Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Розробка клієнт-серверної системи діагностування пацієнтів

Текстова частина до курсової роботи
за спеціальністю «Інженерія програмного забезпечення» 121

Керівник курсової роботи
доц., к. н. Олецький О.В.
_____
(підпис)
“____” _____ 2020 р.

Виконав студент 3-го курсу
Зубко Д.А.
“____” _____ 2020 р.

Київ 2020

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем,

доцент, к.ф.-м.н.

_____ О.П. Жежерун

(підпис)

“____”_____ 2019 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Зубку Дмитру_

3-го курсу факультету інформатики

ТЕМА: Розробка клієнт-серверної системи діагностування пацієнтів

Вихідні дані:

- Клієнт-серверне застосування для роботи з медичними висновками у клініці

Зміст ТЧ до курсової роботи:

Вступ
1. Аналіз предметної області. Постановка завдання курсової роботи
2. Теоретичні відомості
3. Опис реалізації програмного продукту
Висновки
Список джерел
Додатки (за необхідністю)

Дата видачі “___” _____ 2019 р.

Керівник _____ Завдання отримано _____

Календарний план виконання курсової роботи

Тема: Розробка клієнт-серверної системи діагностування пацієнтів

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	Жовтень-листопад 2019р.	
2.	Вивчення та аналіз задачі	Листопад-грудень 2019р.	
3.	Розробка архітектури та загальної структури програми	Січень-лютий 2020р.	
4.	Створення застосунку	Лютий-березень 2020р.	
5.	Написання текстової частини	Березень-квітень 2020р.	
6.	Перегляд курсової роботи науковим керівником	Травень 2020р.	
7.	Створення презентації	Травень 2020р.	
8.	Захист курсової роботи		

Студент Зубко Д. А. _____

Керівник Олецький О. В. _____

“_____” _____ 2020 р.

Зміст

Перелік умовних позначень

DAO – абстрактний інтерфейс до певного типу бази даних.

JDBC – Java DataBase Connectivity (пакет, що входить в склад java.sql. Призначений для взаємодії Java-застосунків з різними системами управління базами даних)

MVC – Model View Controller (схема розділення даних застосунку)

SDK – Software Development Kit (набір інструментів для розробки)

JDK – Java Development Kit

API – Application Programming Interface (набір методів для взаємодії різних компонентів).

URI – уніфікований ідентифікатор ресурсу.

СКБД – система керування базами даних.

Анотація

Робота присвячена створенню клієнт-серверного застосування для полегшення та автоматизації роботи працівників у стоматологічній клініці.

У наведеній роботі розглянуто проблематику створення клієнт-серверних застосувань, сучасні методології створення подібних програм, поширені патерни проектування та розділення архітектурних рівнів при створенні подібних застосунків.

Клієнтський додаток створено на базі платформи JavaFX, серверна частина застосунку використовує Java та Spring Framework. Обраний набір технологій дозволив розгорнути сервер на хмарній PaaS платформі Heroku. В якості системи керування базою даних було обрано PostgreSQL.

Вступ

Розробка клієнт-серверних застосувань у наш час є досить звичною справою, проте правильне проектування бази даних, розділення архітектурних рівнів та вибір певних технологій може видатись не зовсім легкою задачею. Різні підходи до розробки напряму впливають на швидкодію та можливість розширення продукту в майбутньому.

Метою цієї курсової роботи є розробка промислового застосунку в медичній галузі з використанням сучасних технологій і підходів до проектування таких програмних продуктів. Створення такого роду додатку дозволяє спростити та автоматизувати роботу лікарів з медичними висновками та даними пацієнтів.

В якості основної мови програмування було обрано Java, з її широкими можливостями в області клієнт-серверних застосувань. Java в парі з Spring Framework являє собою чудовий інструмент для побудови серверного застосування та пропонує низку можливостей для роботи з СКБД, механізми для побудови систем автентифікації та авторизації користувачів, а також створення відповідного API для застосунків. Розглянуто також прикладні бібліотеки для роботи з клієнтською частиною додатку на кшталт JavaFX – для створення графічного інтерфейсу та Apache HTTP Components задля виконання HTTP-запитів користувачів до відповідного API.

1. Аналіз предметної області. Постановка завдання курсової роботи.

1.1. Аналіз сучасного стану клієнт-серверних застосунків.

Клієнт-серверна архітектура наразі є домінуючою концепцією створення та підтримки мережних застосунків. Перевагами такого підходу до побудови додатків є відсутність дублювання коду на стороні клієнтської та серверної програми. Чутливі дані, зазвичай, зберігаються на сервері, який як правило більш надійно захищений, а ніж клієнтський комп'ютер. Також, однією з найбільш вагомих переваг є обчислювальна потужність. Вимоги до продуктивності комп'ютера, на якому встановлена клієнтська частина зменшуються, так як всі необхідні розрахунки виконує сервер.

Дана архітектура дозволяє створити швидкий, надійний та захищений застосунок за умови дотримання певних концепцій: розділення архітектурних рівнів, аналіз функціональних можливостей додатку та вибір технологій, які зможуть забезпечити швидкодію в парі з іншими програмними сервісами. У іншому ж випадку – вищезгадані переваги клієнт-серверної архітектури можуть з легкістю перетворитись на недоліки, і як наслідок матимемо труднощі у підтримці такого застосунку у майбутньому.

1.2. Огляд існуючих аналогів розробки.

Різноманітні системи та комп'ютерні застосунки активно впроваджуються в різні галузі та сфери діяльності людини. Медична сфера не є виключенням в даному випадку, та налічує в собі велику кількість таких програмних рішень. Нижче буде наведено кілька таких застосувань.

Адента – багатофункціональний програмний застосунок, що автоматизує різні аспекти стоматологічної клініки [1]. Дане рішення складається з окремих модулів, які можуть бути встановлені окремо один

від одного та використовуватись як повноцінні додатки. Приклади модулів *Адента*:

- Журнал платежів – модуль для роботи з фінансовими операціями клієнтів та клініки.
- Історії хвороб – програма для заповнення історій хвороб, рішення для збереження та модифікації інформації щодо процесу лікування пацієнта
- Безпека – модуль для налаштування рівнів доступу окремих працівників, збереження даних в зашифрованому форматі.

Сильними сторонами цього застосунку є багатофункціональність, широкі можливості конфігурації програмних модулів та можливість використання файлів різних типів для використання в повсякденній роботі. З недоліків – труднощі з налаштуванням самого додатку та встановленням окремих модулів, орієнтованість лише на персональні комп'ютери з ОС Windows та використання локальної бази даних, що унеможливлює використання цих даних різними клієнтами і несе за собою загрозу безпеки персональних даних.

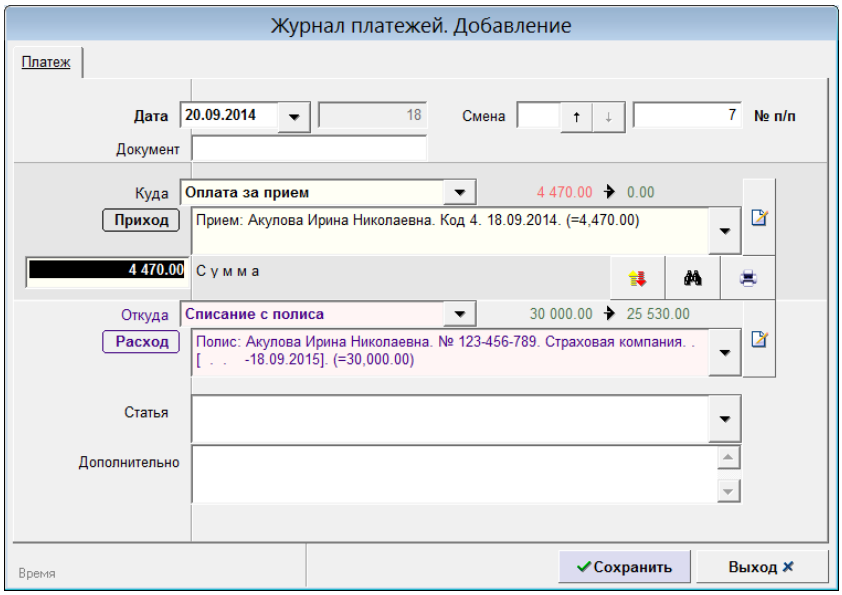

Рисунок 1 – приклад робочого вікна застосунку *Адента*

iClinic – хмарний сервіс для керування клінікою, створений у вигляді веб-застосунку [2]. Має ряд переваг перед іншими аналогами завдяки своїм функціональним можливостям: отримати доступ до різного роду даних можна через браузер, велика кількість компонентів дозволяє керувати різними аспектами клініки, а простота та ергономічність додатку робить процес освоєння інтерфейсу досить простим.

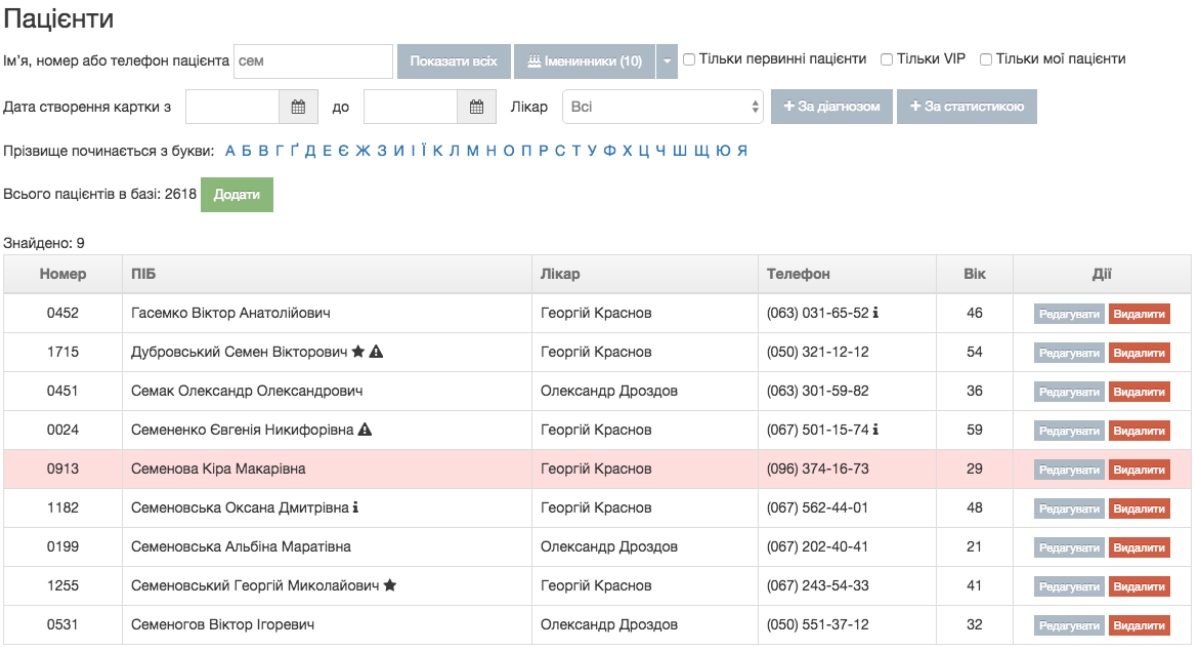

Рисунок 2 – приклад робочого вікна застосунку *iClinic*

1.3. Постановка завдання

Необхідно створити додаток, для роботи з даними пацієнтів та їхніх медичних висновків у стоматологічній клініці.

Ключові модулі системи:

1) Клієнтський застосунок для окремого користувача (в нашому випадку лікаря).

   Основні вимоги до клієнтського застосунку:

   1.1) Створення графічного інтерфейсу користувача.

1.2) Реалізація клієнтської частини клієнт-серверного застосунку з функціоналом, необхідним для взаємодії з сервером.

1.3) Перетворення та подання інформації про медичний висновок у вигляді документу Word.

2) Серверне застосування для організації роботи в межах підприємства (клініки).

Основні вимоги до серверного застосування:

2.1) Реалізація серверної частини клієнт-серверного застосунку з функціоналом, необхідним для взаємодії з клієнтами.

2.2) Взаємодія з СКБД.

2.3) Реалізація автентифікації та авторизації користувачів.

Функціональні можливості:

1) Реєстрація та авторизація користувача.
2) Створення та редагування інформації про окремих пацієнтів у системі.
3) Створення та редагування записів про хвороби.
4) Створення та редагування записів про історії хвороб окремих пацієнтів.
5) Пошук та фільтрування інформації про пацієнтів.
6) Автоматизоване створення історії хвороби в форматі Document Word на базі існуючих даних в графічному інтерфейсі чи базі даних.

2. Теоретичні відомості.

2.1. Spring Framework як засіб для створення веб застосунків

Spring Framework – універсальний додаток з відкритим кодом, який широко використовується в парі з Java для вирішення великої кількості задач [3].

Spring включає в себе велику кількість модулів, які можуть бути використані для вирішення тих чи інших проблем. Так, наприклад, Inversion of Control контейнер дозволяє конфігурувати компоненти застосунку та керувати життєвим циклом Java-об'єктів, Spring MVC використовують для створення всього, що пов'язано з мережею. Часто використовують також Spring Security та Spring Data модулі. Вони необхідні для створення авторизації і автентифікації користувача та взаємодії з поширеними ORM (Object-Relational Mapping) рішеннями і JDBC відповідно.

2.1.1. Spring Boot

Spring Boot – надбудова у Spring Framework. Він створений для автоматизації процедури побудови та конфігурації Spring-додатка [4]. Кожен раз, коли необхідно створити новий додаток на базі Spring Framework, виникає потреба імпортувати потрібні нам модулі, шукати web-контейнери (у випадку web-додатку), конфігурувати компоненти DAO (управління транзакціями, налаштування джерела даних та ін.). Труднощі також виникають при додаванні нових сторонніх бібліотек – необхідно шукати версії цих самих бібліотек сумісних з поточною версією Spring.

Рішенням цих проблем є вищезгаданий Spring Boot. Він включає в себе набір утиліт, що дозволяють розробнику створювати повноцінні web-застосунки затрачуючи мінімум зусиль на конфігурацію та написання коду.

Для пришвидшення керуванням залежностями Spring Boot неявно запаковує необхідні сторонні бібліотеки для різних типів додатків і подає їх

у вигляді «starter-пакетів». У разі необхідності, наприклад, створити web-застосунок на базі Spring, нам потрібно додати залежність “spring-boot-starter-web” використовуючи один з поширених інструментів автоматичної збірки програм – Maven, Ant чи Gradle [5]. Дана залежність запакує та надасть всі необхідні інструменти та бібліотеки для розробки Spring MVC додатків: “spring-webmvc”, “jackson-json”, “validation-api” і вбудований контейнер сервлетів який виконує функцію веб-сервера “Tomcat”.

2.1.2. Spring Security

Один з широковживаних модулів серед Spring-додатків є Spring Security. Це потужний фреймворк, що дозволяє керувати процесом авторизації та автентифікації, створювати рішення для захисту від різного виду атак [6].

Розглянемо принцип налаштування звичайної автентифікації та авторизації користувача на базі Spring Security. Перш за все додамо пакет “spring-boot-starter-security”, надалі, для конфігурації безпеки використовуватимемо свій клас, який потрібно наслідувати від WebSecurityCofigurerAdapter (див. рис. 3).

```
@Configuration
@EnableWebSecurity
public class MySecurity extends WebSecurityConfigurerAdapter {

   //code

}
```

Рисунок 3 – клас конфігурації безпеки

Анотація @Configuration вказує на те, що даний клас є конфігурацією Spring-додатка. Для налаштування доступу до різних ресурсів застосунку, необхідно перевизначити метод configure(HttpSecurity security).

```
@Configuration
@EnableWebSecurity
public class MySecurity extends WebSecurityConfigurerAdapter {

  @Autowired
  private MyUserDetailsService;

  @Autowired
  public void configureGlobal(AuthenticationManagerBuilder auth) {
    auth
        .userDetailsService(customUserDetailsService);
  }

   @Override
   protected void configure (HttpSecurity security) {
       security
              .antMatchers(“/sensitive/**”).hasRole(“ADMIN”)
              .antMathcers(“/usual/**”).hasRole(“USER”)
              .antMatchers(“/”, “/resources/**”).permitAll()
              .anyRequest().authenticated()
              .and()
              .httpBasic();
   }

}
```

Рисунок 4 – клас конфігурації безпеки (продовж.)

У наведеному коді (див. рис. 4) ми налаштовуємо доступ до певних ресурсів застосунку з використанням автентифікації та авторизації. Елемент httpBasic() визначає базову автентифікацію користувача, antMatchers(…) визначає шлях до ресурсу, і тип доступу: у випадку запиту за URI mycoolapp.com чи mycoolapp.com/resources/image - автентифікація не потрібна, у всіх інших випадках доведеться пройти цю процедуру. Також визначено певні ресурси, що потребують окремої авторизації (так званої ролі) [7]. Доступ до ресурсу mycoolapp.com/sensitive/password буде заборонено користувачу, що авторизувався в системі не як “ADMIN”. Важливу роль відіграє метод configureGlobal(AuthenticationManagerBuilder auth) що використовує при авторизації користувача його дані. Створивши свій клас, та реалізувавши інтерфейс UserDetailsService (див. рис. 5) ми можемо віднайти користувача в базі даних, присвоїти йому певні ролі та передати на опрацювання Spring Security.

```
@Service
public class MyUserDetailsService implements UserDetailsService {

  @Autowired
  private UserDAO userdao;

   @Override
   public UserDetails loadUserByUsername (String username) {
       return new User(userdao.findUser(username));
   }

}
```

Рисунок 5 – клас UserDetailsService

2.1.3. Розділення компонентів на архітектурні рівні в Spring

Проектування та розділення компонентів не є такою тривіальною задачею як може видатись з першого погляду. При створенні серверної частини певного застосунку зазвичай користуються певними патернами проектування та розділенням коду на певні рівні. В Spring розділяють такі типи компонентів: Service, Repository, Controller [8]. Розглянемо приклад, що демонструє використання цих компонентів та зв'язок між ними.

Рівень доступу до даних (Repository layer) призначений для зберігання, отримання і пошуку. Як правило, даний рівень використовується для роботи з базами даних. Уявімо, що ми розробляємо сервіс для отримання інформації про номер телефону. Рівень доступу до даних виглядатиме таким чином:

```
@Repository
public class PhoneRepositoryImpl implements IPhoneRepository {

  @Autowired
  private NamedParameterJDBCTemplate template;

   @Override
   public String findNumberByID (int id) {
       SQLParameterSource param = new MapSQLParameterSource()
              .addValue("id", id);
       return this.template.queryForObject(SQL, param, String.class);
   }
```

Рисунок 6 – приклад класу рівня доступу до даних

У наведеному коді створено клас, що реалізовує інтерфейс із заздалегідь визначеним методом findNumberByID(int id).

Основною ідіомою Spring є прагнення до написання слабко зв’язаного коду. Основою Spring Core є ApplicationContext який зберігає в собі інформацію про так звані «біни» (beans). Кожен раз, коли ми анотуємо клас позначкою @Service, @Repository, @RestController – Spring створює об'єкт цього класу за замовчуванням в одиничному екземплярі (singleton) який і є цим самим «біном». Отримати доступ до цього об’єкту можна за допомогою анотації. “Autowired”, що ми і зробили аби використати NamedParameterJDBCTemplate для доступу до бази даних.

Перейдемо до службового рівню (Service layer). Даний рівень являє собою фасад для деякої бізнес-логіки застосунку. Розширення застосунку з номерами телефонів буде мати такий вигляд:

```
@Service
public class PhoneServiceImpl implements IPhoneService {

   @Autowired
   private IPhoneRepository phoneDAO;

   @Override
   public String findNumberByID (int id) {
       String number = phoneDAO.findNumberByID(id);
       if (number.charAt(1) == ‘3’ && number.charAt(2) == ‘8’)
              return “Ukraine ” + number;
       return “Unknown country ” + number;
```

Рисунок 7 – приклад класу рівня предметної області

У наведеному фрагменті, ми замінили анотацію на @Service, створили новий бін рівню предметної області, використали існуючий об’єкт репозиторію та додали певної логіки застосунку.

Відокремлення компонентів є важливою складовою будь-якого застосування. Таким чином бізнес-логіка застосунку ізольована від

компонента, що працює з базою даних, а це в свою чергу спрощує розширення та підтримку застосунку в майбутньому.

Підхід Spring Framework при створенні RESTful web-сервісу полягає в обробці HTTP-запитів контролерами. Такі компоненти розпізнаються за допомогою анотації @RestController.

```
@RestController
@RequestMapping(“/api/v1”)
public class PhoneController {

   @Autowired
   private IPhoneService phoneService;

   @GetMapping(“/phone/{id}”)
   public ResponseEntity<String> findNumber (
       @PathVariable(“id”) final int id) {
       return new ResponseEntity<>(phoneService.findNumberByID(id), OK);
   }
}
```

Рисунок 8 – приклад класу REST-контролера

Для обробки GET-запиту, створимо контролер, що включає в себе сервіс, який допоможе обробити інформацію щодо запиту клієнта (див. рис. 8). Анотація @GetMapping гарантує, що HTTP-запити на /phone/{id} будуть співставлені findNumber(int id) методу. @PathVariable зв’язує значення змінної id (в URI) з параметром id. Анотація @RequestMapping вказує на те, що даний контролер також слухатиме запити що починаються з /api/v1. Таким чином, для отримання певної інформації щодо номеру телефону, клієнту необхідно створити запит вигляду: domain/api/v1/phone/{id}. Spring MVC підтримує також інші типи запитів, що можуть бути зконфігуровані відповідними анотаціями @PostMapping, @DeleteMapping, @PutMapping та ін.

2.2. JavaFX. Платформа для створення додатків з графічним інтерфейсом користувача.

JavaFX – це платформа, що являє собою частину Java 7. Починаючи з JDK 11, більше не входить до складу Java SE та існує як окремий модуль компанії Gluon [9]. Даний модуль дозволяє з легкістю створювати візуальні програми для різних платформ – Windows, macOS, Linux та Solaris. Існує також мобільний емулятор, що входить до складу SDK та дозволяє створювати мобільні застосунки на базі JavaFX.

2.2.1. FXML та створення користувацького інтерфейсу

Всі графічні елементи застосунку можуть бути описані прямо в Java-коді, проте це не є гарним прикладом розділення архітектурних рівнів. На допомогу приходить FXML – мова розмітки, що створена Oracle Corporation для проектування графічного інтерфейсу JavaFX-додатків. FXML базується на основі XML-розмітки, що в свою чергу дозволяє створювати інтуїтивно зрозумілі та ієрархічно структуровані елементи [10].

Кожен компонент може бути описаний за допомогою FXML-розмітки за умови, що цей компонент імпортований з відповідного пакету.

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>

<VBox>
    <Label text="Sample text"/>
    <Label text="Another text"/>
    <Button text="Swap text"/>
</VBox>
```

Рисунок 9 – типове наповнення FXML-файлу

У наведеному прикладі демонструється створення таких графічних компонентів як стрічки та кнопка, що мають певний текст та розміщені вертикально по відношенню один до одного використовуючи

компонування елементом VBox. Стилізація окремих елементів може здійснюватися як за допомогою підключення зовнішнього css-файлу до кореневого елементу, так і з використанням вбудованих атрибутів.

2.2.2. Обробка поведінки та взаємодії з користувачем

Елементи, створені на базі FXML, є статичними. Для взаємодії користувача з певними компонентами, необхідно створити певний прошарок, що дозволить спроектувати певну бізнес-логіку застосунку по відношенню до цих компонентів. Кожному файлу FXML-розмітки може бути зіставлений контролер, що являє собою звичайний Java клас.

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>

<VBox fx:controller=”SampleController”>
    <Label fx:id=”firstLabel” text="Sample text"/>
    <Label fx:id=”secondLabel” text="Another text"/>
    <Button text="Swap text" onMouseClicked="#swapValues"/>
</VBox>
```

Рисунок 10 – елементи, що містять специфічні атрибути

Дана FXML-розмітка зв’язана з контролером з назвою SampleController за допомогою атрибуту fx:controller. Цей атрибут має бути розміщений в кореневому елементі. Для доступу до окремих компонентів в контролері або в інших FXML-файлах існує можливість надати унікальний ідентифікатор цьому самому елементу. Цей ідентифікатор можна використовувати як посилання (fx:id). До певного елементу можна також додати обробника подій (див. рис. 11), для прикладу: обробник, що реагує на те, чи клацнули на елемент мишкою (onMouseClicked).

```
public class SampleController {

	public Label firstLabel;
	public Label secondLabel;

	public void swapValues(){
		String firstLabelText = firstLabel.getText();
		firstLabel.setText(secondLabel.getText());
		secondLabel.setText(firstLabelText);
	}
}
```

Рисунок 11 – клас-контролер певного FXML-файлу

JavaFX автоматично намагається співставити елементи з таким же ж fx:id як і поля в класі-контролері. За таким принципом працює і співставлення обробника подій з методами. Варто зазначити, що якщо поле потребує більш строгий модифікатор доступу, то воно повинно бути анотоване позначкою @FXML.

3. Опис реалізації програмного продукту

3.1. Обґрунтування вибору засобів розробки

Клієнтська частина застосунку реалізована на базі Java, з використанням бібліотеки для створення графічного інтерфейсу користувача JavaFX.

Так як програма являє собою десктопний застосунок, мова програмування Java стала досить гарним рішенням, адже однією з головних переваг JavaFX є багатоплатформовість. Для створення готового для використання програмного продукту, достатньо збудувати проект, та надати користувачу виконуваний JAR-файл. З необхідного програмного забезпечення, що має бути встановлено на комп'ютері користувача є JRE (Java Runtime Environment). У іншому випадку, можливо забезпечити користувачів певної операційної системи готовими інсталяторами, без необхідності у встановленні JRE [11].

Для надсилання запитів до сервера було використано бібліотеку Apache HTTP Components. Вона надає гнучкий та зручний інструментарій для роботи з HTTP-запитами. Дане рішення дозволяє створювати запити різного типу (GET, POST), керувати автентифікацією користувача та відповіддю від сервера, і що важливо, модифікувати запит для тих чи інших потреб.

Сервер для додатку створений за допомогою Java та Spring Framework. Разом, вони надають можливість проектувати масштабні застосунки підприємницького рівня, які досить легко підтримувати та розширювати в будь-який момент часу.

Для розгортки серверної частини застосунку обрана хмарна PaaS-платформа Heroku [12]. Дана платформа є досить поширеною завдяки підтримці низки мов програмування таких як: Ruby, Python, Java, PHP, Go,

Scala, Clojure. Також, Heroku надає зручні інструменти для роботи базами даних, як приклад – доповнення Heroku Postgres.

В якості системи керування базами даних використовується PostgreSQL. Це некомерційна, об'єктно-реляційна СКБД, що входить в п'ятірку найпопулярніших та найпоширеніших СКБД [13]. PostgreSQL завоювала довіру багатьох розробників завдяки своїй надійності, безпеці та швидкодії.

IntelliJ IDEA – інтегроване середовище розробки, що створено компанією JetBrains [14]. Підтримує велику кількість мов програмування та має різноманітні інструменти для роботи з базами даних, системами контролю версій і рішення для рефакторингу коду.

3.2. Опис розробки програми

Java – об'єктно-орієнтована мова програмування, тому розробка проводилась з ухилом на основні принципи ООП.

Клієнтська частина реалізована за поширеним патерном MVC. У ролі моделей виступають класи, що являють собою шаблони для створення тих чи інших об'єктів. View представляє собою FXML-файл, що містить у собі всі елементи, які відображаються користувачу в графічному інтерфейсі. Контролерами являються класи, що попарно співставлені та зв'язані з FXML-файлами. Кожен з контролерів виконує роль обробника команд користувача та містить певну бізнес-логіку.

Серверна частина складається з наступних рівнів: рівень мережі (Web Layer), службовий рівень (Service Layer), рівень доступу до даних (Repository Layer) та доменна модель (Domain Model). Рівень мережі включає в себе REST-контролери, що слухають запити клієнтів і контролери, які працюють над обробкою виключень. Службовий рівень представляють класи, які займаються формуванням певної бізнес-логіки застосунку. Repository Layer – це всі класи та інтерфейси, що формують

з’єднання з СКБД та відповідають за отримання і пошук інформації з бази даних. Доменна модель складається з класів, що формують сутності, та класів які зберігають певні об’єкти-значення для повноцінної роботи додатку.

3.3. Створення об’єктів та розробка головної програми

Як було сказано раніше, клієнтська частина додатку створена на основі JavaFX. Це означає, що основними компонентами клієнтського застосунку являються FXML-файли, класи-контролери та моделі.

При запуску програми, відправною точкою є клас Launcher, що містить метод main(String args[]). Саме тут задаються основні параметри роботи застосунку: створення вікна, надання йому певних властивостей (ширина, висота, тип відображення). Ключову роль також відіграє клас ScreenManager. Він містить в собі метод, що завантажує FXML-файли та присвоює їм відповідні класи-контролери (див. рис. 12).

```
public class ScreenManager {

	public static PatientInfoController patientInfoController;
	public static LoginController loginController;
	// ...
	private static ScreenMap<String, Pane> screenMap;
	public static Scene main;

	private static void addScreen(String name, Pane pane){
		screenMap.put(name, pane);
	}

	public static void activate(String screenName){
		main.setRoot(screenMap.get(name));

	}

	public static void updateScreens(){
		FXMLLoader loader = new FXMLLoader();
		addScreen(“LoginView”, loader.load(“path/to/fxml”));
		loginController = loader.getController();
		// ...
	}
}
```

Рисунок 12 – клас ScreenManager

Даний клас також містить в собі статичний метод activate (String screenName), що дозволяє з легкістю переміщатись між сценами відповідно до FXML-файлів розмітки.

Кожному FXML-файлу поставлений у відповідність контролер, що включає в себе елементи з розмітки та керує їх поведінкою (див. рис. 13). У контролері, можлива також реалізація логіки застосунку та створення методів для валідації дій користувача.

```
public class AnamnesisController {

    public TextArea complaintsTextArea;
    public TextArea aMorbiTextArea;
    public TextArea aVitaeTextArea;

    public void onBackButtonClick() {
        ScreenManager.activate("PatientInfoView");
    }

    public void onNextButtonClick() {
        if (validFields()) {
            ScreenManager.activate("DataFillingView");
        }
    }

    private boolean validFields() {
        if (complaintsTextArea.getText().equals("")){
            AlertManager.showAlert("Скарги не можуть бути пустими!");
            return false;
        }
        //...
        return true;
    }
}
```

Рисунок 13 – клас-контролер

Пакет model.entity містить в собі класи, які використовуються для створення різних об’єктів, які в свою чергу беруть участь у формуванні тіла запиту та в інших елементах графічного інтерфейсу.

Для взаємодії з сервером та можливості відправки HTTP-запитів, реалізований клас ClientUtil. В ньому реалізовані методи для формування та відправки запитів за допомогою Apache Components [15].

Серверна частина містить в собі низку пакетів, що включають в себе різні класи (див. рис. 14)

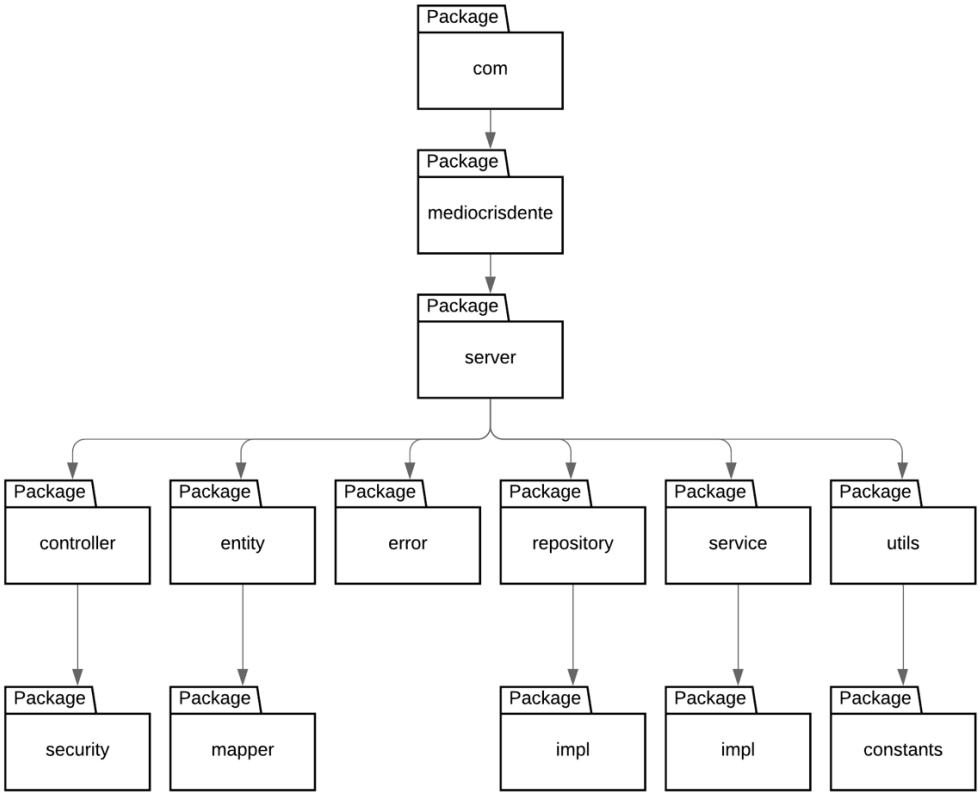

Рисунок 14 – діаграма структури пакетів сервера

Коротка характеристика основних пакетів:

- Пакет controller – містить усі класи, які виконують функцію REST-контролера.
- Пакет entity – включає класи-шаблони для створення різного роду об’єктів.
- Пакет error – набір класів користувацьких помилок.

- Пакет repository – інтерфейси і їх реалізації, що використовуються для доступу до даних.
- Пакет service – інтерфейси і їх реалізації, що використовуються для створення деякої логіки застосунку.
- Пакет utils – містить у собі enumerations, та утиліти для роботи з різними даними.

Кожен з елементів Service, Controller та Repository створений на базі Spring @Component.

Контролери, що слухають на запити користувачів містять в собі методи та їхні мапінги з вказаними типами самих запитів (див. додаток «А»). Майже кожній сутності з пакету entitiy встановлений у відповідність контролер, що опрацьовує запити пов'язані з цим об'єктом. Так наприклад клас-контролер DoctorController керує запитами клієнтів, що пов'язані з лікарями (користувачами системи).

Елементи пакету service містять в собі інтерфейси та їх реалізації для подальшого зручного розширення та підтримки. Реалізований також клас CustomDoctorDetailsService, що використовується Spring Security для авторизації користувача, та встановлення наданих йому компетенцій.

Класи DAO використовуються для формування запитів, та отримання необхідної інформації з бази даних. Ці класи містять в собі посилання на зовнішні файли в яких зберігаються самі SQL-запити (див. додаток «Б»).

3.4. Тестування програми і результати її виконання

Робота з клієнтським застосунком розпочинається зі входу в систему – користувач може увійти в головне меню використовуючи свої реєстраційні дані, або ж перейти безпосередньо до самої реєстрації.

Рисунок 15 – вікно входу в систему

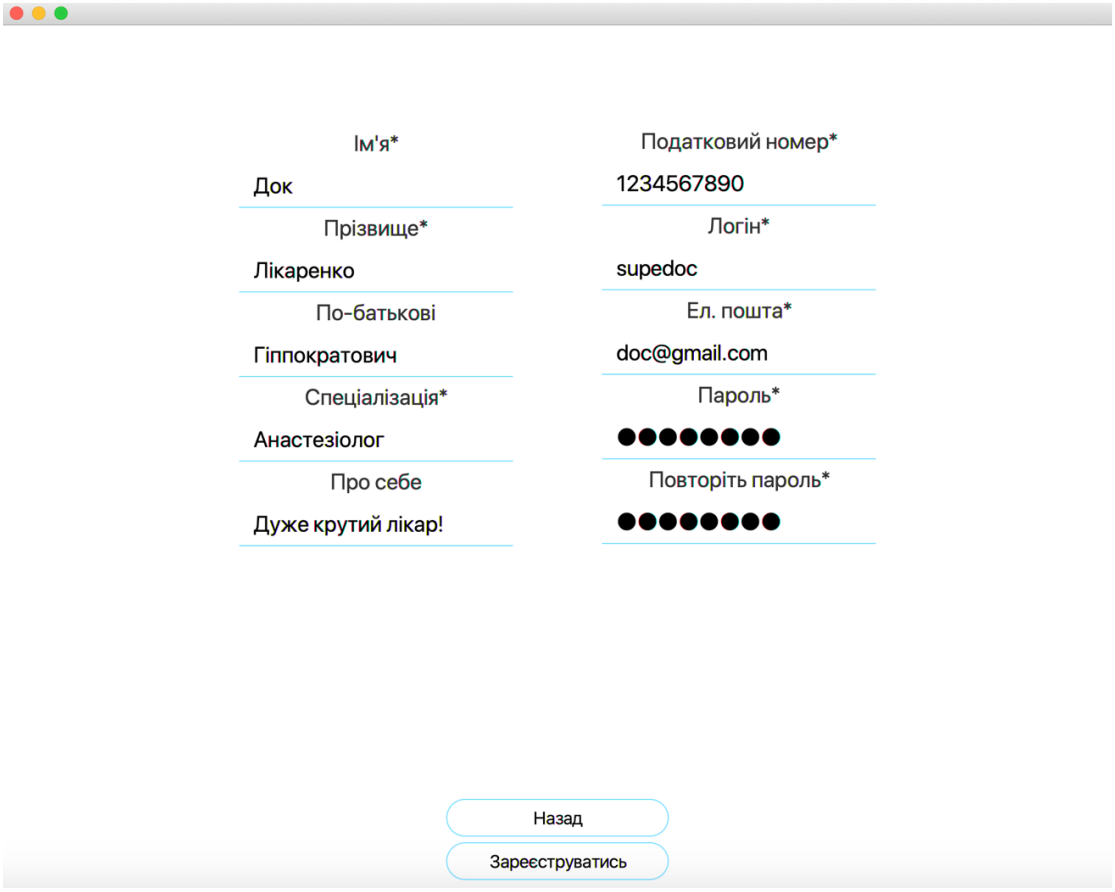

Рисунок 16 – вікно реєстрації

Система повідомляє користувача про різного роду помилки, що виникають під час роботи з застосунком. Для прикладу – повідомлення відображуються за умови невірно введених даних авторизації чи реєстрації (див. рис. 17).

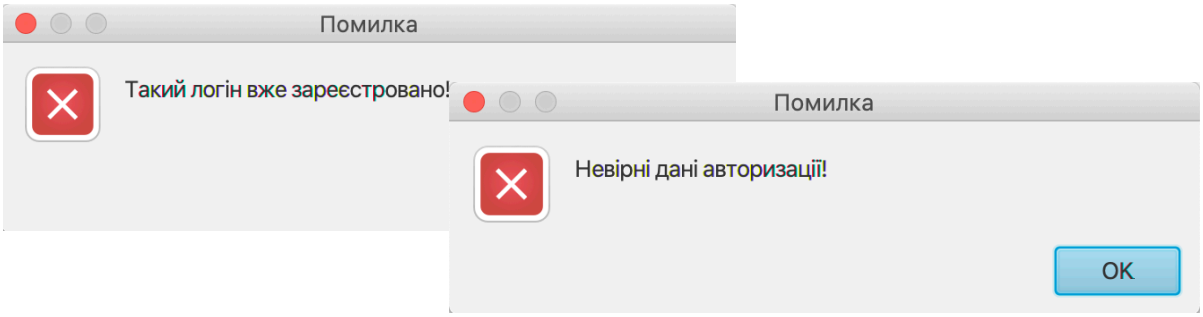

Рисунок 17 – приклади інформаційних вікон

Після успішного входу в систему, користувач бачить перед собою головне меню програми. В меню він може перейти до створення нової історії хвороби, переглянути вже створені ним історії в меню «Історії хвороб», або ж за наявності відповідних прав перейти в конфігурацію бази даних хвороб.

Рисунок 18 – головне меню програми

При створенні нової історії хвороби, користувач вносить персональні дані нового пацієнта, або ж шукає вже існуючі дані щодо нього за умови присутності запису клієнта в базі даних (див. рис. 19). Огляд пацієнта відбувається з заповненням усіх необхідних полів. Обов’язкові поля містять валідації, що унеможливлять перехід до наступних вікон, допоки дані не будуть надані. До обов’язкових даних можна віднести прізвище і ім’я пацієнта, дату народження, стать та контактні дані.

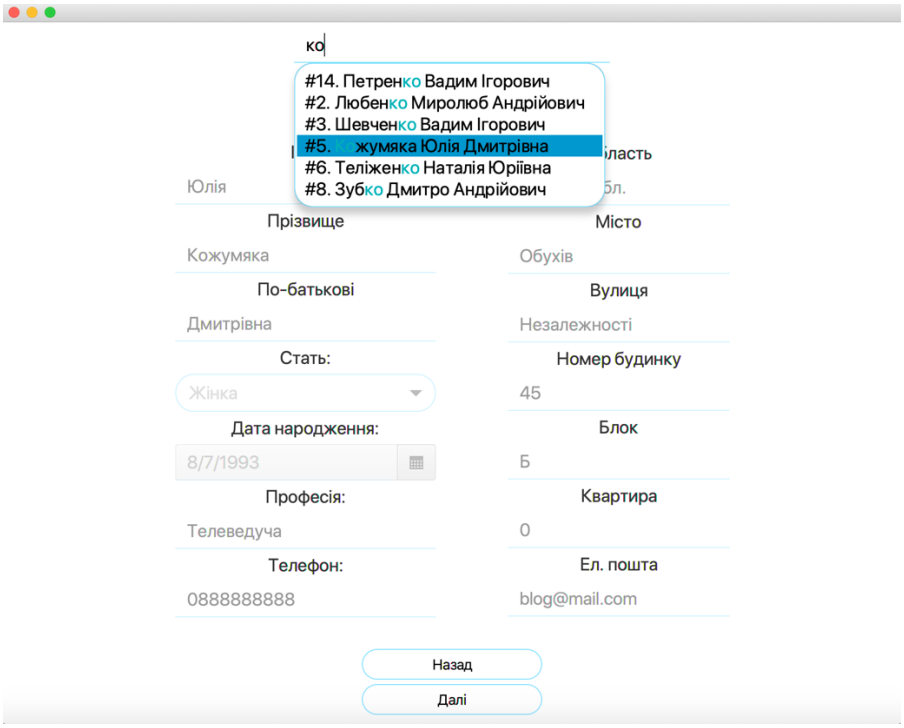

Рисунок 19 – пошук та внесення персональних даних відвідувача

Користувач заповнює всі дані обстеження, обирає супутній діагноз, та назначає тип лікування, так як окрема хвороба може мати декілька типів лікування.

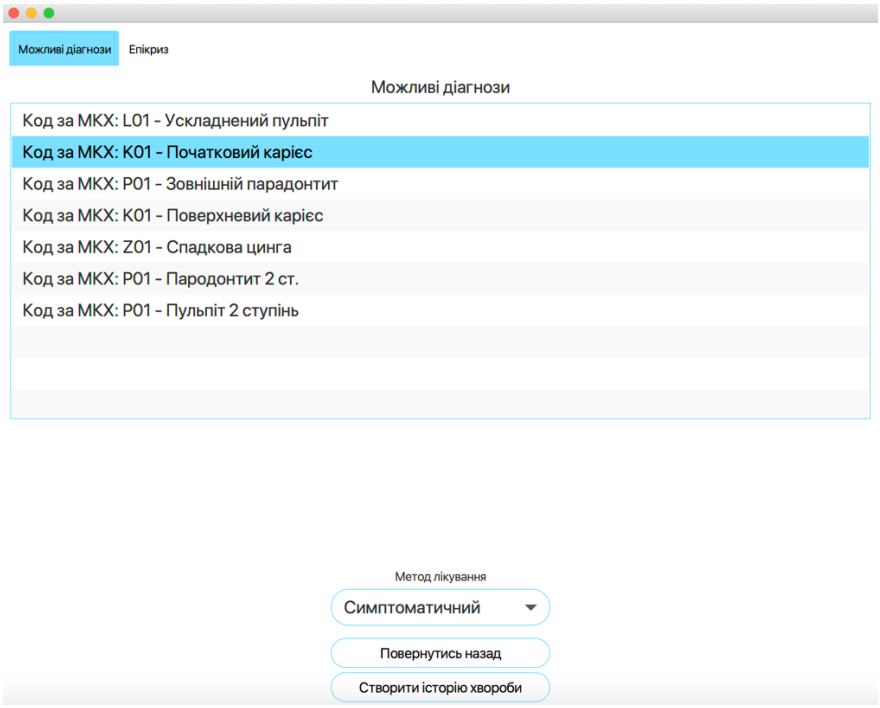

Рисунок 20 – вибір діагнозу та методу лікування

Після остаточної перевірки даних, при натисканні кнопки «Створити історію хвороби», користувачу відображається діалогове вікно. Це вікно дозволяє зберегти документ у форматі Word (див. додаток «В»). Сама історія хвороби також зберігається у базі даних.

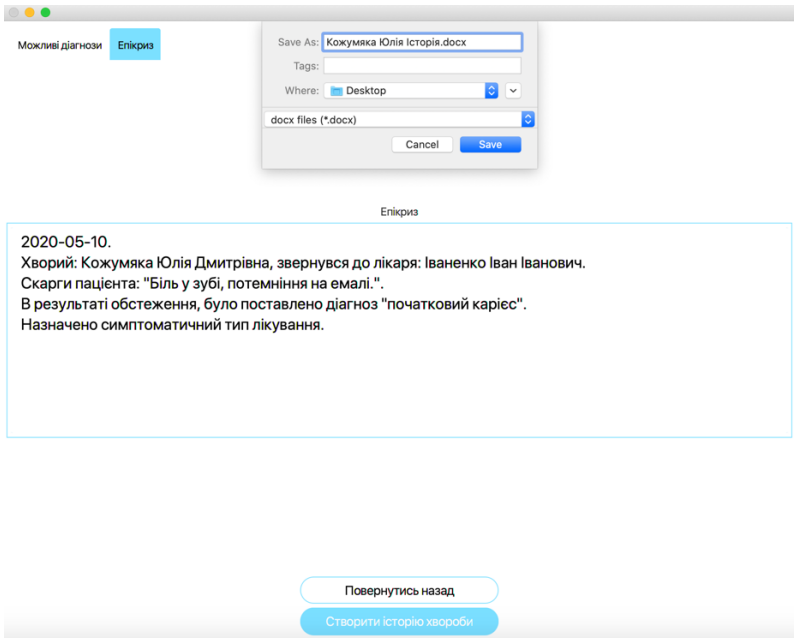

Рисунок 21 – остаточна перевірка та збереження історії хвороби

Для редагування чи доповнення історії хвороби, користувач може скористатись вищезгаданим меню «Історії хвороб». Тут відображаються форми, що були створені раніше поточним користувачем.

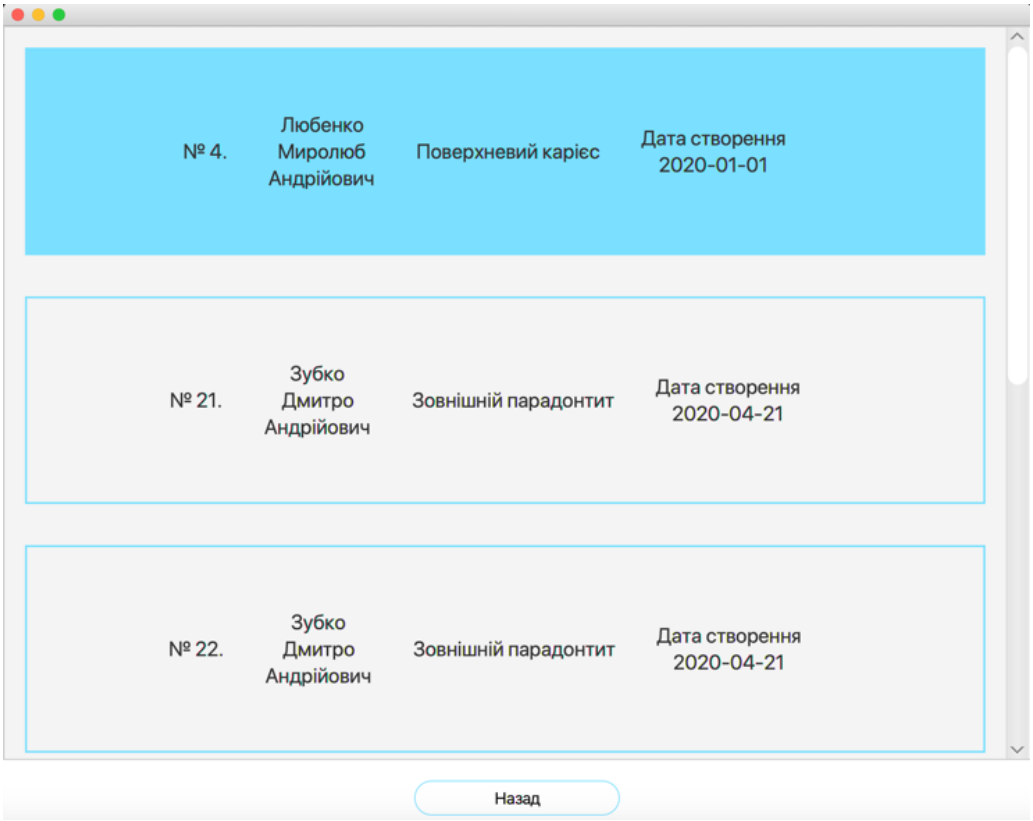

Рисунок 22 – список створених історій

Користувач може отримати доступ до редагування інформації щодо хвороб. Для цього використовується система ролей, тому за умови наявності відповідної компетенції у лікаря, меню «База хвороб» дозволяє проводити різноманітні операції з записами у ній (див. рис. 23). Якщо ж авторизований користувач не має прав доступу, то відповідне інформаційне повідомлення буде представлено в графічному інтерфейсі.

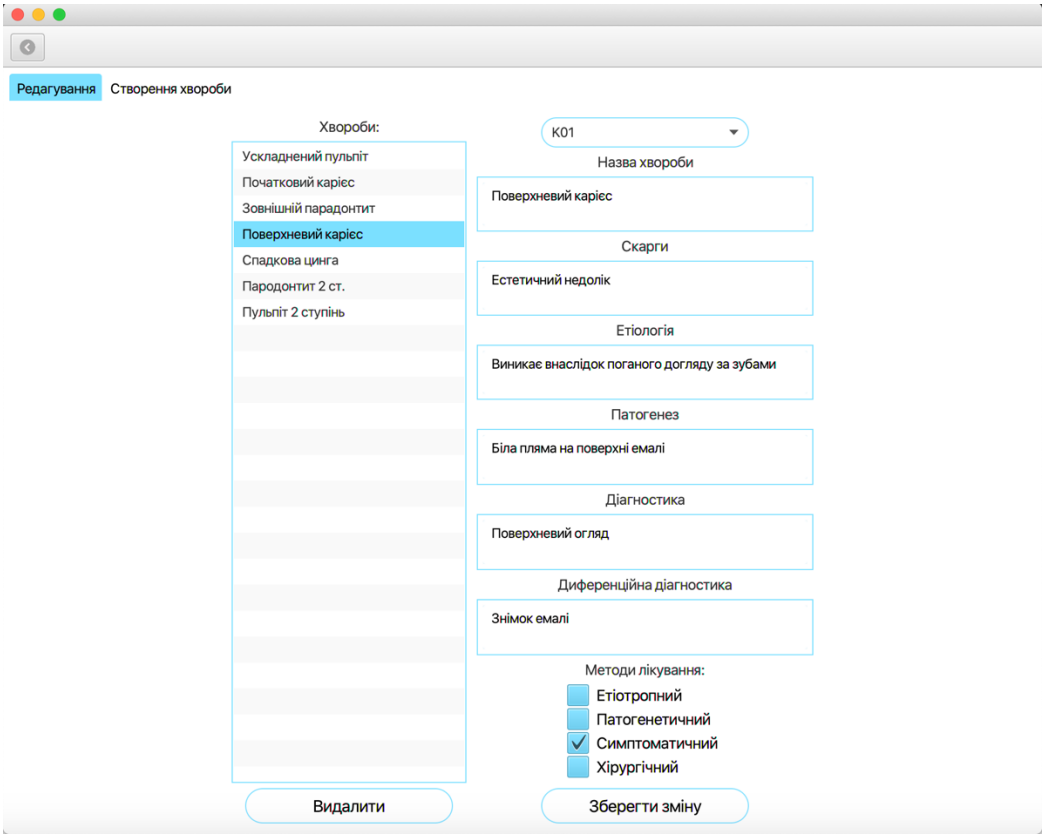

Рисунок 23 – вікно редагування хвороби

Висновок

В рамках дослідження клієнт-серверних систем, було розроблено застосунок, функціональні можливості якого відповідають вимогам, що описані у постановці завдання. Розглянуто основні принципи побудови даного роду додатків, архітектурні рішення та основні патерни проектування клієнт-серверних застосунків. Також у даній роботі описано можливості сучасних фреймворків для роботи як з серверною частиною так і клієнтською.

Створений додаток можна повноцінно використовувати у медичній сфері. Окрім цього, гнучкість використаних патернів та підходів дозволяє розширити його можливості за потреби. Прикладом розширення може слугувати, наприклад, автоматичне надання діагнозів, що відповідають отриманій інформації від користувача та створення веб-версії клієнтської частини для доступу до функціоналу з веб браузера клієнта.

Список використаної літератури

1. Головний сайт Адента [Електронний ресурс]: https://adenta.pro/
2. Головний сайт iClinic [Електронний ресурс]: https://iclinic.ua/
3. Spring Framework [Електронний ресурс]: https://spring.io
4. Spring Boot [Електронний ресурс]: https://spring.io/projects/spring-boot
5. Using Spring Boot with build tools [Електронний ресурс]: https://docs.spring.io/spring-boot/docs/current/reference/html/using-spring-boot.html
6. Spring Security [Електронний ресурс]: https://spring.io/projects/spring-security
7. Fraser, B. (1997), "Site Security Handbook", 29 p.
8. "Component" annotation [Електронний ресурс]: https://www.journaldev.com/21429/spring-component
9. Gluon and JavaFX [Електронний ресурс]: https://gluonhq.com/gluon-and-javafx/
10. FXML Documentation [Електронний ресурс]: https://docs.oracle.com/javase/8/javafx/api/javafx/fxml/doc-files/introduction_to_fxml.html
11. Packaging JavaFX applications [Електронний ресурс]: https://docs.oracle.com/javafx/2/deployment/self-contained-packaging.htm
12. Головний сайт Heroku [Електронний ресурс]: https://www.heroku.com
13. Рейтинг СКБД [Електронний ресурс]: https://db-engines.com/en/ranking
14. Головний сайт IntelliJ IDEA [Електронний ресурс]: https://www.jetbrains.com/idea/
15. Apache HTTP Components [Електронний ресурс]: https://hc.apache.org

ДОДАТОК А. Серверний код – клас-контролер DiseaseTypeController

```
package com.mediocrisdente.server.controller;

import com.mediocrisdente.server.entity.DiseaseType;
import com.mediocrisdente.server.service.impl.DiseaseTypeServiceImpl;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/v1")
public class DiseaseTypeController {

  private DiseaseTypeServiceImpl diseaseTypeService;

  @Autowired
  public DiseaseTypeController(final DiseaseTypeServiceImpl diseaseTypeService){
    this.diseaseTypeService = diseaseTypeService;
  }

  @GetMapping("/diseasetype/count")
  public ResponseEntity<Long> getDiseaseTypeCount() {
    return new ResponseEntity<>(diseaseTypeService.count(), HttpStatus.OK);
  }

  @PostMapping(value = "/diseasetype")
  public ResponseEntity createDiseaseType(
    @RequestBody final DiseaseType diseaseType) {
    diseaseTypeService.insert(diseaseType);
    return new ResponseEntity(HttpStatus.CREATED);
  }

  @PutMapping("/diseasetype")
  public ResponseEntity updateDiseaseType(
    @RequestBody final DiseaseType diseaseType) {
    diseaseTypeService.update(diseaseType);
    return new ResponseEntity<>(HttpStatus.OK);
  }

  @DeleteMapping("/diseasetype/{icd_code}")
  public ResponseEntity deleteDiseaseType(
    @PathVariable("icd_code") final String diseaseTypeIcdCode) {
    diseaseTypeService.delete(diseaseTypeIcdCode);
    return new ResponseEntity<>(HttpStatus.OK);
  }
```

```
  @GetMapping("/diseasetype/{icd_code}")
  public ResponseEntity<DiseaseType> getDiseaseType(
    @PathVariable("icd_code") final String diseaseTypeIcdCode) {
    return new ResponseEntity<>(diseaseTypeService.find(diseaseTypeIcdCode),
HttpStatus.OK);
  }

  @GetMapping("/diseasetype")
  public ResponseEntity<List<DiseaseType>> getDiseasesTypes() {
    return new ResponseEntity<>(diseaseTypeService.findAll(), HttpStatus.OK);
  }
}
```

ДОДАТОК Б. Зовнішній файл з SQL-запитами

```
find_number_of_diseases_types   = SELECT count(1) \
                        FROM diseases_types

insert_disease_type           = INSERT INTO diseases_types \
                        (icd_code, name, distinctness) \
                        VALUES (:icd_code, :name, :distinctness)

update_disease_type           = UPDATE diseases_types \
                        SET icd_code = :icd_code, name = :name, distinctness = :distinctness \
                        WHERE icd_code = :icd_code

delete_disease_type           = DELETE FROM diseases_types \
                        WHERE icd_code = :icd_code

find_disease_type             = SELECT icd_code, name, distinctness \
                        FROM diseases_types \
                        WHERE icd_code = :icd_code

find_all_diseases_types       = SELECT icd_code, name, distinctness \
                        FROM diseases_types
```

ДОДАТОК В. Сформований Word-файл з історією хвороби

Історія хвороби

ПІБ: Кожумяка Юлія Дмитрівна
Дата народження: 1993-08-07
Стать: Жінка
Адреса: м. Обухів, вул. Незалежності 45
Професія: Телеведуча
Скарги:
Біль у зубі, потемніння на емалі.
Anamnesis morbi:
Відсутній.
Anamnesis vitae:
Веде здоровий спосіб життя, не має шкідливих звичок.

Об'єктивні дані

Основний метод дослідження:
Зондування безболісне, перкусія безболісна.
Об'єктивний метод дослідження:
Об'єктивно: бура пляма на поверхні емалі.
Додатковий метод дослідження:
Відсутній.

Дані хвороби

Основний діагноз:
Початковий карієс
Тип лікування:
Симптоматичний

Епікриз:

2020-05-10. Хворий: Кожумяка Юлія Дмитрівна, звернувся до лікаря: Іваненко Іван Іванович. Скарги пацієнта: "Біль у зубі, потемніння на емалі.". В результаті обстеження, було поставлено діагноз "початковий карієс". Назначено симптоматичний тип лікування.