відносно один одного, форму та розмір. Такий опис документу дозволяє зробити сайт приємним для перегляду, інтуїтивно зрозумілим та виділити більш важливу інформацію. Крім цього CSS дозволяє опитмізувати відображення сайту на різних пристроях та деяку зробити динамічну реакцію на дії користувача.

Для створення дизайну сайту часто використовують набір інструментів **Bootstrap.** Цей фреймворк включає в себе багато створених класів для елементів, та задає їх поведінку завдяки javascript. Використання цього фреймворку дозволяє значно прискорити створення сайту, оскільки більшість його елементів будуть уже реаліозованими, програмісту залишається тільки коректувати їх вигляд, та добавляти неіснуючі елементи.

Для задання динамічної поведінки сайту зазвичай використовують мову програмування **javascript,** це дозволяє сторінці по різному реагувати на дії користувача, надає можливість краще взаємодіяти з ним. Скріпти javascript можуть міняти стилі елементів сторінки, добавляти нові елементи, відправляти асинхронні та синхронні запити на сервер та оновлювати сторінку за отриманою з сервера інформацією.

На практиці це реалізується привязуванням функцій скрипта на різні події браузера та елементів сторінки. Функції можуть взаємодіяти з DOM деревом документу або безпосередньо текстом HTML файлу. Завдяки цьому можна, наприклад показувати та ховати форму при нажиманні на кнопку, обновляти данні частково, так само це дозволяє проводити пейжинацію данних без оновлення всієї сторінки, а тільки відправляючи асинхронний запит для отримання потрібних данних і динамічно вставляти їх замість старих.

Для полегшення написання javascript коду буде застосовуватись бібліотека **jQuery**[19] - Її синтаксис розроблений, щоб зробити орієнтування у навігації зручнішим завдяки вибору елементів DOM, створенню анімації, обробки подій, і розробки AJAX-застосунків. jQuery також надає можливості для розробників, для створення плагінів у верхній частині бібліотеки JavaScript. Використовуючи ці об'єкти, розробники можуть створювати абстракції для низькорівневої взаємодії та створювати анімацію для ефектів високого рівня. Це сприяє створенню потужних і динамічних веб-сторінок.

Що до серверної частини на цьому веб сайті буде використовуватись сервер Tomcat з виконуваним кодом написаним на java з фреймворком Spring.

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 відправлення сторніки з данними на браузер буде застосовуватися технологія **JSP.**

JSP[20] (Java Server Pages) — технологія, що дозволяє веб-розробникам динамічно генерувати HTML, XML та інші веб-сторінки. Робота над JSP розпочалась в 1997 році. Згодом JSP було включено у склад Java EE — програмної платформи для програмування веб-додатків. Технологія дозволяє вставляти Java-код, в статичний вміст сторінки. Також можуть використовуватись бібліотеки JSP тегів для вставки їх в JSP-сторінки. Сторінки компілюються JSP-компілятором в сервлети, які є Java-класами, і виконуються на сервері. Сервлети також можуть бути написані розробником, не використовуючи JSP-сторінки. Ці технології доповнюють одна одну. Ця технологія позволить сформувати сторінку на сервері, а користувач отримає уже готовий HTML файл з потрібною інформацією.

Для полегшення написання JSP файлів буде використовуватись біблотека JSTL.

JSTL (JavaServer Pages Standard Tag Library) — розширення специфікації JSP яке добавляє бібліотеку JSP тегів для загальних потреб, таких як розбір XML даних, умовна обробка, форматування дати, створення циклів і підтримка інтернаціоналізації.

JSTL це альтернатива такому виду вбудованої в JSP логіки, як скріплети, тобто прямі вставки Java коду. Використання стандартизованної множини тегів бажаніше, оскільки отриманий код легше підтримувати и простіше відділяти бізнес-логіку от логіки відображення.

Для написання бізнес логіки та організації REST апі серверу використовується мова програмування java.

Java[21] — об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.

«Oracle» надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензією GNU General Public License.

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування.

Java, насамперед, – це мова, яка використовується для серверних додатків у великих корпораціях. Особливо часто Java використовується в банках, страхових компаніях, роздрібних мережах і т.д. Наприклад, такі банки, як Deutsche Bank, Citigroup, Barclays, Goldman Sachs і багато-багато інших, використовують Java для написання бек-енд і фронт-енд офісних електронних систем тощо.

Для полегшення виконання більшості тривіальних завдань створення сайту буде використовуватись фрейиворк **Spring.**

Spring Framework (скорочено Spring) - універсальний фреймворк з відкритим вихідним кодом для Java-платформи. Spring Framework широко поширений в Java-співтоваристві, незважаючи на те, що не забезпечує будь-яку конкретну модель програмування. Даний фреймворк надає Java-розробникам

набагато більше маневреності в проектуванні, плюс дає можливість використовувати добре документовані засоби вирішення проблем. Spring Framework застосовується в будь-якому Java-додатку, існує безліч розширень і удосконалень для побудови веб-додатків на Java.

Spring розділений на багато проектів, тому в конкретному сайті потрібно використовувати тільки деякі з них:

Spring boot – це корисний проект, метою якого є спрощення створення додатків на основі Spring. Він дозволяє найбільш простим способом створити web-додаток, вимагаючи від розробників мінімум зусиль по його настройці. Цей проект значно полегшує імпорт всіх інших проектів спрінга за допомогою збиральника **maven,** він забезпечує базову конфігурацію майже до всіх модулів тому відпадає потреба в самостійному їх конфігурувані.

Spring core – Забезпечує основну базу для реалізації всіх інших компонентів фреймворка. Забезпечує реалізацію SOLID паттерну інверсії контролю, завдяки реєстрації джава класів та автоматичної інєкції, яка конфігрується анотаціями та xml або java конфігурацією. Типова корпоративна програма на базі Java складається з декількох класів java. Для досягнення визначеної функціональності кожен клас (A.java) може залежати від одного або декількох інших класів. Ці інші класи відомі як залежності класу A. Як правило, кожен клас бере на себе відповідальність за отримання посилань класів, від яких він залежить. Це призводить до сильно звязаної архітектури.

Spring Framework допомагає розробити слабко поєднані звязані, делегуючи відповідальність за придбання залежностей класу до контейнера Spring та дозволяючи класу java зосередитись лише на визначеній ним функціональності. Контейнер Spring вводить залежності в клас java під час ініціалізації контейнера (як правило, при запуску програми.) Замість того, щоб клас java отримував свої залежності від контейнера, саме контейнер вводить залежності в клас java. Так відбувається інверсія контролю.

Spring MVC – забезпечує архітектуру паттерну Model — View — Controller за допомогою слабо повязаних готових компонентів. Паттерн MVC розділяє

аспекти застосунку (логіку введення, бізнес-логіку і логіку UI), забезпечуючи при цьому вільний звязок між ними**.**

На практиці це реалізовується завдяки диспетчер сервлету, який обробляє всі HTTP запити серверу і делегує виконання на інші сервіси та повертає отриманий результат у вигляді нової сторінки або JSON/XML відповіді. Це дозволяє розробляти сайт на більш абстрактному рівні та приділяти менше уваги конфігуруванню сервлетів.

Spring Security – надає механізми побудови систем аутентифікації та авторизації, а також інші можливості забезпечення безпеки для корпоративних додатків, створених за допомогою Spring Framework. Проект був розпочатий Беном Алексом (Ben Alex) в кінці 2003 року під ім'ям «Acegi Security», перший реліз вийшов в 2004 році. Згодом проект був поглинений Spring'ом і став його офіційним дочірнім проектом.

Spring data JPA – дозволяє легко реалізувати сховища на базі JPA. Цей модуль надає покращену підтримку рівня доступу до даних(DAO) яка базується на JPA. Це спрощує доступ до даних в додатках, що працюють на Spring.

Реалізація рівня доступу до даних у програмах була громіздкою протягом тривалого часу. Для виконання простих запитів, а також виконання сторінок і аудиту потрібно писати занадто багато кодових шаблонів. Spring Data JPA має на меті значно покращити реалізацію цього рівня за рахунок зменшення зусиль до фактично необхідної кількості. Розробники пишуть тільки інтерфейси сховища до якого потрібно отримати доступ а спрінг автоматично забезпечує реалізацію DAO класів. Найбільш популярною реалізацією JPA є **Hibernate,** тому він включений в Spring data JPA за замовчуванням

JPA (Java Persistence API) - стандартизований інтерфейс для Java ORM фреймворків. Є частиною EJB 3 та J2EE 5, хоча може використовуватись незалежно від них. Виник через популярність вільного ORM фреймворку Hibernate, та бажання мати незалежний від конкретної реалізації стандарт.

Hibernate[12] – засіб відображення між об'єктами та реляційними структурами (object-relational mapping, ORM) для платформи Java. Hibernate є

вільним програмним забезпеченням, яке поширюється на умовах GNU Lesser General Public License. Hibernate надає легкий для використання каркас (фреймворк) для відображення між об'єктно-орієнтованою моделлю даних і традиційною реляційною базою даних. Метою Hibernate є звільнення розробника від значних типових завдань із програмування взаємодії з базою даних. Розробник може використовувати Hibernate як при розробці з нуля, так і для вже існуючої бази даних.

Hibernate піклується про зв'язок класів з таблицями бази даних (і типів даних мови програмування із типами даних SQL), і надає засоби автоматичної побудови SQL запитів й зчитування/запису даних, і може значно зменшити час розробки, який зазвичай витрачається на ручне написання типового SQL і JDBC коду. Hibernate генерує SQL виклики і звільняє розробника від ручної обробки результуючого набору даних, конвертації об'єктів і забезпечення сумісності із різними базами даних.

Для прискорення написання програми та зменшення кількості бойлер плейт коду буде використовуватися бібліотека **Lombok**. Цей інструмент автоматично перетворює деякі анотації в джава код ще до початку компіляції, завдяки цьому відпадає потреба в реалізації методів які можуть бути написані автоматично, наприклад антоація @AllArgsConstructor повністю заміняє конструктор класу в якому задаються значення кожної змінної.

Під час написання програми, я планую керуватись методологією TDD.

JUnit – це фреймворк, розроблений для тестування програм, написаних з використанням технології Java. Він лежить в основі TDD (Test-Driven Development) і входить в сімейство фрейморков для тестування xUnit.

Головна ідея даного фреймворка - "спочатку тести, потім код". Це означає, що спочатку ми визначаємо, що повинно вийти в результаті роботи того чи іншого шматка коду і пишемо тести, які перевіряють ідентичність результату з потрібним, після чого пишемо сам шматок коду, який і будемо тестувати. Даний підхід збільшує ефективність роботи розробника і дозволяє писати більш

стабільний код. В результаті цього ми атимо менше часу на налагодження програми.

властивості JUnit:

- Фреймворк з відкритим вихідним кодом, який використовується для написання і виконання тестів.
- Дозволяє писати код швидше і якісно.
- Вкрай простий у використанні.
- Підтримує анотації для ідентифікації методів.
- Підтримує затвердження для тестування отриманих результатів.
- Тести можуть бути організовані в зв'язки тестів (test suites).
- Має візуальну індикацію стану тестів (червоні - не пройдено, зелені - пройдені).

Mockito[8] **–** надає ряд можливостей для створення заглушок замість реальних класів або інтерфейсів при написанні JUnit тестів.

Найбільшого поширення набули такі можливості Mockito:

- створення заглушок для класів і інтерфейсів;
- перевірка викликаючи методу і значень переданих методу параметрів;
- використання концепції «часткової заглушки», при якій заглушка створюється на клас з визначенням поведінки, необхідну для деяких методів класу;
- підключення до реального класу «шпигуна» spy для контролю виклику методів.

Spring Test – Модуль фреймворку спрінг, який забезпечує кращу інтеграцію з jUnit та Mockito, дозволяє інжектити біни в інтеграційних тестах та тестувати роботу основного контролера spring MVC.

Для легшого виправлення помилок, знаходження аномалій в роботі сайту та відновлення історії подій в разі якоїсь несправності потрібно вести логи роботи програми. Для цього буде використовуватися бібліотека **log4j.** Вона пропонує розділення логів по рівням важливості та працює за спільним з Slf4j інтерфейсом.

Для створення бази данних буде використовуватися СУБД **MySQL.**

MySQL –вільна реляційна система управління базами даних Розробка та підтримка сайта MySQL здійснює корпорація Oracle, яка отримала права на торговельну марку з поглиненої Sun Microsystems, яка раніше придбала шведську компанію MySQL AB. Продукт поширюється як під GNU General Public License, так і під власною комерційною ліцензією. Крім цього, розробники створюють функціональність за замовленням ліцензійних користувачів.

Гнучкість СУБД MySQL забезпечується підтримкою великої кількості типів таблиць: користувачі можуть вибрати як таблиці типу MyISAM, що підтримують повнотекстовий пошук, так і таблиці InnoDB, що підтримують транзакції на рівні окремих записів. Більш того, СУБД MySQL поставляється із спеціальним типом таблиць EXAMPLE, що демонструє принципи створення нових типів таблиць. Завдяки відкритій архітектурі і GPL-ліцензуванню, в СУБД MySQL постійно з'являються нові типи таблиць.

Tensorflow - це безкоштовна бібліотека програмного забезпечення з відкритим кодом для машинного навчання. Вона може бути використана для цілого ряду завдань, але приділяє особливу увагу навчанню глибоких нейронних мереж. TensorFlow пропонує кілька рівнів абстракції, тож можна вибрати відповідний для своїх потреб. Бібліотека дозволяє створювати та тренувати моделі за допомогою високорівневого API Keras, це дозволило швидко створити нейронну мережу, яка вирішує потрібну цій системі задачу.

Phyton[18] - загальноприйнята мова програмування високого рівня з великим пакетом стандартних бібліотек, основною ідеєю якої є читабельність та чіткість вихідного коду. Python підтримує як об’єктно-орієнтовані так і функціональні парадигми програмування. Він має повністю динамічну систему типу та автоматичне управління пам’яттю, подібну до Perl, Ruby, Scheme та Tcl. Як і інші динамічні мови, він часто використовується як мова сценаріїв. Інтерпретатори Python доступні для багатьох операційних систем. Стандартною реалізацією мови є CPython (написана на C), але є й інші, такі як Jython (написана

на Java), CLPython, написана на Common Lisp, IronPython (для .NET) та PyPy (написана на Python).

Amazon SageMaker - платформа хмарного машинного навчання, яка була запущена в листопаді 2017 року. SageMaker дозволяє розробникам працювати на різних рівнях абстракції під час навчання та розгортання моделей машинного навчання. На найвищому рівні абстракції, SageMaker пропонує ряд вбудованих алгоритмів ML, які розробники можуть тренувати на власних даних. Крім того, SageMaker надає керовані екземпляри TensorFlow та Apache MXNet, де розробники можуть створювати власні алгоритми ML з нуля. Незалежно від того, який рівень абстракції використовується, розробник може підключити свої моделі ML із підтримкою SageMaker до інших служб AWS, таких як база даних Amazon DynamoDB для структурованого зберігання даних, AWS Batch для офлайн пакетної обробки, або Amazon Kinesis для обробки в режимі реального часу

AWS Lambda - це обчислювальна технологія, яка дозволяє запускати код без забезпечення серверів та керування ними. Lambda запускає код на високодоступній обчислювальній інфраструктурі, та виконує всі задачі адміністрування обчислювальних ресурсів, включаючи обслуговування сервера та операційної системи, забезпечення пропускної здатності та автоматичне масштабування, моніторинг коду та ведення журналу. За допомогою Lambda можливо запускати код практично для будь-якого типу програми або серверної служби. Програмний код розбивається на лямбда функції. Lambda запускає функцію лише тоді, коли це потрібно, і масштабується автоматично, від кількох запитів на день до тисяч на секунду. Користувач платить лише за час обчислень, який він споживаєте, якщо код не працює, то плата за використання сервісу не стягується.

AWS API Gateway - це сервіс, який полегшує розробникам створення, публікацію, обслуговування, моніторинг та захист API в будь-якому масштабі. API виступають як "вхідні двері" для програм для доступу до даних, ділової логіки або функціональних можливостей серверних сервісів. Використовуючи

API Gateway, можна створювати RESTful API та WebSocket API, які дозволяють здійснювати двосторонній зв'язок мережею інтернет в режимі реального часу. Шлюз API підтримує контейнерні та безсерверні робочі навантаження, а також веб-додатки.

API Gateway обробляє всі завдання, пов'язані з прийняттям та обробкою до сотень тисяч одночасних викликів API, включаючи управління трафіком, підтримку CORS, авторизацію та контроль доступу, регулювання, моніторинг та управління версіями API. API Gateway не має мінімальних комісій або стартових витрат. Оплата провдится за отримані виклики API та кількість переданих даних.

AWS DynamoDB - служба баз даних NoSQL, яка підтримує структури даних ключ-значення та документи і пропонується Amazon.com як частина портфеля веб-служб Amazon. DynamoDB забезпечує швидку продуктивність при будь-якому масштабі. Це повністю керована багаторегіональна, багатоактивна, довговічна база даних із вбудованою системою безпеки, резервного копіювання та відновлення та кешування в пам’яті. DynamoDB може обробляти понад 10 трильйонів запитів на день і підтримувати пік понад 20 мільйонів запитів в секунду.

AWS IAM - це веб-служба, яка допомагає безпечно контролювати доступ до ресурсів AWS. IAM використовується щоб контролювати, хто має автентифікацію (увійшов в систему) та уповноважений (має дозволи) використовувати ресурси.

Amazon S3 - це технологія AWS, яка забезпечує зберігання об’єктів через інтерфейс веб-служби. Amazon S3 використовує таку саму масштабовану інфраструктуру зберігання, яку Amazon.com використовує для запуску своєї глобальної мережі електронної комерції. Amazon S3 можна використовувати для зберігання будь-яких типів об'єктів, що дозволяє використовувати такі способи, як зберігання для Інтернет-додатків, резервне копіювання та відновлення, аварійне відновлення, архіви даних, озера даних для аналітики та гібридне хмарне зберігання.

2.2 Структура сторінок проектованої системи

Структура сторінок веб−сайту та його вміст повинні відповідати вимогам, описаним в першому розділі. В процесі написання дипломної роботи було створено сторінки які будуть забезпечувати ефективну роботу проектованої системи.

Структура сайту складається з таких сторінок:

- Головна сторінка авторизації;
- Сторінка реєстрації;
- Сторінка створення запиту на передбачення хвороби системою
- Сторінка результату визначення хвороби
- Сторінка запиту на допомогу спеціаліста з діагностуванням
- Сторінка списку існуючих запитів на допомогу спеціаліста з діагностуванням
- Сторінка перегляду діагнозів пацієнта;
- Сторінка перегляду списку пацієнтів(доступна тільки для лікарів);

Ці сторінки поводяться по різному в залежності від ролі користувача який їх переглядає, на приклад, на сторінці перегляду діагнозу у лікарів є можливість добавляти нові медикаменти а пацієнти можуть тільки переглядати список уже приписаних медикаментів процедур та операцій. Також сторінка списку існуючих запитів на допомогу спеціаліста з діагностуванням буде відображати тільки запити, які створив користувач, якщо її переглядає пацієнт, і всі існуючі запити, якщо її переглядає лікар.

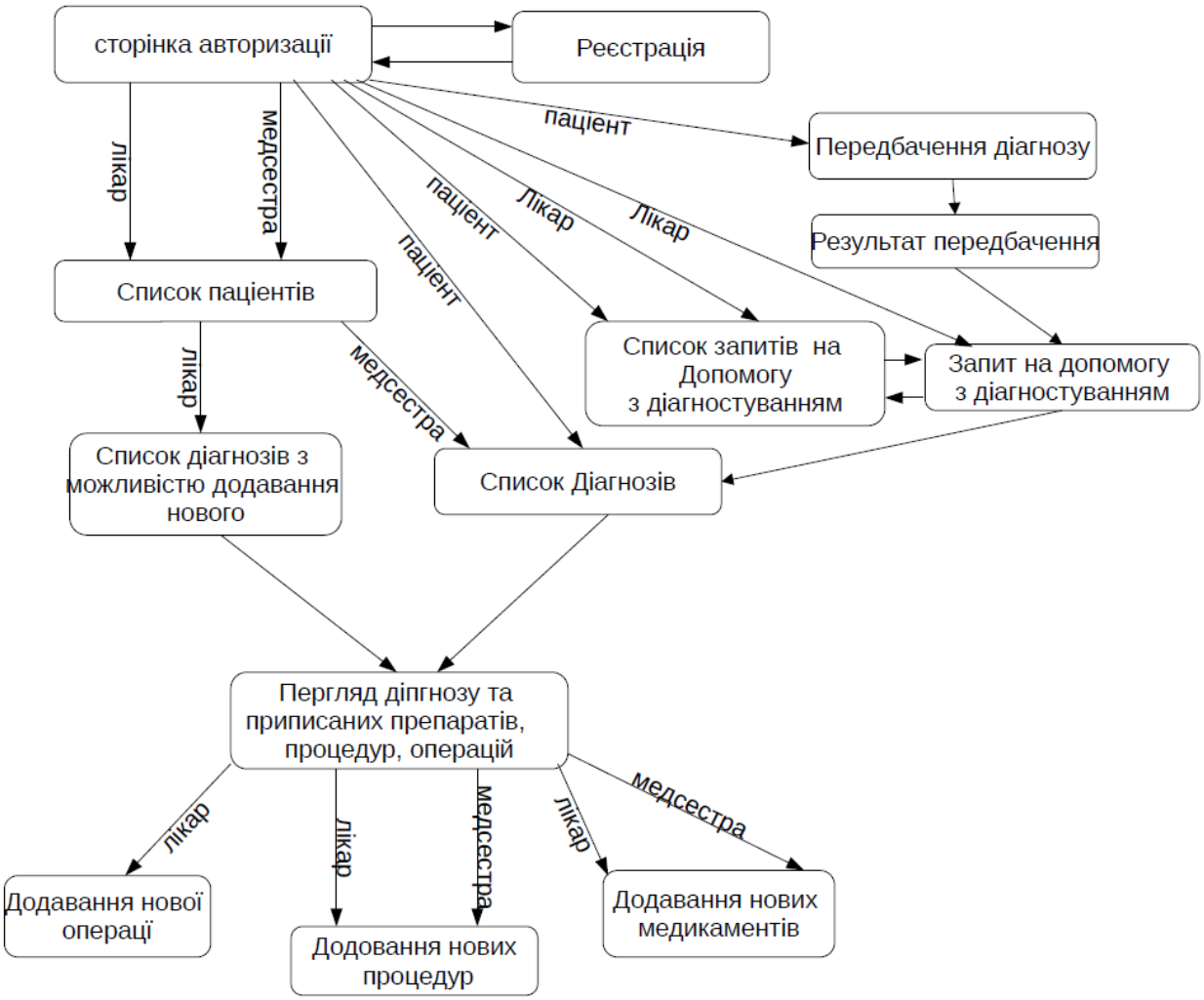

Рисунок 2.1. Структура сторінок веб–сайту «технологія визначення хвороб»

На сторінці запиту на допомогу спеціаліста з діагностуванням є можливість обговорення запиту, шляхом його коментування користувачем лікарями.

Також весь сайт локалізований на Українську та Англійську мову, користувачі можуть перемикати вибрану мову спінером в верхній правій частині сайту.

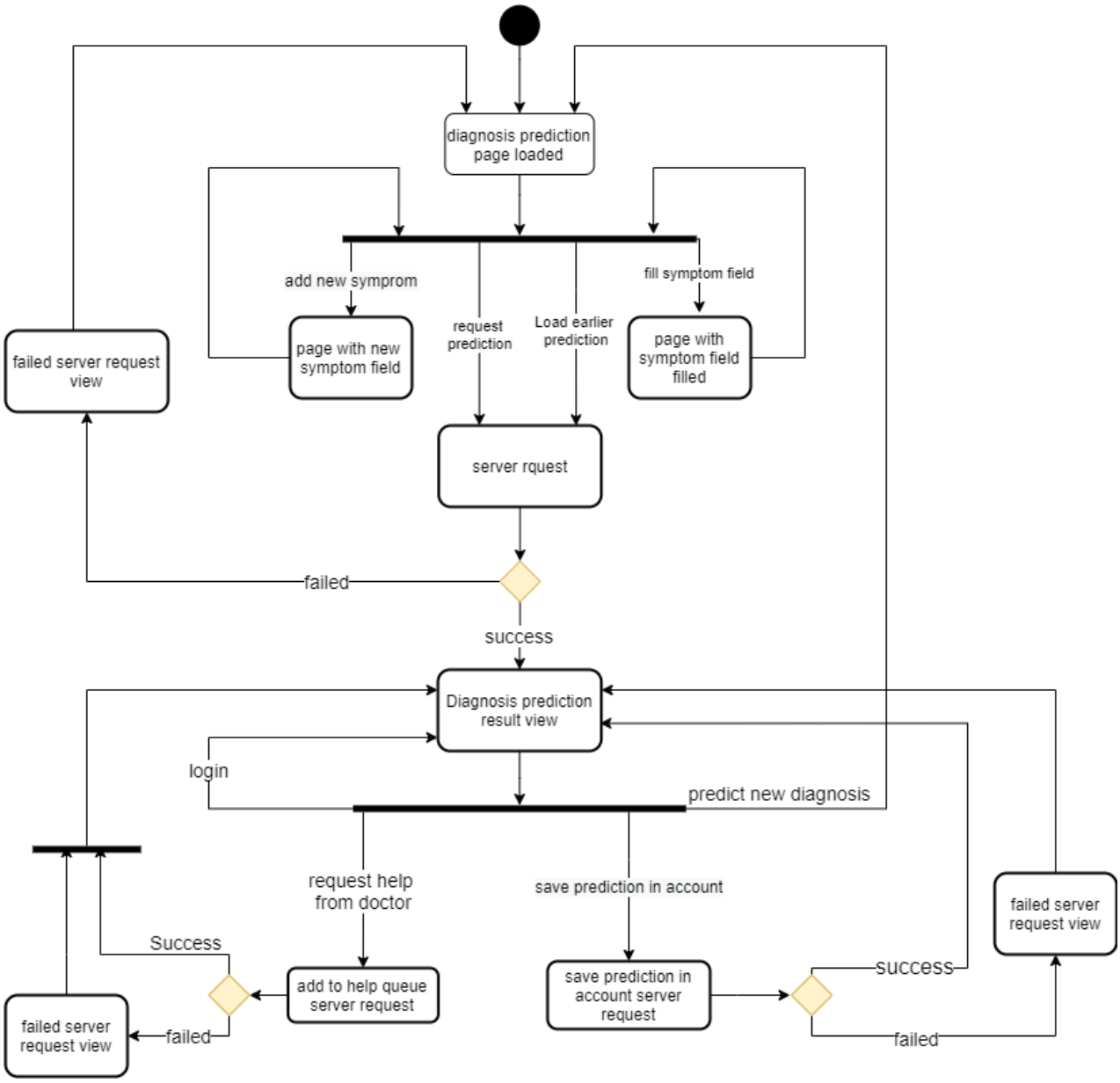

Рисунок 2.2 діаграма станів сторінки визначення хвороби

2.3 Структура сервісу для передбачення хвороб

Для реалізації алгоритму визначення діагнозу було використано платформу хмарних обчислень AWS та створено сервіс з безсерверною аріхтектурою. Цей сервіс повинен надавати можливість отримувати список доступних симптомів, діагнозів, отримувати переклад вибраного діагнозу або симптому на потрібну мову, визначати діагноз за переданими симптомами, зберігати результат діагностування пацієнта лікарем, для подальшого використання в покращенні роботи сервісу.

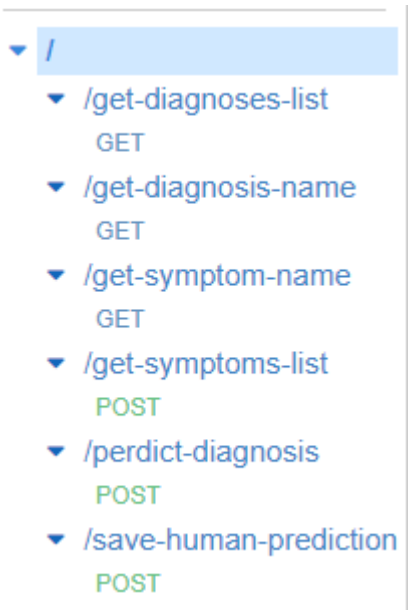

Рисунок 2.1. Список REST ендопоїнтів розробленого сервісу для визначення хвороб

Внутрішня структура цього сервісу включає в себе нереляційну базу даних DynamoDB та модель нейронної системи, яка розвернута за допомогою сервісу Amazon SageMaker endpoint. Доступ до цих систем організовано через використання AWS Lambda функцій, привязаних до веб ендпоїнтів сервісом AWS API Gateway.

2.4 Функціональні можливості проектованої системи

Система має забезпечувати основні функції для визначення діагнозів, їх обліку та збереження засобів для їх лікування.

Функціонал технології «Інформаційна технологія діагностування хвороб»

Таблиця 2.1

Опис функціоналу системи

Назва функціоналу	Опис функціоналу
Авторизація користувача	Для можливості ідентифікування користувачів та збереженням їх даних на сервері кожен користувач повинен пройти авторизації та підтвердити свої реєстраційні дані.
Локалізація сайту	Дозволяє перемикати мови на яких сайт відображається, перемикач доступний на верхній панелі сайту та зберігає своє значення незалежно від користувача завдяки використанню cookie файлів
Визначення діагнозу за симптомами	Основна функція системи, яка доступна всім користувачам. Дозволяє отримувати передбачення діагнозу за заданими симптомами, та ймовірність його коректності. Працює за допомогою машинного навчання нейронної мережі.
Перегляд результату визначення діагнозу	Сторінка яка показує результат визначення діагнозу системою, та пропонує подальші дії з цим результатом.

Продовження таблиці 2.1

Створення запиту на допомогу спеціаліста з діагностуванням хвороби	Можливість створювати в системі запит на допомогу кваліфікованого лікаря з визначенням хвороби по заданим параметрам. Доступна тільки авторизованим користувачам системи.
Коментування запиту на допомогу з діагностуванням хвороби	Функція доступна тільки автору запиту та персоналу лікарні, дозволяє створювати додаткові коментарі та спілкуватися з лікарем, для надання більш точної інформації та ефективного визначення хвороби.
Створення діагнозу на запиту на допомогу з діагностуванням хвороби	Функція, яка доступна тільки лікарям, дозволяє на основі інформації наданої пацієнтом в запиті, та коментарів до запиту поставити остаточний діагноз, та зберегти його в системі.
Пейжинація даних	Розбиває всі данні які відображаються у вигляді списку на сторінки, завдяки цьому користувач не повинен завантажувати всі доступні записи. Оскільки даних може бути багато, це значно збільшує швидкість оновлення сторінок, та зменшує навантаження на сервер. Користувач може вибирати кількість записів на сторінці, та номер сторінки яка зараз відображається.

Продовження таблиці 2.1

Перегляд списку запитів на допомогу з діагностуванням хвороби	Функція, яка доступна авторизованим користувачам, дозволяє переглядати список створених запитів на допомогу з діагностуванням, та переходити на основну сторінку вибраного запиту для отримання більш детальної інформації та коментування або виставлення остаточного діагнозу. Користувачі з рівнем доступу пацієнт можуть переглядати тільки свої запити. Персонал лікарні може переглянути всі існуючі в системі запити.
Пергляд списку пацієнтів	Функція яка доступна тільки для персоналу лікарні, дозволяє отримати список всіх пацієнтів, та перейти до перегляду їх діагнозів.
Пергляд діагнозів	Функція яка дозволяє переглядати діагнози пацієнта, Користувачі з рівнем доступу пацієнт можуть переглядати тільки діагнози приписані їм, тоді як персонал лікарні може переглядати діагнози любого користувача.
Додавання нових діагнозів	Функція яка доступна тільки лікарям, дозволяє додати до профілю пацієнта новий діагноз.

Продовження таблиці 2.1

Перегляд діагнозу	Функція яка дозволяє переглянути деталі конкретного діагнозу, пацієнти можуть переглядати деталі тільки своїх діагнозів, а персонал лікарні може переглядати деталі діагнозу любого пацієнта.
Закриття діагнозу	Функція, яка доступна тільки лікареві, дозволяє закрити вилікуваний діагноз, після цього до нього неможна додавати нові засоби лікування, в списку діагнозів з’являється дата його закриття. Кнопка закриття знаходиться на сторінці перегляду деталей діагнозу
Перегляд медикаментів	Дозволяє переглядати медикаменти які приписані до діагнозу, пацієнти можуть переглядати тільки медикаменти своїх діагнозів, а працівники лікарні мають доступ до даних всіх пацієнтів.
Перегляд деталей медикаменту	Оскільки в таблиці приписаних медикаментів відображається тільки скорочена інформація, кнопка показати більше розгортає запис і сторінка починає відображати прив’язані деталі та email лікаря який зробив це призначення

Продовження таблиці 2.1

Перегляд операцій	Дозволяє переглядати операції які приписані до діагнозу, пацієнти можуть переглядати тільки медикаменти своїх діагнозів, а працівники лікарні мають доступ до даних всіх пацієнтів.
Перегляд деталей операції	Оскільки в таблиці приписаних операцій відображається тільки скорочена інформація, кнопка показати більше розгортає запис і сторінка починає відображати прив’язані деталі та email лікаря який зробив це призначення
Перегляд процедур	Дозволяє переглядати процедури які приписані до діагнозу, пацієнти можуть переглядати тільки медикаменти своїх діагнозів, а працівники лікарні мають доступ до даних всіх пацієнтів.
Перегляд деталей процедури	Оскільки в таблиці приписаних процедур відображається тільки скорочена інформація, кнопка показати більше розгортає запис і сторінка починає відображати прив’язані деталі, повний список дат на які призначена процедура та email лікаря який зробив це призначення

Продовження таблиці 2.1

Додавання нового медикаменту	Доступне тільки для персоналу лікарні, дозволяє додавати нові медикаменти до діагнозу. Викликається кнопкою з вікна відображення медикаментів,
Додавання нової процедури	Доступне тільки для персоналу лікарні, дозволяє додавати нові процедури до діагнозу, до процедури можна прив’язати декілька дат, на які вона буде назначена. Викликається кнопкою з вікна відображення процедур,
Додавання нового медикаменту	Доступне тільки для лікарів лікарні, дозволяє додавати нові операції до діагнозу. Викликається кнопкою з вікна відображення операцій,

2.5 Схема бази даних веб серверу

Для виконання бакалаврської роботи було створено декілька таблиць:

- Користувачі
- Діагнози
- Запити на допомогу з діагностуванням
- Симптоми в запитах на допомогу з діагностуванням
- Коментарі до запитів на допомогу з діагностуванням
- Медикаменти
- Хірургічні операції
- Процедури
- Дати на які назначені процедури

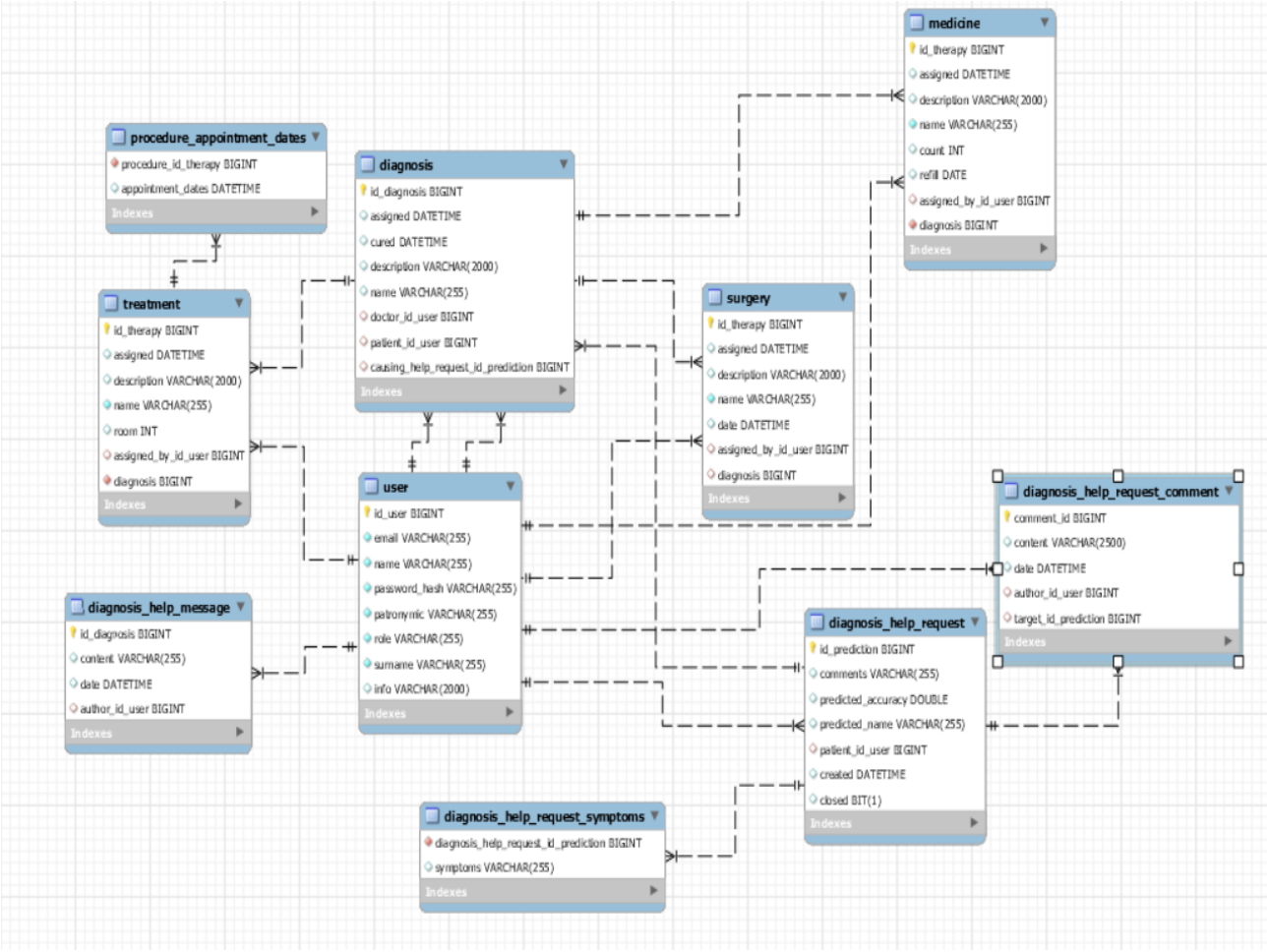

Рисунок 2.2 Схема бази даних

2.6 Опис бази даних веб серверу

Весь доступ на базу даних відбувається через фреймворк Spring data JPA через технологію ORM Hibernate. Всі назви та типи таблиць і полів в базі даних відповідають таким в сутностях джава програми, завдяки цьому Hibernate автоматично пов’язує їх з даними в базі даних.

Користувачі мають текстове поле роль, яке мапится на енам в джава коді і забезпечує розділення дозволів користувачів. Пацієнти зв’язані зв’язком один до багатьох з своїм діагнозами, а лікарі з діагнозами які вони поставили. До діагнозів зв’язками один до багатьох прив’язані таблиці медикаментів, операцій, та процедур. У кожної операції, медикаменту та процедури є посилання на користувача який її приписав, це також забезпечує зв'язок багато до одного з персоналом лікарні.

Крім таблиці процедур існує допоміжна таблиця дат записів на процедури вона містить записи дат процедур зв’язані зв’язком багато до одного до записів процедур і при отриманні об’єкту процедури через ORM данні з цієї таблиці маплятся в масив дат на які призначена дана процедури.

База даних запускається незалежно від серверу завдяки сервісам MySQL. Усі текстові поля бази даних записуються в кодуванні utf 8 що забезпечує підтримку додаткових символів та багатьох мов світу.

2.7 Схема бази даних хмарного сервісу для визначення хвороб

Крім бази даних веб сайту, системі потрібно зберігати список всіх можливих симптомів та діагнозів. Оскільки сервіс передбачення в майбутньому може використовуватися не тільки основним сайтом, було вирішено використати мікросервісну архітектуру, де сервіс визначення діагнозу за симптомами буде окремим застосунком розвернутим на хмарі. Для зберігання потрібних даних він використовує нереляційну базу даних DynamoDB. Для роботи сервісу потрібно три таблиці, одна для збереження списку симптомів, їх перекладів на різні мови, і синонімів для швидшого пошуку, друга містить таку саму інформацію для діагнозів. Остання таблиця містить записи про затверджені лікарем діагнози, та використовується для покращення точності нейронної мережі.

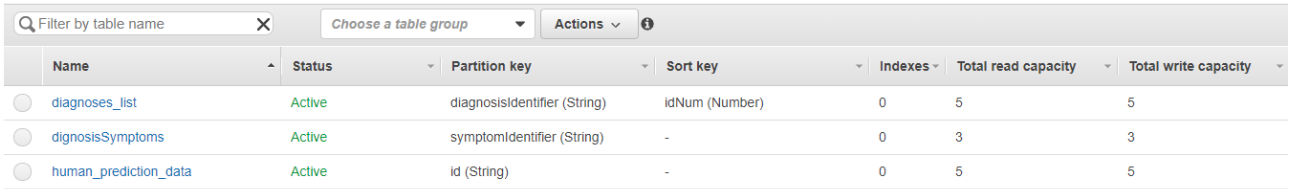

Рисунок 2.3 Список таблиць DynamoDB

2.2.1 Опис структури серверу веб-сайту

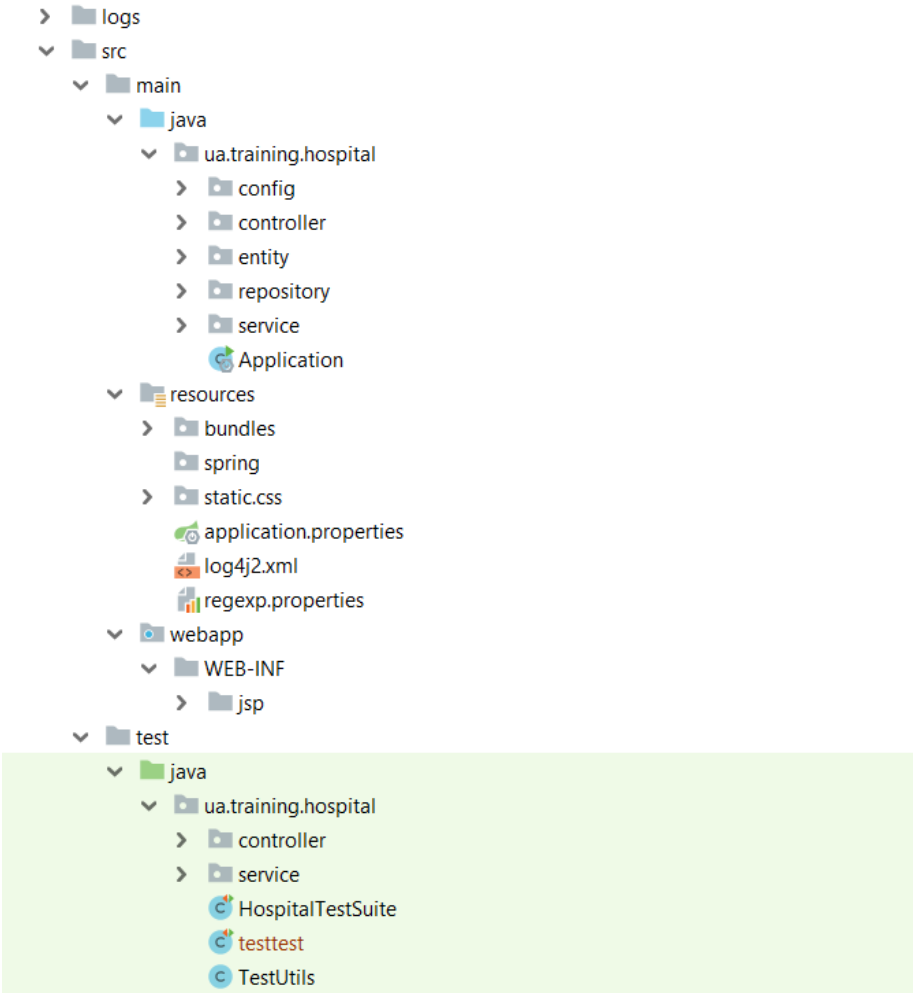

Рисунок 2.1 структура проекту

Проект розділений на декілька папок та пакетів. Проект розділений на 2 частини: main та test. Main включає в себе сам сайт, його веб сторінки, конфігураційні файли та класи серверної частини. Test – включає в себе набір тестів, для перевірки правильності написаного коду, та полегшення введення нових змін.

Опис системних папок веб−сайту «Інформаційна технологія діагностування хвороб»

Таблиця 2.1

Опис структури проекту

Назва папки	Опис
logs	Папка куди зберігаються логи сайту
main/java.ua.training.hospital.config	Пакет з класами джава конфігурації web mvc і spring security.
main/java.ua.training.hospital.controller	Пакет з контроллерами веб сторінок, містить підпакети з DTO обєктами та утилітами валідації введених данних на стороні серверу.
main/java.ua.training.hospital.entity	Тут зберігаються класи для основних обєктів системи які напряму мапляться в базу данних, або обєктів які містять менше інформації та створенні для зменшення навантаження при вибірці даних.
main/java.ua.training.hospital.repository	Пакет в якому знаходиться основна JPA конфігурація ORM мапінгу бази даних.
main/java.ua.training.hospital.service	Пакет з сервісами для реалізації бізнес логіки застосунку.

Продовження таблиці 2.1

main/resources/bundles	Папка з .properties файлами в яких зберігається весь текст інтерфейсу сайту перекладений на доступні користувачам мови (Українську та Англійську).
main/resources/static.css	Папка з css стилями сайту.
main/webapp/WEB-INF/jsp	Папка з jsp файлами сторінок в яких знаходиться html розмітка сайту та javascript код
test/java.ua.training.hospital	Пакт з юніт тестами основних модулей системи.

Висновки

Для створення даної системи краще використовувати сучасні технології створення веб сайтів там хмарних сервісів. Проаналізувавши вимоги до системи було вибрано розробляти веб сайт з використанням мови java та її фреймворку Spring, як веб сервер вибрано Apache Tomcat. Сам сервіс для визначення діагнозу з використанням машинного навчання розвернутий на платформі хмарних обчислень AWS, та комунікує з основним веб сервером через мережу інтернет. Вибраний стек технлогій має забезпечити безпроблемне виконання всіх поставлених перед системою завдань.

РОЗДІЛ 3. ОПИС РОБОТИ СИСТЕМИ.

3.1 Запуск та встановлення системи

Для запуску веб серверу на комп’ютері чи сервері повинні бути встановлені: maven, java8 та СУБД mySQL. Перш ніж запускати систему необхідно ініціалізувати базу даних. Це можна зробити двома способами, або використавши існуючу схему бази даних та наперед заповнивши її данними з дампу іншої бази, або використавши автогенерування схеми за допопомогою технології Hibernate. Для цього потрібно просто створити пусту модель та змінити конфігурацію в файлі «src\main\resources\application.properties» з «spring.jpa.hibernate.ddl-auto=none» на «spring.jpa.hibernate.ddl-auto=update». Також в цьому файлі потрібно вставити посилання на базу даних та логін інформацію для користувача з дозволом на читання та запис від імені якого система буде вносити зміну в базу данних.

Сам сайт запускається за допомогою фреймворку автомтаизації збірки maven. Для цього потрібно відкрити в терміналі папку з проектом та ввести команду «mvn spring-boot:run», Після цього виконається перевірка залежностей, завантаження відсутніх пакетів та запуск серверу.

Сервіс визначення хвороб працює на платформі хмарних обчислень AWS, для доступу до нього використовються веб ендпоїнти, при зміні їх адреси треба змінити конфігурацію в файлі «src\main\resources\application.properties».

3.2 Інструкція користувача

Веб-сайт працює по різному з різними типами користувачів. Для доступу до любих сторінок крім сторінки передбачення діагнозу користувачу потрібно авторизуватись. Тому до того як користувач зайде, любе посилання в межах сайту крім сторінки реєстрації, логіну, та передбачення діагнозу за симптомами буде перенаправлене на сторінку логіну (http://localhost:8080/login).

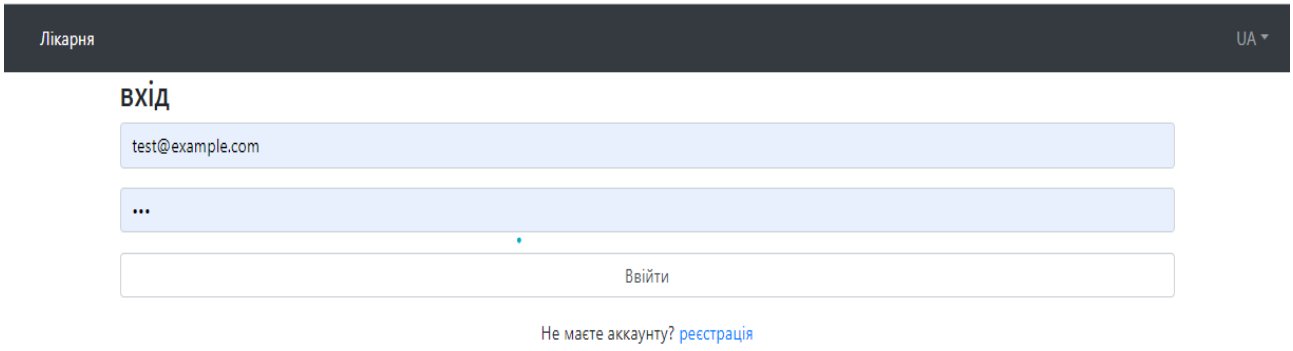

Рисунок 3.1 сторінка входу

В разі неправильно введених даних сторінка сповістить користувача про помилку.

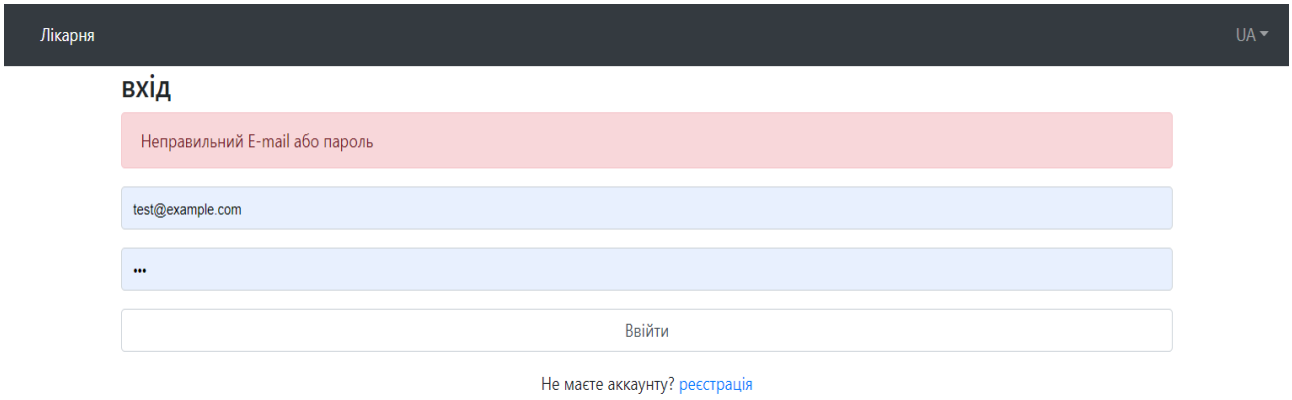

Рисунок 3.2 сторінка входу після спроби логіну з неправильними даними

Якщо користувач ще не має облікового запису в системі йому потрібно зареєструватися, для цього на сайті існує сторінка:

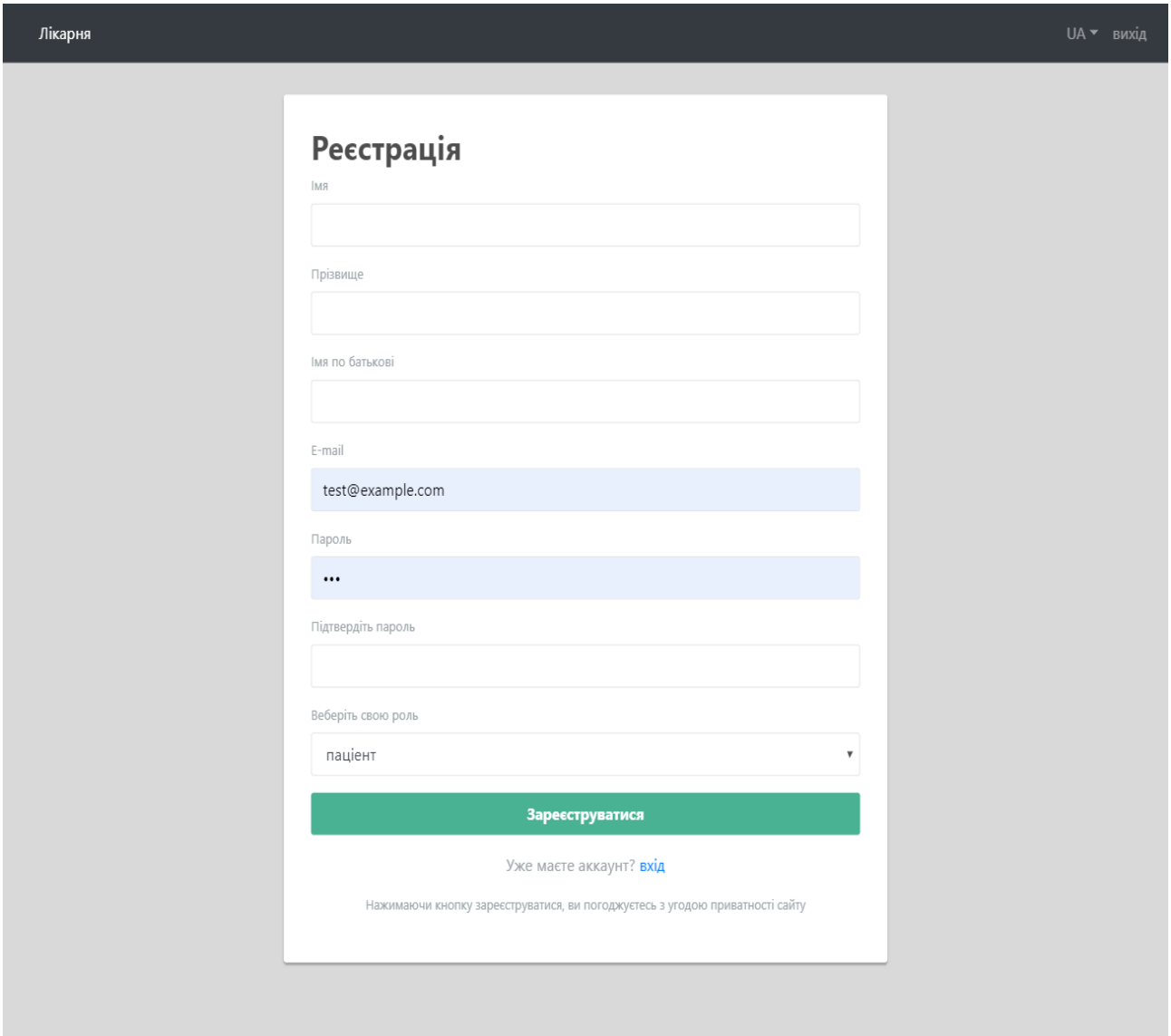

Рисунок 3.3 сторінка реєстрації

Користувач може в любий момент змінити мову, на якій буде відображатися сайт, для цього йому треба вибрати її в селекторі, що знаходиться в правому верхньому куті сайту. Як на стороні користувача так і на сервері відбувається перевірка введених даних на валідність, імейл має відповідати вимогам стандарту RFC 5322, пароль має мати довжину більше за 6 символів та включати в себе хоча б одну цифру та латинську літеру. Правильність імен перевіряється відносно деяких граматичних правил, наприклад імя має починатися з великої літери та включати в себе символи тільки однієї мови.

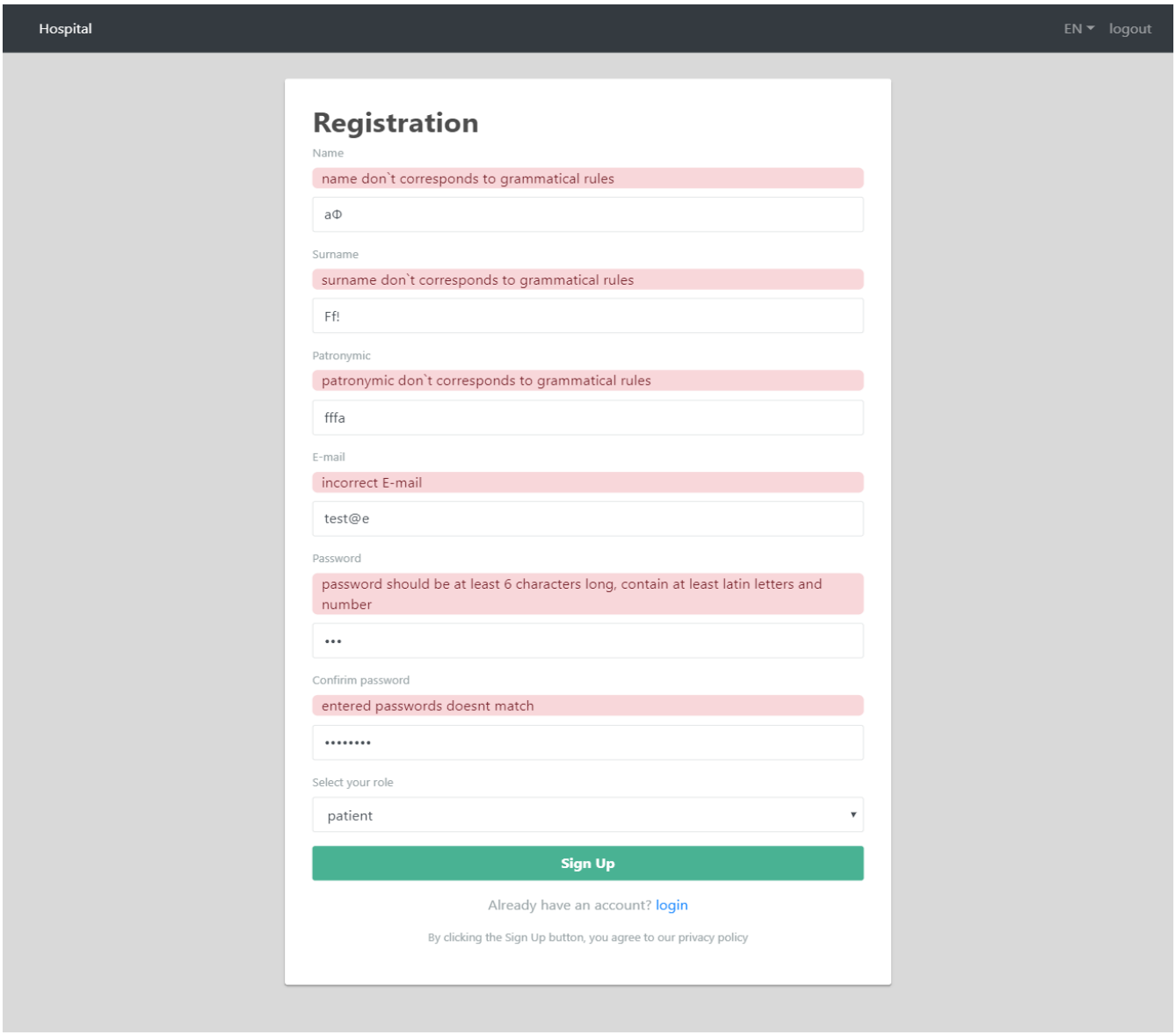

Рисунок 3.4 сторінка реєстрації на англійській мові, та показаними попередженнями про неправильне введення

Після логіну користувача перенаправляє на сторінку на яку він намагався зайти, якщо це сторінка логіну, сторінка якої не існує в системі, або в користувача немає прав для перегляду цієї сторінки, він буде пере направлений на базову сторінку. Для персоналу це сторінка перегляду списку пацієнтів (http://localhost:8080/patientsList), а для пацієнтів це сторінка перегляду їхніх діагнозів (http://localhost:8080/patient(id)). Також на базову сторінку можна попасти натиснувши на лого сайту в лівій верхній частині вікна.

Також навіть для не авторизованих користувачів доступна сторінка для передбачення діагнозу за симптомами (http://localhost:8080/diagnosis-prediction).

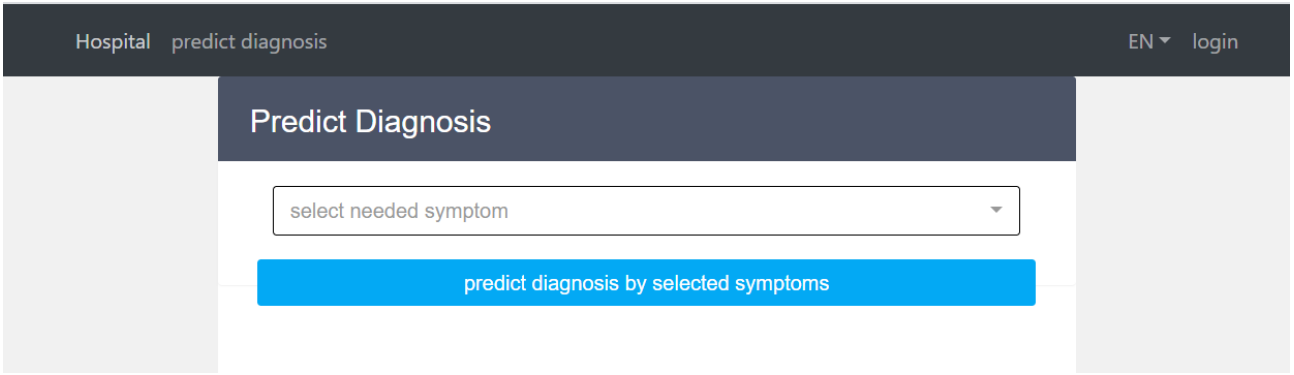

Рисунок 3.5 сторінка передбачення діагнозу

На цій сторінці можна вибрати декілька симптомів. Симптоми для пошуку беруться з сервісу передбачення хвороб, тому є завжди актуальними. Для пошуку симптомів можна почати вводити їх назву в любій з доступних мов.

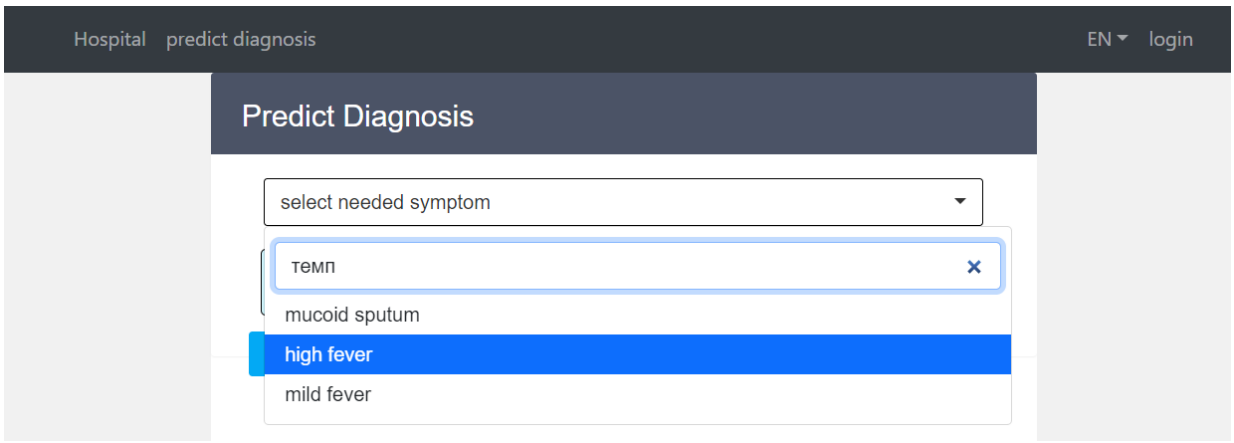

Рисунок 3.6 пошук по симптомам

Після вибору декількох симптомів можна натиснути копку отримання результату

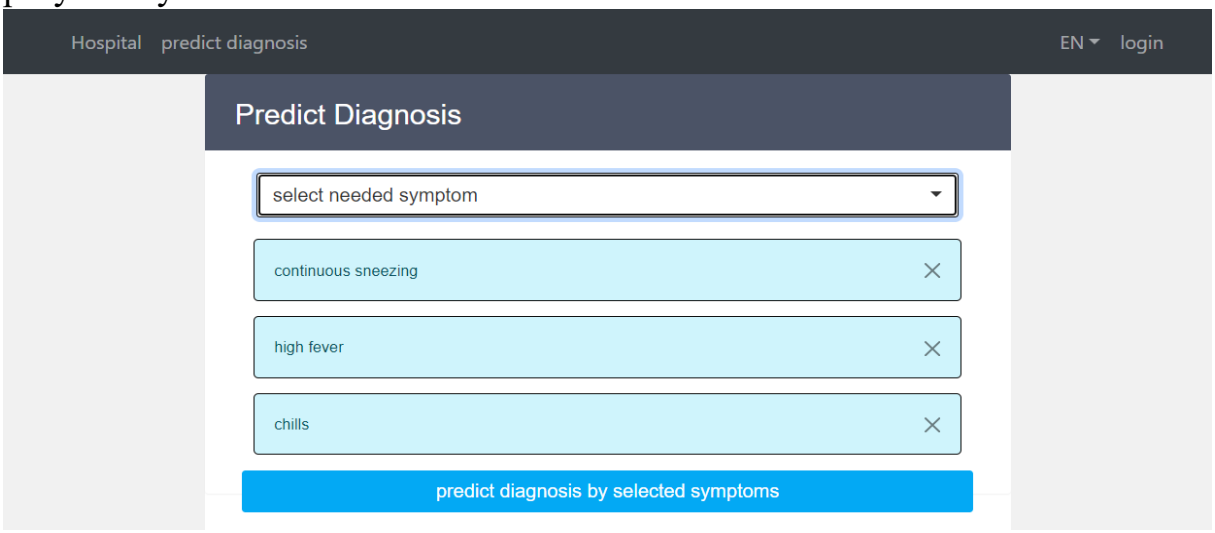

Рисунок 3.7 сторінка передбачення хвороби з декількома вибраними симптомами

Після кліку на кнопку отримання результату відкриється сторінка з результатом передбачення

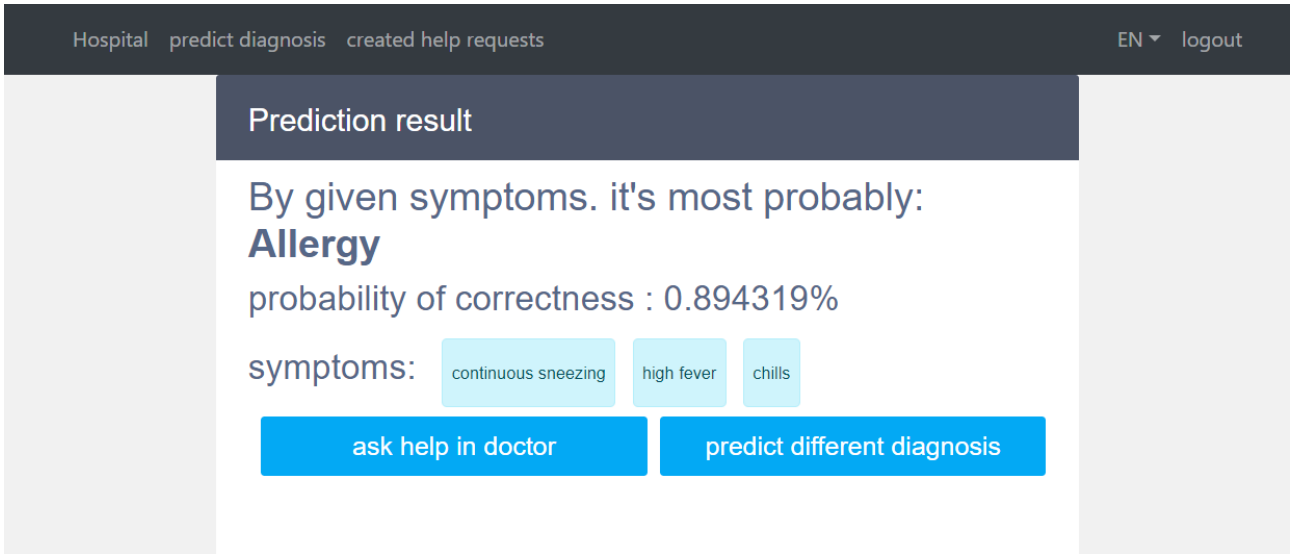

Рисунок 3.8 сторінка з результатом передбачення за вибраними симптомами

На цій сторінці можна створити запит на допомогу з діагностуванням у спеціаліста, або натиснути кнопку «передбачити інакший діагноз», щоб вернутись на сторінку з вибором симптомів.

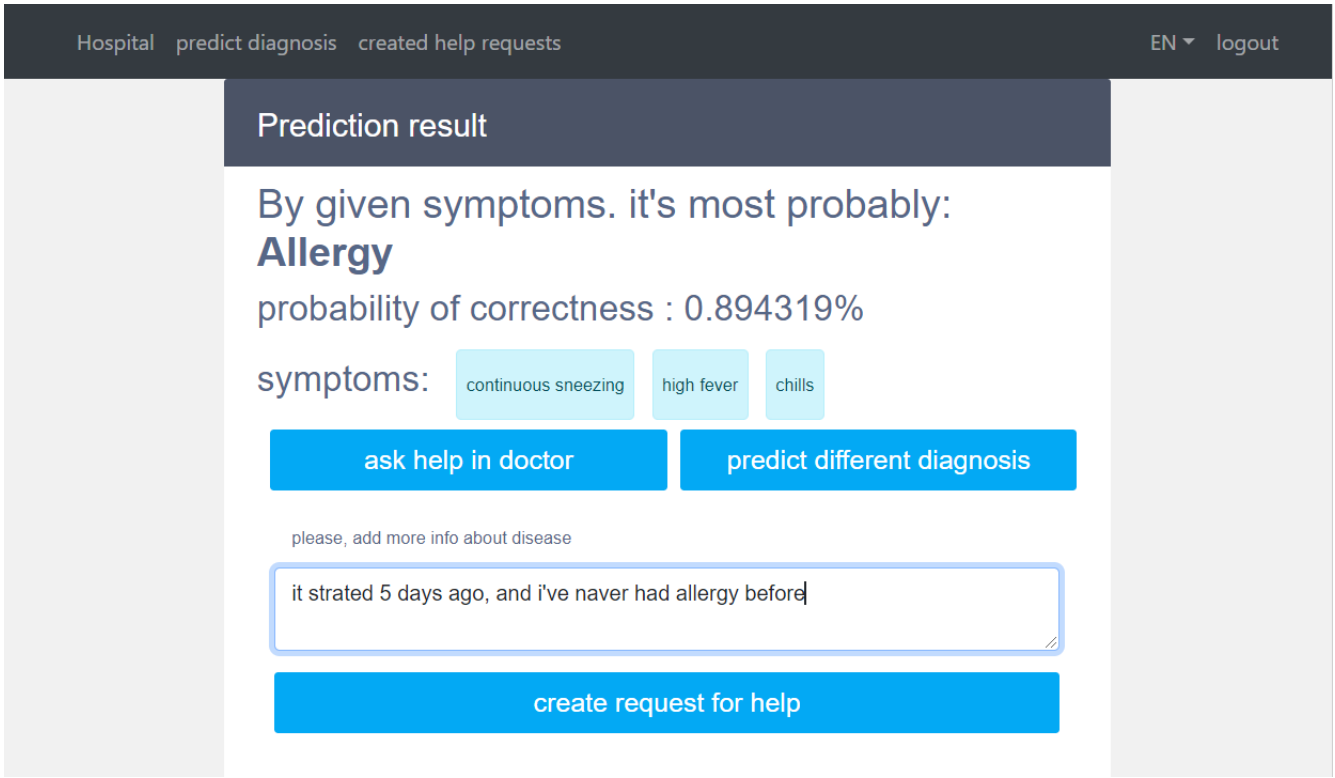

Рисунок 3.10 створення запиту на допомогу з діагностуванням хвороби

Після кліку на кнопку «створити запит на допомогу з діагностуванням» відкриється сторінка запиту, на якій можна додавати коментарі, та міняти список симптомів.

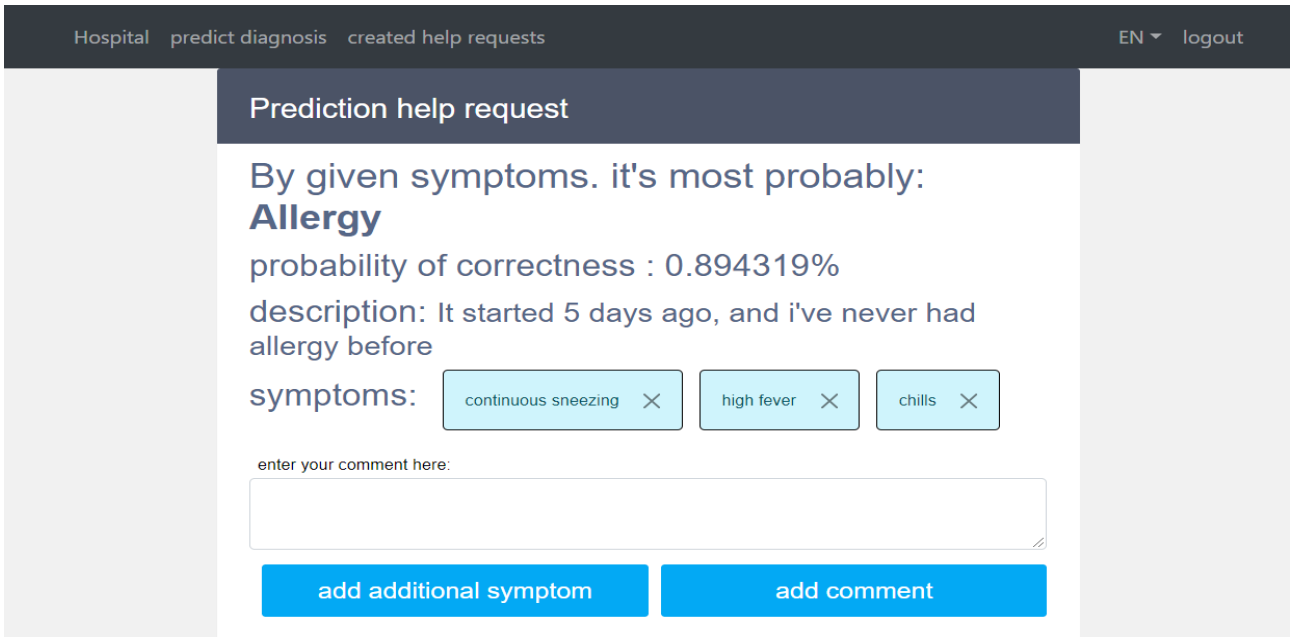

Рисунок 3.11 сторінка допомоги з визначенням діагнозу

Браузер автоматично оновляє список коментарів, тому надає змогу лікарю та пацієнту спілкуватися в реальному часі

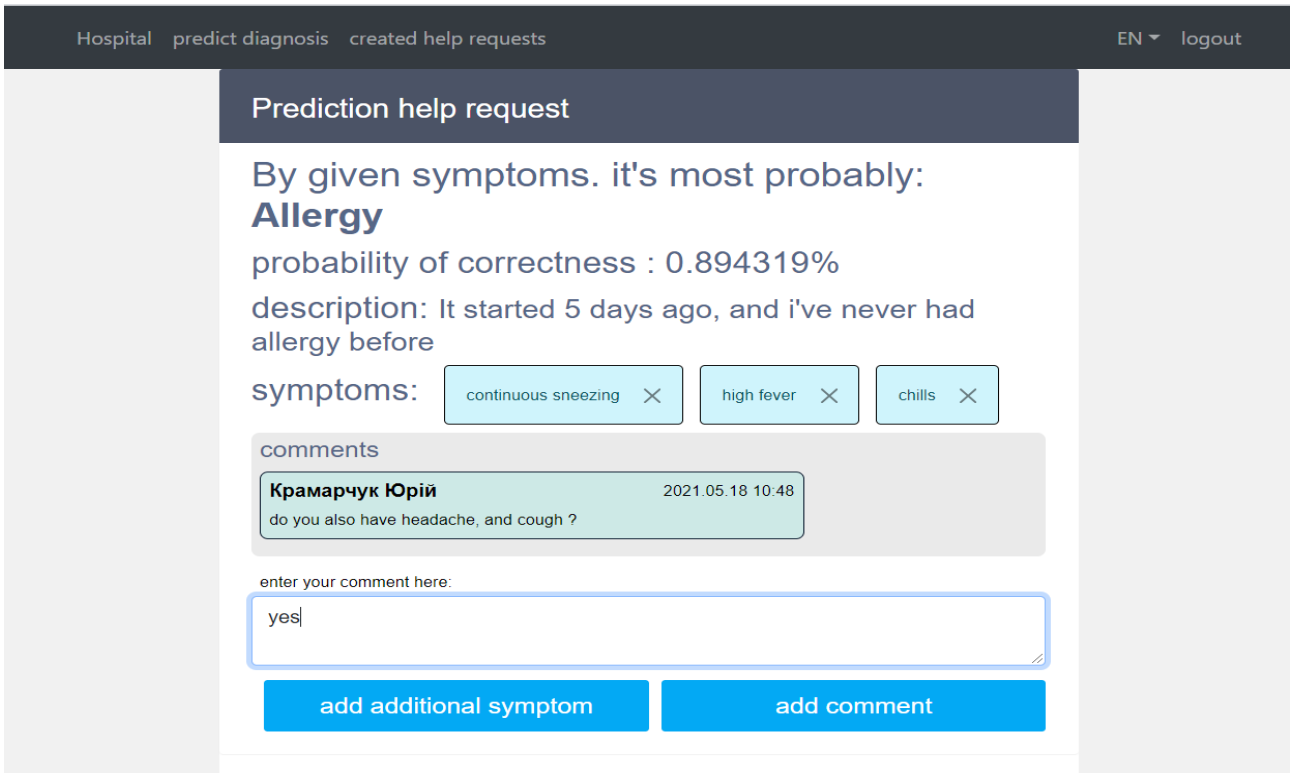

Рисунок 3.12 процес коментування запиту на допомогу з діагностуванням

Також персонал медичного закладу та пацієнт, який відкрив запит може видаляти та додавати нові симптоми. Для того, щоб зробити це, потрібно натиснути на кнопку «додати додатковий симптом», після цього з’явитися поле для вибору нового симптому. Неактуальні симптоми можна видаляти натиснувши на кнопку «х» біля назви симптому.

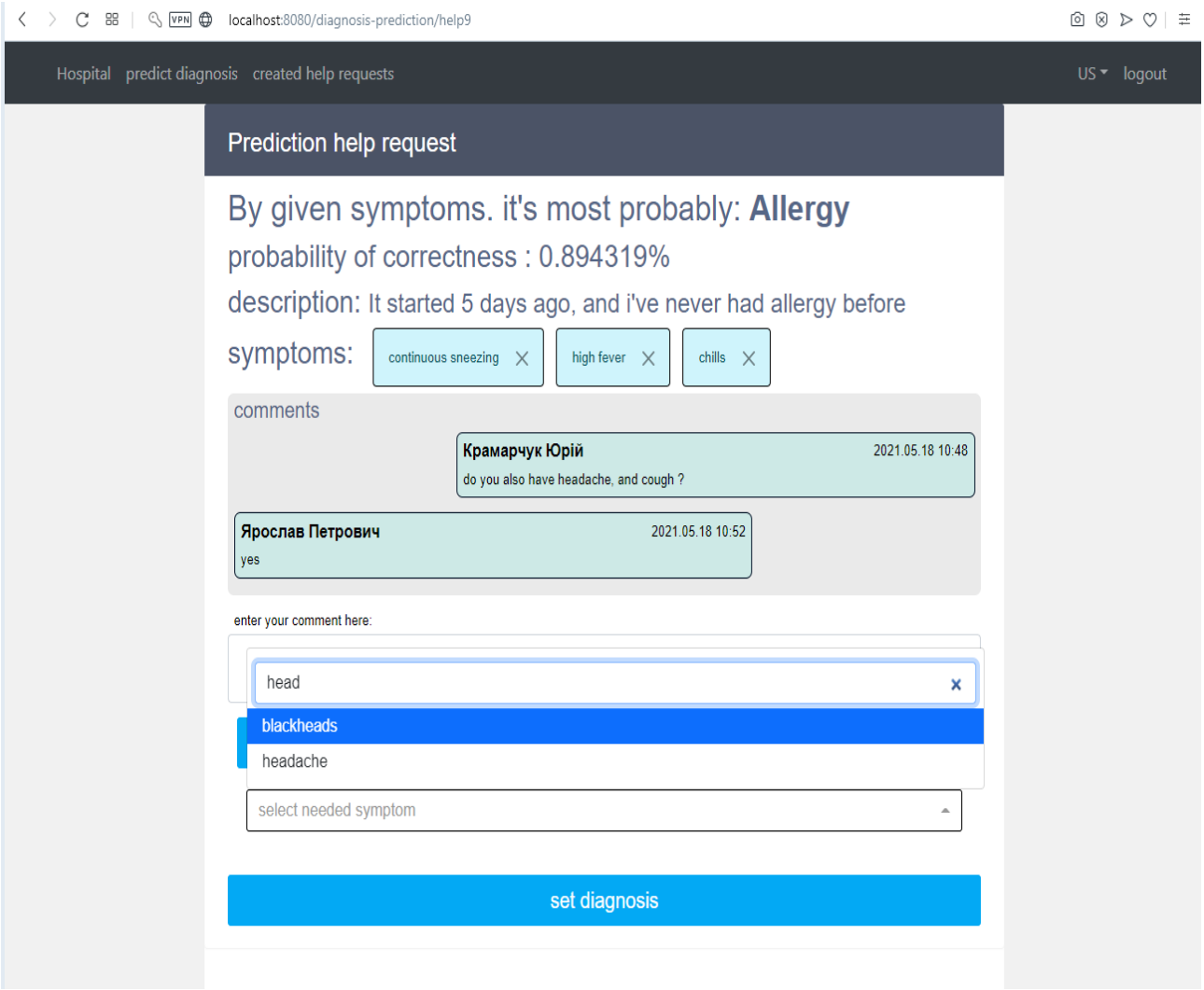

Рисунок 3.13 додавання нового симптому

Для того, щоб поставити остаточний діагноз, лікарю потрібно натиснути кнопку «встановити діагноз» в результаті цього відкриється форма для вводу інформації про хворобу. В ній необхідно вибрати назву діагнозу, та за потреби ввести його опис. Після повторного кліку на кнопку «встановити діагноз», діагноз збережетеся, а користувача перенаправить на сторінку перегляду інформації про діагноз.

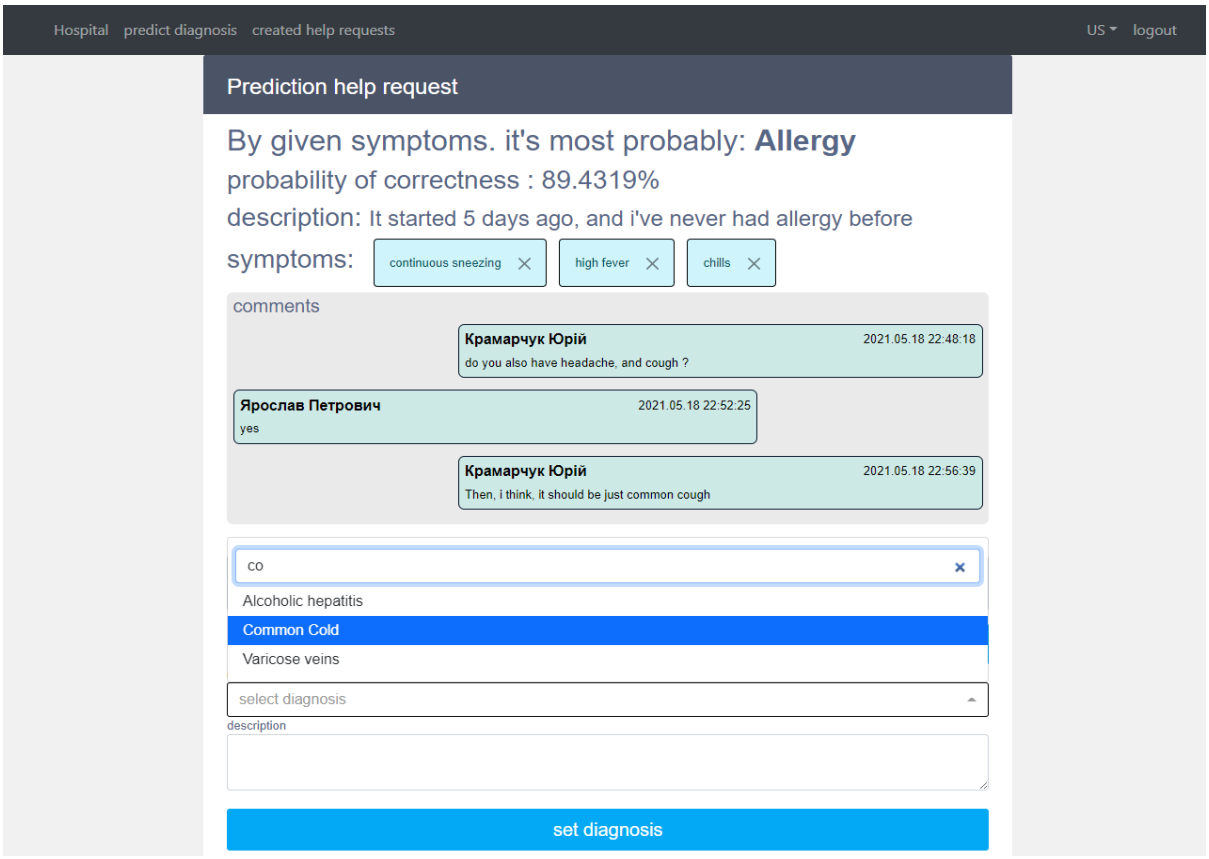

Рисунок 3.14 вибір потрібного діагнозу

Створені запити на допомогу з діагностування можна переглянути настиснувши на посилання «створені запити на допомогу» на навбарі. На відкритій сторінці будуть відображатися всі запити створені в системі, якщо її відкрив співробідник медичної установи. Пацієнтам відображаються тільки запити ствоені ними.

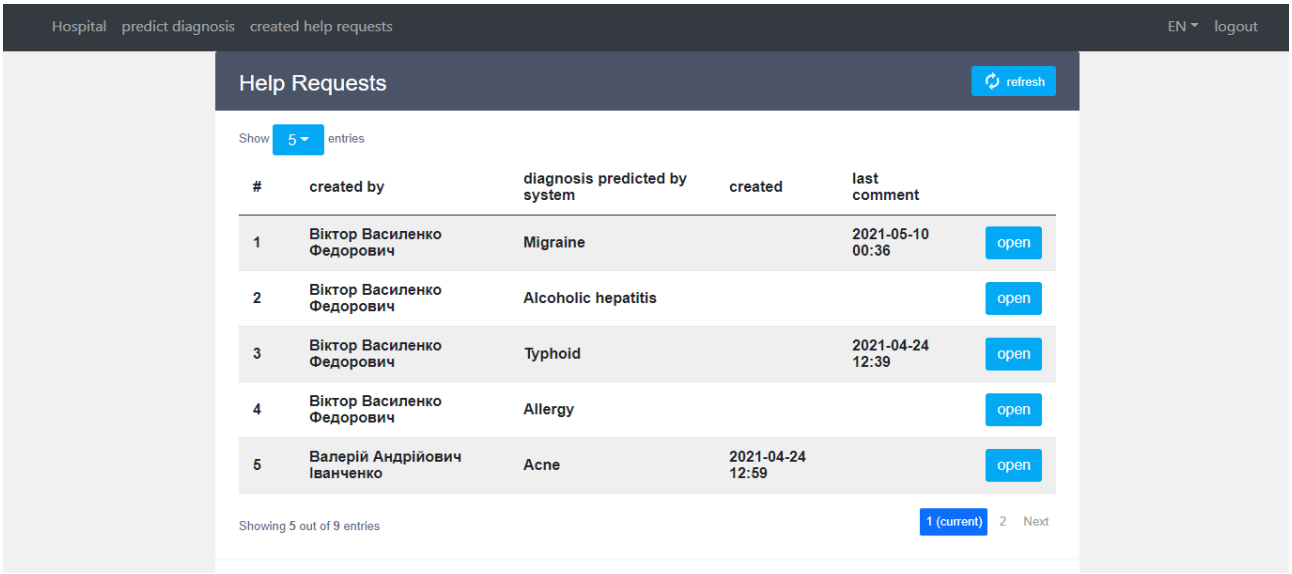

Рисунок 3.15 перегляд запитів на допомогу з діагностування

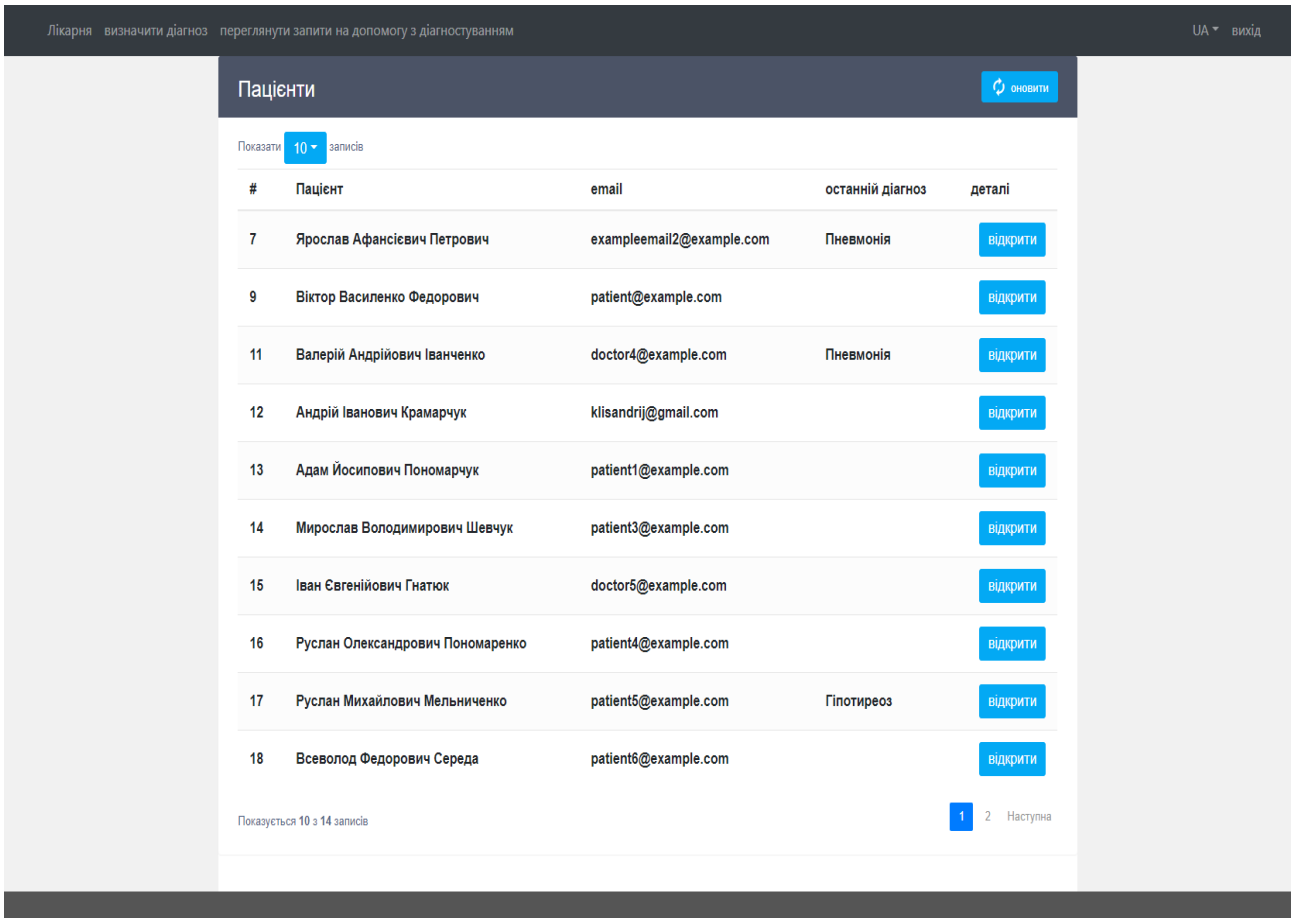

Рисунок 3.6 список пацієнтів

Ця сторінка доступна тільки для персоналу сайту, після кліку на кнопку “відкрити” відкривається сторінка перегляду діагнозів пацієнта.

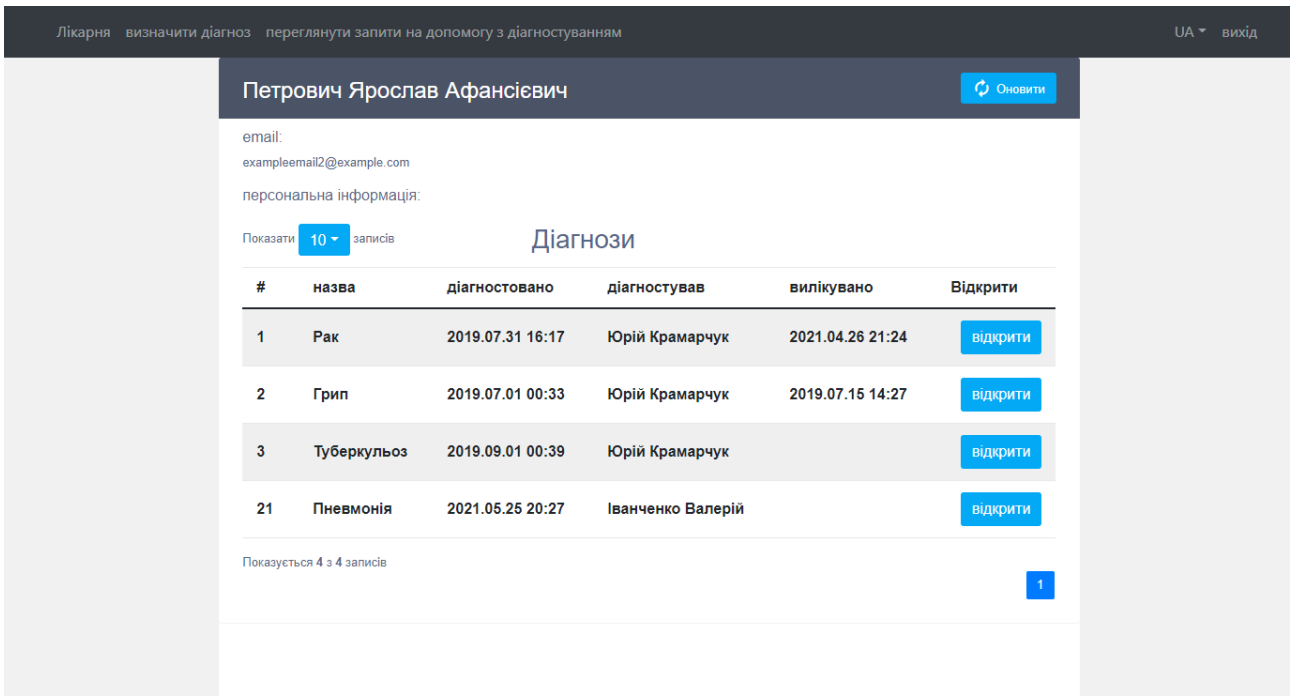

Рисунок 3.7 сторінка перегляду власних записів пацієнта

Сторінка перегляду пацієнта лікарем, виглядає так само як перегляд власних діагнозів пацієнтом, за виключенням того що відображається кнопка додати діагноз. Після кліку на цю кнопку з’являється вікно додавання діагнозу.

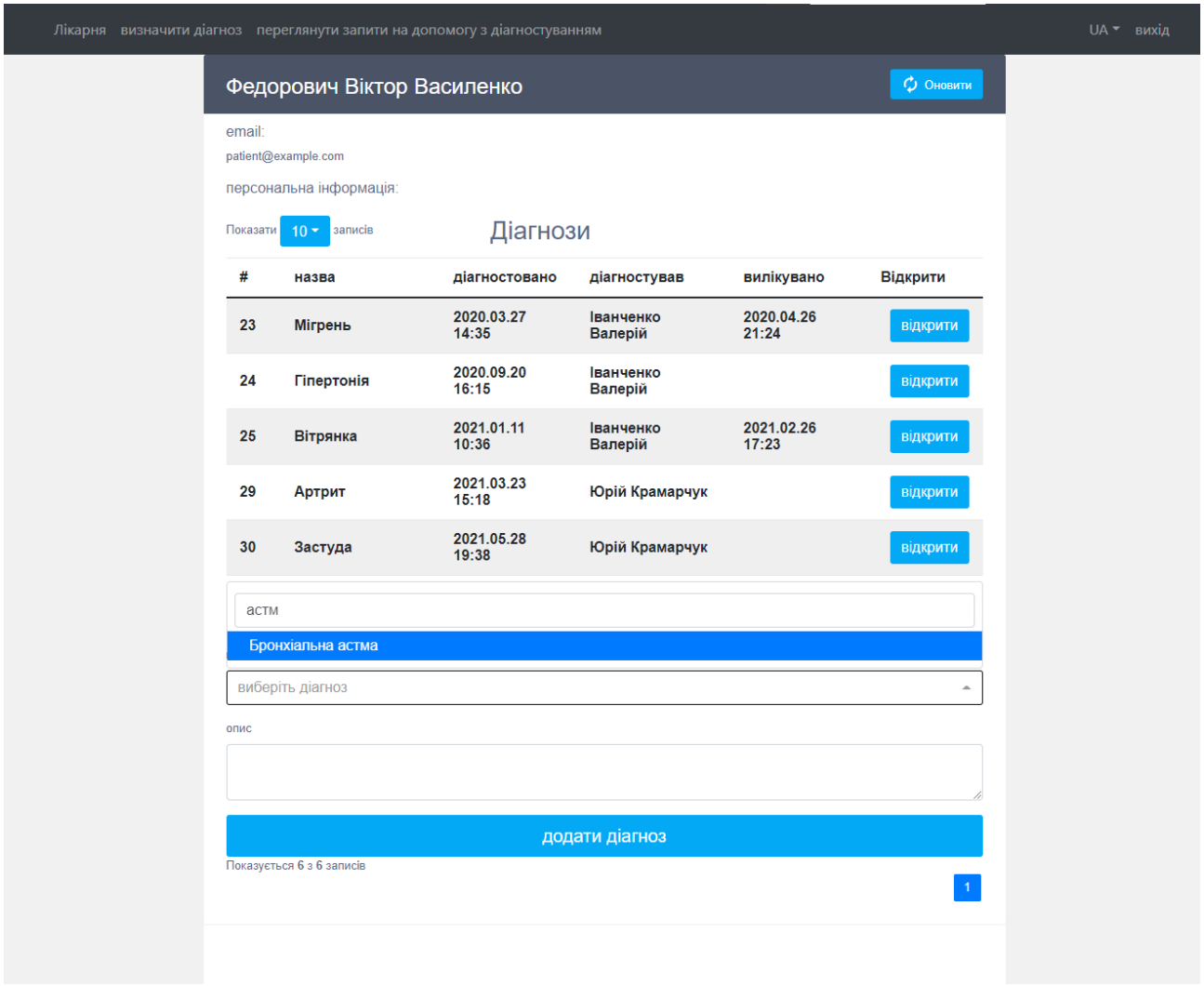

Рисунок 3.8 Сторінка перегляду пацієнта лікарем

Після кліку на кнопку «відкрити» відкривається сторінка перегляду діагнозу, на якій відображаються деталі діагнозу, всі види лікування які до нього відносяться, можна добавляти нові медикаменти, процедури та операції. Всі списки спочатку появляються закритими та починають відображатися після кліку на кнопку. На відображуваних списках можна змінювати кількість записів які показуються за раз та перемикати сторінку.

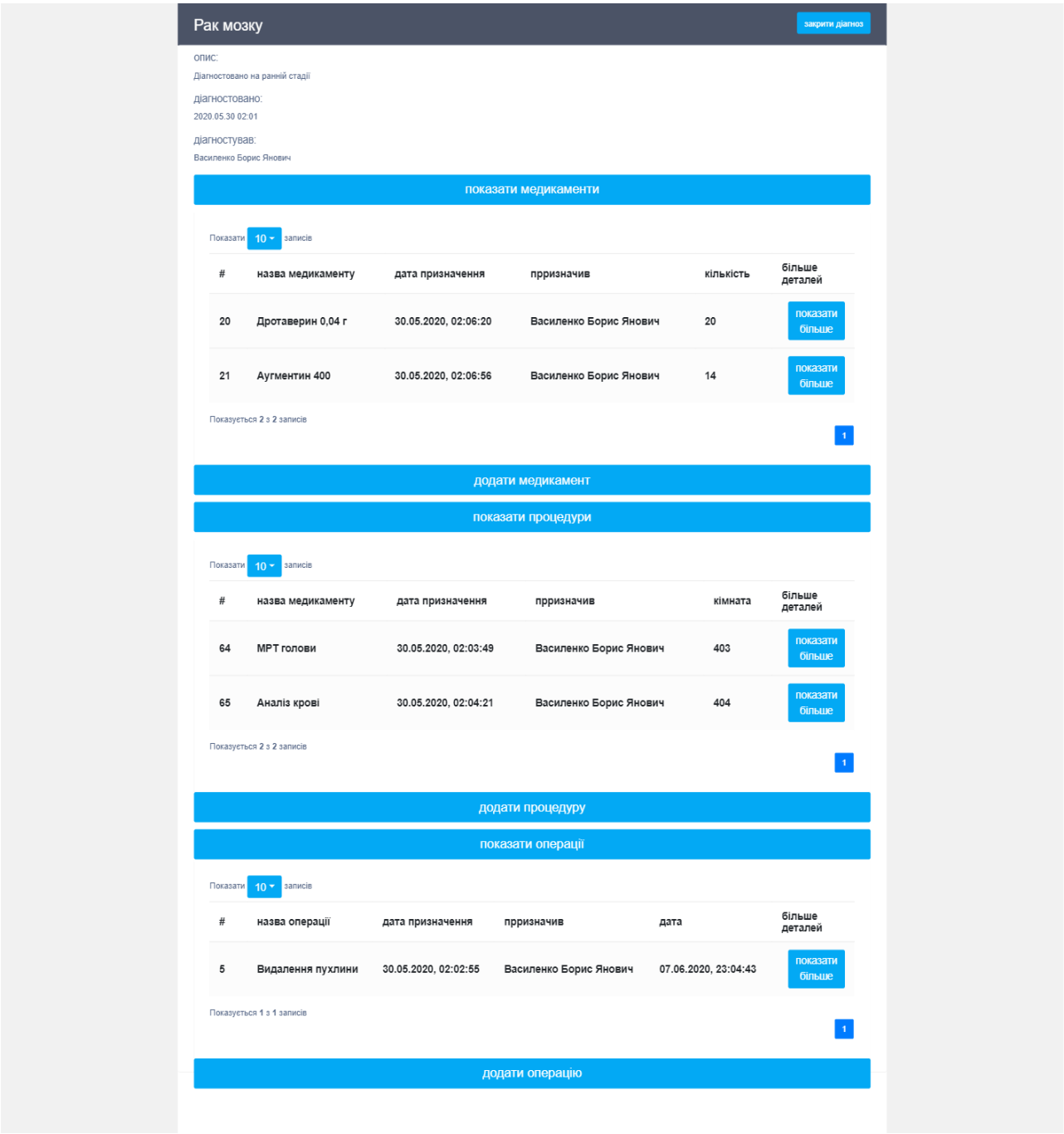

Рисунок 3.9 сторінка перегляду діагнозу лікарем

Коли лікар пробує приписати нові медикаменти відкривається вікно схоже до додавання нового діагнозу, в яке можна записати деталі. Остаточне збереження нового медикаменту відбувається після повторного натиску на кнопку «додати медикамент». Код контролеру сторінки перегляду діагнозу наданий в додатку А

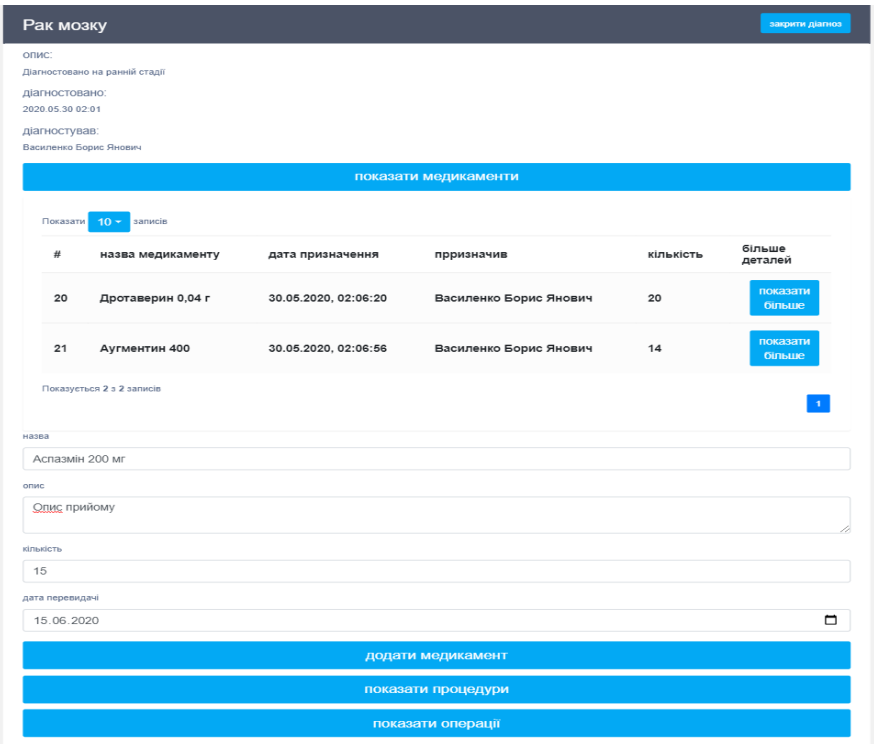

Рисунок 3.10 Вікно додавання нового медикаменту

Перегляд сторінки діагнозу пацієнтом виглядає так само як для лікаря за виключенням того що кнопки додавання нових лікувань та закриття діагнозу відсутні.

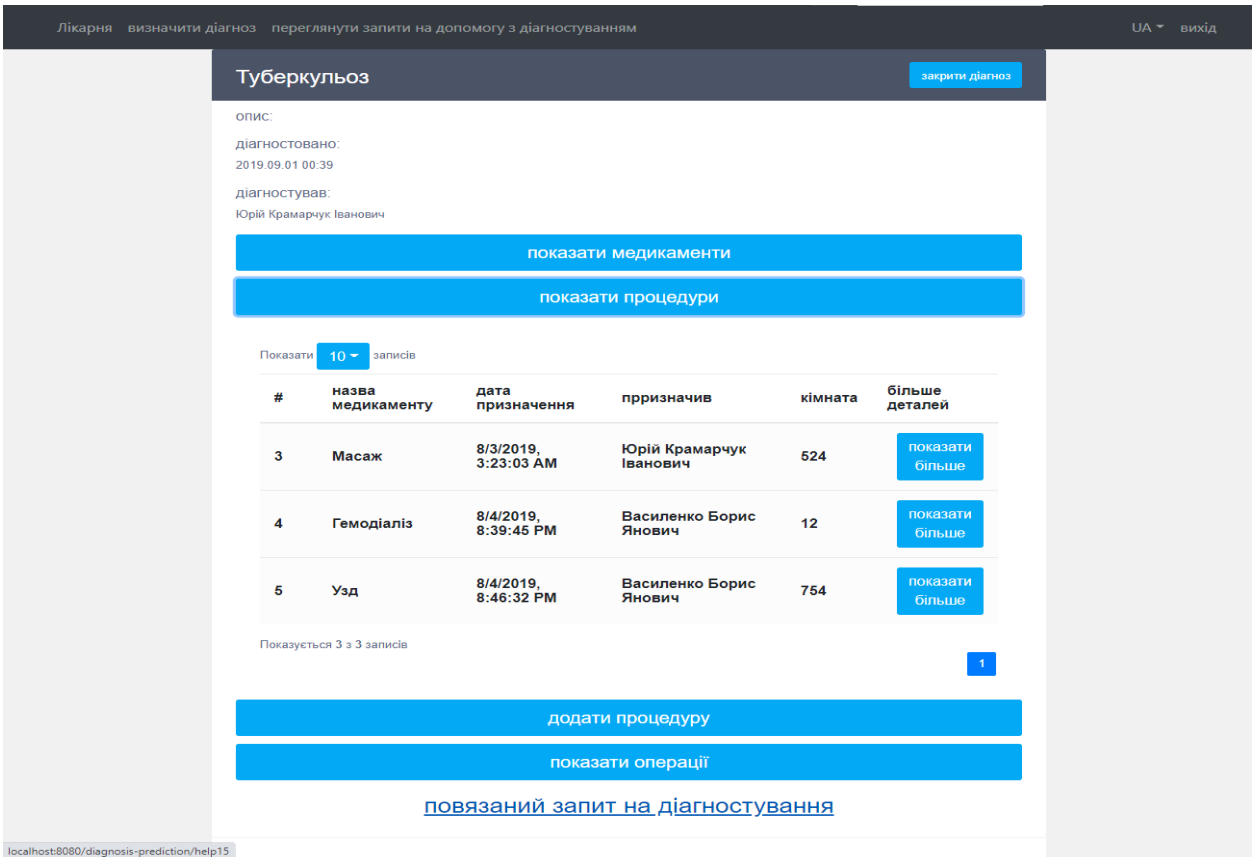

Рисунок 3.11 Сторінка перегляду діагнозу пацієнтом

Сторінка перегляду діагнозу також містить в собі посилання на запит з допомогою в діагностуванні, якщо цей діагноз було поставлено в наслідок цього запиту.

Висновки

Створений веб інтерфейс є достатньо зручним, та забезпечує безпроблемне виконання всіх поставлених перед системою задач. Сайт є добре захищеним, та розділяє можливості користувачів в залежності від їхнього рівню доступу. Локалізація сторінок дозволяє переглядати їх англійською та українською мовами, сервіс визначення хвороб також є локалізованим, і надає назви для діагнозів та симптомів в обох підтримуваних мовах. Реалізація коментарів під запитом на допомогу в діагностуванні дозволяє ефективне спілкування в режимі реального часу з низькою латентністю. Загалом, інтерфейс повністю справляється з своєю метою та забезпечує відносно швидкий доступ до всього функціоналу системи.

ВИСНОВОК

У результаті виконаної бакалаврської роботи розроблено програмну систему для діагностування хвороб за симптомами, обліку записаних діагнозів, та підтримки лікування для них.

Під час виконання бакалаврської роботи було досліджено теоретичні основи для побудови програмної цієї системи. Для дослідження теоретичних основ було взято існуючі програмні забезпечення та теоретичні відомості про ведення цифрового обліку пацієнтів лікарні, їхніх діагнозів, та способів лікування які їм приписані. За основу для машинного навчання системи визначення хвороб було взято дата сет з 40 тисячами записів про відповідність симптомів певним хворобам.

Сервіс для передбачення хвороб було зроблено на основі нейронної мережі, побудованої за допомогою бібліотеки Tensorflow, та розвернутій на платформі AWS за допомогою технології sageMaker. Інтерпритція запитів до моделі нейронної мережі відбувається через Lambda функції написані на мові Phyton. Також цей сервіс надає змогу отримувати список симптомів та хвороб, перекладати окрему назву діагноза або симптому мовами які підтримуються.

Веб сайт для доступу до сервісу було написана на мові програмування java, з використанням технологій та бібліотек: Spring Boot, Spring Core, Spring Web, Spring Data JPA, Spring Security, Spring Testing, Lombok, Mockito, log4j, javax servlet api. Як СУБД використовується MySQL. Для front-end частини сайту використовується html, css, javaScript та бібліотека jQuery. Сам сайт працює за допомогою серверу Tomcat та має відкритий доступ для всіх локальних користувачів.

Взагальному сервіс працює стабільно, точність передбачення хвороб не є дуже високою, але вона повинна збільшитись при введенні даної системи в експулатацію та тренування на більшій кількості даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Raoul-Gabriel Urma, Mario Fusco, Alan Mycroft «Modern Java in Action» 2018 592с
2. Robert C. M. «Clean Code: A Handbook of Agile Software Craftsmanship» 2008 464с
3. Kent Beck «Test Driven Development: By Example» 2002 240с
4. Herbert Schildt «Java: The Complete Reference 11th Edition» 2018 1077с
5. Документація фреймворку спрінг [Електронний ресурс] Режим доступу : https://spring.io/docs
6. Документація бібліотки jUnit [Електронний ресурс] Режим доступу :https://junit.org/junit5/docs/current/user-guide/
7. Документація MySQL [Електронний ресурс] Режим доступу : https://dev.mysql.com/doc/
8. Сайт бібліотеки Mockito [Електронний ресурс] Режим доступу : https://site.mockito.org/
9. Документація бібліотеки Log4j [Електронний ресурс] Режим доступу : https://logging.apache.org/log4j/2.x/log4j-api/apidocs/index.html
10. Документація технології JSTL [Електронний ресурс] Режим доступу : https://www.oracle.com/technetwork/java/index-141251.html
11. Документація бібліотеки jQuery [Електронний ресурс] Режим доступу : https://api.jquery.com/
12. Сайт технології Hibernate [Електронний ресурс] Режим доступу : https://hibernate.org/
13. Документація серверу Tomcat 9 [Електронний ресурс] Режим доступу : http://tomcat.apache.org/tomcat-9.0-doc/index.html
14. Сайт проекту Apache Maven [Електронний ресурс] Режим доступу: https://maven.apache.org/

15. Датасет з хворобами та симптомами, які при них виникають [Електронний ресурс] Режим доступу: https://www.kaggle.com/itachi9604/disease-symptom-description-dataset
16. Сайт бібліотеки TensorFlow [Електронний ресурс] Режим доступу: https://www.tensorflow.org/
17. Сайт платформи хмарних обчислень AWS [Електронний ресурс] Режим доступу: https://aws.amazon.com/
18. Сайт мови програмування Phyton [Електронний ресурс] Режим доступу: https://www.python.org/
19. Опис біблотеки jQuery [Електронний ресурс] Режим доступу : https://uk.wikipedia.org/wiki/JQuery
20. Опис технології JSP [Електронний ресурс] Режим доступу : https://uk.wikipedia.org/wiki/JSP
21. Опис мови програмування java [Електронний ресурс] Режим доступу : https://uk.wikipedia.org/wiki/Java
22. Опис технології JSP [Електронний ресурс] Режим доступу : https://uk.wikipedia.org/wiki/Hibernate
23. Сайт технології apimedic [Електронний ресурс] Режим доступу : https://apimedic.com/
24. Сайт технології symptomate [Електронний ресурс] Режим доступу : https://symptomate.com/uk/
25. Сайт технології webmd [Електронний ресурс] Режим доступу : https://symptoms.webmd.com/

ДОДАТОК А

```
package ua.training.hospital.controller;


import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;
import ua.training.hospital.controller.dto.*;
import ua.training.hospital.entity.Medicine;
import ua.training.hospital.entity.Procedure;
import ua.training.hospital.entity.Surgery;
import ua.training.hospital.service.diagnosis.DiagnosisService;
import ua.training.hospital.service.medicine.MedicineService;
import ua.training.hospital.service.procedure.ProcedureService;
import ua.training.hospital.service.surgery.SurgeryService;
```

```
import java.security.Principal;
import java.util.Optional;

@Controller
public class ShowDiagnosisController {
   private static final Logger logger = LogManager.getLogger(ShowDiagnosisController.class);

   @Autowired
   DiagnosisService diagnosisService;

   @Autowired
   MedicineService medicineService;

   @Autowired
   ProcedureService procedureService;

   @Autowired
   SurgeryService surgeryService;

   @RequestMapping(value = "patient{idPatient}/diagnosis{idDiagnosis}", method = RequestMethod.GET)
```

```
    public String getDoctorPage(@PathVariable long idPatient,
                                @PathVariable long idDiagnosis,
                                Model model) {
        logger.debug("requested patient" + idPatient + "/diagnosis" + idDiagnosis);
        diagnosisService.getDiagnosis(idDiagnosis).ifPresent(diagnosis -> {
            model.addAttribute("diagnosis",diagnosis);
        });
        logger.debug("returning showDiagnosis.jsp page");
        return "showDiagnosis";
    }


    @ResponseBody
    @RequestMapping(value = "/getMedicine{idDiagnosis}",produces = MediaType.APPLICATION_JSON_VALUE, method = RequestMethod.GET)
    public Page<Medicine> getMedicine(@PathVariable long idDiagnosis,
                                      @RequestParam(defaultValue = "0") int pageNumber,
                                      @RequestParam(defaultValue = "10") int recordsPerPage) {
        logger.debug("requested /getMedicine" + idDiagnosis);
        Page<Medicine> page = medicineService.findMedicineByDiagnosisId(pageNumber,recordsPerPage,idDiagnosis);
        logger.debug("returning org.springframework.data.domain.Page<Medicine> with size: " + page.getNumberOfElements());
```

```
    return page;
  }

  @ResponseBody
  @RequestMapping(value = "/getProcedures{idDiagnosis}",produces = MediaType.APPLICATION_JSON_VALUE, method = RequestMethod.GET)
  public Page<Procedure> getProcedures(@PathVariable long idDiagnosis,
                                       @RequestParam(defaultValue = "0") int pageNumber,
                                       @RequestParam(defaultValue = "10") int recordsPerPage) {
    logger.debug("requested /getProcedures" + idDiagnosis);
    Page<Procedure> page = procedureService.findProceduresByDiagnosisId(pageNumber,recordsPerPage,idDiagnosis);
    logger.debug("returning org.springframework.data.domain.Page<Procedure> with size: " + page.getNumberOfElements());
    return page;
  }

  @ResponseBody
  @RequestMapping(value = "/getSurgeries{idDiagnosis}",produces = MediaType.APPLICATION_JSON_VALUE, method = RequestMethod.GET)
  public Page<Surgery> getSurgeries(@PathVariable long idDiagnosis,
                                    @RequestParam(defaultValue = "0") int pageNumber,
```

```
                                @RequestParam(defaultValue = "10") int recordsPerPage) {
        logger.debug("requested /getSurgeries" + idDiagnosis);
        Page<Surgery> page = surgeryService.findSurgeriesByDiagnosisId(pageNumber,recordsPerPage,idDiagnosis);
        logger.debug("returning org.springframework.data.domain.Page<Surgery> with size: " + page.getNumberOfElements());
        return page;
    }

    @ResponseBody
    @RequestMapping(value = "/doctor/diagnosis{idDiagnosis}/addMedicine",
            method = RequestMethod.POST)
    public ResponseEntity<CreationResponse> addMedicine(
            @PathVariable long idDiagnosis,
            @Validated @RequestBody MedicineDTO medicineDto,
            BindingResult result,
            Principal principal){
        logger.debug("requested /doctor/diagnosis" + idDiagnosis + "/addMedicine post method");
        if(result.hasErrors()){
            logger.debug("medicineDTO contains errors returning \"wrongData\" creation response");
```

```
        return new ResponseEntity<>(new CreationResponse("wrongData",result.getAllErrors()), HttpStatus.BAD_REQUEST);
    }

    Optional<Medicine> created = medicineService.createMedicine(medicineDto,idDiagnosis,principal.getName());
    if(!created.isPresent()){
        logger.info("medicineService.createMedicine returned empty optional, returning \"cant create entity\" creation response");
        result.reject("{medicine.cannotCreate}");
        return new ResponseEntity<>(new CreationResponse("cant create entity",result.getAllErrors()), HttpStatus.BAD_REQUEST);
    }
    logger.debug("creation of medicine successful returning  \"created\" creation response\"");
    return new ResponseEntity<>(new CreationResponse("created",result.getAllErrors()), HttpStatus.OK);
  }

  @ResponseBody
  @RequestMapping(value = "/doctor/diagnosis{idDiagnosis}/addProcedure",
        method = RequestMethod.POST)
  public ResponseEntity<CreationResponse> addProcedure(
        @PathVariable long idDiagnosis,
```

```
    @Validated @RequestBody ProcedureDTO procedureDto,
    BindingResult result,
    Principal principal){
logger.debug("requested /doctor/diagnosis" + idDiagnosis + "/addProcedure post method");
if(result.hasErrors()){
    logger.debug("ProcedureDTO contains errors returning \"wrongData\" creation response");
    return new ResponseEntity<>(new CreationResponse("wrongData",result.getAllErrors()), HttpStatus.BAD_REQUEST);
}

Optional<Procedure> created = procedureService.createProcedure(procedureDto,idDiagnosis,principal.getName());
if(!created.isPresent()){
    logger.info("procedureService.createProcedure returned empty optional, returning \"cant create entity\" creation response");
    result.reject("{procedure.cannotCreate}");
    return new ResponseEntity<>(new CreationResponse("cant create entity",result.getAllErrors()), HttpStatus.BAD_REQUEST);
}
logger.debug("creation of procedure successful returning  \"created\" creation response\"");
```

```
    return new ResponseEntity<>(new CreationResponse("created",result.getAllErrors()), HttpStatus.OK);
  }

  @ResponseBody
  @RequestMapping(value = "/doctor/diagnosis{idDiagnosis}/addSurgery",
      method = RequestMethod.POST)
  public ResponseEntity<CreationResponse> addSurgery(
      @PathVariable long idDiagnosis,
      @Validated @RequestBody SurgeryDTO surgeryDto,
      BindingResult result,
      Principal principal){
    logger.debug("requested /doctor/diagnosis" + idDiagnosis + "/addSurgery post method");
    if(result.hasErrors()){
      logger.debug("SurgeryDTO contains errors returning \"wrongData\" creation response");
      return new ResponseEntity<>(new CreationResponse("wrongData",result.getAllErrors()), HttpStatus.BAD_REQUEST);
    }

    Optional<Surgery> created = surgeryService.createSurgery(surgeryDto,idDiagnosis,principal.getName());
    if(!created.isPresent()){
```

```
        logger.info("surgeryService.createSurgery returned empty optional, returning \"cant create entity\" creation response");
        result.reject("{surgery.cannotCreate}");
        return new ResponseEntity<>(new CreationResponse("cant create entity",result.getAllErrors()), HttpStatus.BAD_REQUEST);
    }
    logger.debug("creation of surgery successful returning  \"created\" creation response\"");
    return new ResponseEntity<>(new CreationResponse("created",result.getAllErrors()), HttpStatus.OK);
  }

  @ResponseBody
  @RequestMapping(value = "/doctor/diagnosis{idDiagnosis}/closeDiagnosis",
        method = RequestMethod.PATCH)
  public ResponseEntity<ClosingResponse> closeDiagnosis(
        @PathVariable long idDiagnosis,
        Model model){
    logger.debug("requested /doctor/diagnosis" + idDiagnosis + "/closeDiagnosis patch method");
    if(diagnosisService.closeDiagnosis(idDiagnosis)){
        logger.info("diagnosis with id: " + idDiagnosis + " closed");
        return new ResponseEntity<>(new ClosingResponse("closed"),HttpStatus.OK);
```

```
        }else{
            logger.info("cant close diagnosis with id: "+ idDiagnosis);
            return new ResponseEntity<>(new
ClosingResponse("cantClose"),HttpStatus.BAD_REQUEST);
        }
    }
}
```