

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ КОМП'ЮТЕРНИХ
СИСТЕМ
КАФЕДРА «ІНФОРМАЦІЙНИХ СИСТЕМИ»

КУРСОВА РОБОТА

з дисципліни «**ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ**»
на тему : «Система управління медичними прийомами»

Виконав: Студент (ка) _2_ курсу
спеціальності 122 «Комп'ютерні науки»
Замковий Кирило Вячеславович

Керівник:

М.А. Годовиченко

(підпис) (прізвище та ініціали)

(підпис) (прізвище та ініціали)

Одеса 2025

Зміст

Вступ.....	3
1. Аналіз предметної області	4
Постановка задачі	4
Основні бізнес-процеси	4
2. Проєктування програмного забезпечення	5
1. Проєктування структури даних	5
2. Опис архітектури застосунку	6
3. Опис REST API.....	6
4. Реалізація програмного продукту	9
1. Моделі (Entity-класи, DTO)	9
2. Репозиторії (інтерфейси доступу до даних)	15
3. Сервіси (бізнес-логіка)	19
4. Контролери (обробка HTTP-запитів)	23
5. Конфігурація	30
5.Тестування та налагодження	34
Висновки.....	38

Вступ

Метою курсового проекту є створення програмного забезпечення для управління медичними прийомами в лікувальному закладі з використанням сучасних технологій Java, Spring Framework і JPA (Hibernate). Система забезпечує зручне додавання, редагування, видалення та перегляд даних про пацієнтів, лікарів, їхні спеціальності, медичні картки й записи на прийоми, пропонуючи інтуїтивно зрозумілий інтерфейс для роботи з інформацією.

Інформаційні системи в медицині відіграють ключову роль в автоматизації обліку пацієнтів і лікарів, спрощенні запису на прийоми, збереженні та обробці медичних даних, а також формуванні статистичних звітів для підвищення ефективності роботи закладів охорони здоров'я. Важливими аспектами таких систем є забезпечення цілісності, актуальності та безпеки даних, а також можливість їх оперативного оновлення.

Для реалізації проєкту обрано сучасний технологічний стек:

- **Java** забезпечує надійність, безпеку та масштабованість програмного продукту.
- **Spring Framework** спрощує створення веб-додатків, зокрема RESTful-сервісів, і підтримує гнучку архітектуру.
- **JPA (Hibernate)** полегшує роботу з базою даних завдяки об'єктно-реляційному відображенню та зручному управлінню сутностями.

Розробка системи включає кілька основних етапів. Спочатку аналізуються вимоги: вивчаються особливості роботи медичного закладу, визначаються ключові сутності (пацієнти, лікарі, спеціальності, прийоми, медичні картки) та їх взаємозв'язки, формулюється функціонал. На етапі проєктування створюється структура бази даних, розробляються REST-запити та обирається багаторівнева архітектура з поділом на контролери, сервіси та репозиторії. Далі реалізується код, створюється REST API для CRUD-операцій і впроваджується бізнес-логіка. Тестування передбачає перевірку модулів і системи загалом, а також

виправлення помилок. На етапі документування описується структура, функціонал і REST-інтерфейси, додаються інструкції для користувачів. Після впровадження система встановлюється в медичному закладі, проводиться навчання персоналу, забезпечується технічна підтримка та оновлення на основі зворотного зв'язку.

Розроблена система автоматизує управління медичними прийомами, підвищує якість обслуговування пацієнтів і ефективність роботи персоналу, забезпечуючи надійне зберігання та обробку даних відповідно до сучасних стандартів.

1. Аналіз предметної області

Постановка задачі

Медичний заклад щодня обслуговує значну кількість пацієнтів, які звертаються за консультаціями, діагностикою чи лікуванням до лікарів різних спеціальностей. Для ефективної організації роботи необхідна інформаційна система, яка забезпечує автоматизацію ключових процесів: облік даних про пацієнтів, лікарів і їхні спеціальності, організацію запису на прийоми, ведення медичних карток з історією хвороб і призначень, а також формування статистичних звітів для аналізу завантаженості персоналу та популярності спеціальностей.

Система має надавати зручний інтерфейс для роботи з даними, забезпечувати їх цілісність і актуальність, а також підтримувати можливість швидкого доступу до інформації та її оновлення. Такий підхід дозволить оптимізувати роботу медичного закладу, підвищити якість обслуговування пацієнтів і полегшити аналіз діяльності установи.

Основні бізнес-процеси

Розроблена система підтримує наступні сценарії використання:

Медичний заклад щодня обслуговує значну кількість пацієнтів, які звертаються за консультаціями, діагностикою чи лікуванням до лікарів різних спеціальностей. Для ефективної організації роботи необхідна інформаційна система, яка забезпечує автоматизацію ключових процесів: облік даних про пацієнтів, лікарів і їхні спеціальності, організацію запису на прийоми, ведення медичних карток з історією хвороб і призначень, а також формування статистичних звітів для аналізу завантаженості персоналу та популярності спеціальностей.

2. Проєктування програмного забезпечення

1. Проєктування структури даних

Для системи управління медичними прийомами визначено основні сутності та їх взаємозв'язки:

- **Patient (Пацієнт)**

Атрибути: id (Long), fullName (String), birthDate (LocalDate), phone (String), email (String)

Зв'язки: Один пацієнт може мати багато прийомів (OneToMany), одну медичну картку (OneToOne)

- **Doctor (Лікар)**

Атрибути: id (Long), fullName (String), email (String)

Зв'язки: Кожен лікар має одну спеціальність (ManyToOne), приймає багато пацієнтів через прийоми (OneToMany)

- **Specialty (Спеціальність)**

Атрибути: id (Long), name (String)

Зв'язки: Одна спеціальність може бути у багатьох лікарів (OneToMany)

- **Appointment (Прийом)**

Атрибути: id (Long), dateTime (LocalDateTime), notes (String)

Зв'язки: Кожен прийом пов'язаний з одним лікарем (ManyToOne) та одним пацієнтом (ManyToOne)

- **MedicalRecord (Медична картка)**

Атрибути: id (Long), diagnosis (String), treatment (String), createdAt (LocalDate)

Зв'язки: Кожна картка належить одному пацієнту (OneToOne)

2. Опис архітектури застосунку

Застосунок реалізовано за багаторівневою архітектурою з розподілом на такі рівні:

- **Controller**

Відповідає за прийом HTTP-запитів, валідацію вхідних даних та повернення відповідей клієнту. Викликає сервіси для виконання бізнес-логіки;

- **Service**

Містить бізнес-логіку: обробка даних, перевірки, взаємодія з репозиторіями. Забезпечує транзакційність операцій;

- **Repository**

Відповідає за безпосередню роботу з базою даних через JPA. Виконує CRUD-операції над сутностями.

3. Опис REST API

Нижче наведено перелік основних REST-запитів із описом їх призначення, структури запитів та відповідей (таблиця 2.1).

Таблиця 2.1- Перелік запитів API

№	Метод	URL	Призначення	Відповідь
1	POST	/patients	Додати нового пацієнта	Створений пацієнт з id
2	GET	/patients	Отримати список пацієнтів	Масив пацієнтів
3	PUT	/patients/{id}	Оновити дані пацієнта	Оновлений пацієнт
4	DELETE	/patients/{id}	Видалити пацієнта	Статус операції
5	POST	/doctors	Додати лікаря	Створений лікар з id
6	GET	/doctors	Отримати список лікарів	Масив лікарів
7	GET	/doctors /specialty	Отримати лікарів певної спеціальності	Масив лікарів певної спеціальності
8	GET	/doctors /patients	Отримати лікарів певного пацієнта	Масив лікарів певного пацієнта
9	GET	/doctors /patients_count	Отримати кількість пацієнтів	Кількість пацієнтів певного лікаря
10	PUT	/doctors/{id}	Оновити дані лікаря	Оновлений лікар
11	DELETE	/doctors/{id}	Видалити лікаря	Статус операції
12	POST	/specialties	Додати спеціальність	Створена спеціальність з id

№	Метод	URL	Призначення	Відповідь
13	GET	/specialties	Отримати список спеціальностей	Масив спеціальностей
14	POST	/appointments	Створити запис на прийом	Створений прийом з id
15	GET	/patients	Отримати відфільтровані прийоми	Масив прийомів
16	GET	/patients/{dateTime}	Прийоми на певну дату	Масив прийомів
17	PUT	/appointments/{id}	Оновити прийом	Оновлений прийом
18	DELETE	/appointments/{id}	Скасувати прийом	Статус операції
19	POST	/records	Додати медичну картку пацієнта	Створена картка з id
20	GET	/records/{id}	Отримати медичну історію	Медична картка
21	GET	/records/patient	Отримати медичну історію пацієнта	Медична картка
22	PUT	/records/{id}	Оновити запис медичної картки	Оновлена картка
23	DELETE	/records/{id}	Видалити запис картки	Статус операції
25	GET	/specialties/average	Отримати середню кількість прийомів по спеціальності	Середня кількість прийомів по спеціальності

4. Реалізація програмного продукту

Нижче наведено структуру реалізації основних компонентів системи управління медичними прийомами з описом їх призначення та ролі у програмному продукті.

1. Моделі (Entity-класи, DTO)

Entity у Spring Boot — це клас, який представляє таблицю в базі даних. Він позначається анотацією `@Entity`, а кожен його об'єкт відповідає одному рядку таблиці. Сутності використовуються для зберігання, читання, оновлення та видалення даних через JPA без написання SQL-запитів.

Основне призначення: моделювання структури даних бази та об'єктно-реляційне відображення (ORM).

Сутність Appointment

Сутність Appointment моделює запис на медичний прийом у системі, реалізований за допомогою Java, Spring Boot та Hibernate. Клас позначений анотацією `@Entity`, що визначає його як JPA-сутність, яка відображається на таблицю `appointment` у базі даних (через `@Table(name = "appointment")`).

Для скорочення шаблонного коду використовуються анотації Lombok:

- `@Data` — автоматично генерує гетери, сетери, `toString`, `equals`, `hashCode`;
- `@NoArgsConstructor` і `@AllArgsConstructor` — створюють конструктори без аргументів і з усіма полями відповідно.

Основні поля сутності:

- `id` — унікальний ідентифікатор запису (первинний ключ), генерується автоматично (`@Id, @GeneratedValue(strategy = GenerationType.IDENTITY)`).
- `doctor` — зв'язок багато-до-одного з сутністю `Doctor`, не може бути `null` (`@ManyToOne, @JoinColumn(nullable = false)`).
- `patient` — зв'язок багато-до-одного з сутністю `Patient`, також обов'язковий (`@ManyToOne, @JoinColumn(nullable = false)`).
- `dateTime` — дата та час прийому (`LocalDateTime`, анотація `@Column(name = "date_time")`).
- `notes` — текстові нотатки, що можуть містити додаткову інформацію про прийом (тип `String`, не є обов'язковим полем).

Сутність `Doctor`

Сутність `Doctor` відповідає за збереження інформації про лікарів у системі управління медичними прийомами. Вона є JPA-сутністю, що позначена анотацією `@Entity` та відображається на таблицю `doctor` у базі даних (через `@Table(name = "doctor")`).

Для автоматичного створення стандартних методів (гетери, сетери, конструктори тощо) використовується набір анотацій `Lombok`:

- `@Data` — генерує гетери/сетери, методи `equals`, `hashCode`, `toString`;
- `@NoArgsConstructor` і `@AllArgsConstructor` — створюють конструктори без і з параметрами відповідно.

Основні поля:

- `id` — унікальний ідентифікатор лікаря, первинний ключ таблиці (`@Id, @GeneratedValue(strategy = GenerationType.IDENTITY)`).

- `fullName` — повне ім'я лікаря; обов'язкове поле (`@Column(name = "full_name", nullable = false)`).
- `specialty` — зв'язок багато-до-одного з сутністю `Specialty`, що визначає спеціалізацію лікаря. Спеціальність є обов'язковою (`@ManyToOne, @JoinColumn(name = "specialty_id", nullable = false)`).
- `email` — необов'язкова електронна пошта лікаря (`@Column(name = "email")`).

Сутність **MedicalRecord**

Сутність `MedicalRecord` моделює медичні картки пацієнтів у системі та відповідає таблиці `medical_record` у базі даних. Вона позначена анотацією `@Entity` і зіставлена з відповідною таблицею через `@Table(name = "medical_record")`.

Для зменшення обсягу шаблонного коду використано анотації Lombok:

- `@Data` — автоматично генерує гетери, сетери, а також методи `toString`, `equals`, `hashCode`;
- `@NoArgsConstructor`, `@AllArgsConstructor` — генерують відповідні конструктори.

Основні поля:

- `id` — первинний ключ запису, автоматично генерується (`@Id, @GeneratedValue(strategy = GenerationType.IDENTITY)`);
- `patient` — посилання на пацієнта, якому належить ця медична картка; встановлюється зв'язок багато-до-одного із сутністю `Patient` (`@ManyToOne, @JoinColumn(name = "patient_id", nullable = false)`);
- `diagnosis` — текстовий опис діагнозу пацієнта (`@Column(name = "diagnosis")`);
- `treatment` — опис призначеного лікування (`@Column(name = "treatment")`);

- `createdAt` — дата створення медичної картки; поле типу `LocalDate`, що фіксує час додавання запису (`@Column(name = "created_at")`).

Сутність Patient

Сутність Patient моделює інформацію про пацієнтів у медичній системі та відповідає таблиці patient у базі даних. Вона позначена анотацією `@Entity` і зіставлена з відповідною таблицею через `@Table(name = "patient")`.

Для зменшення обсягу шаблонного коду використано анотації Lombok:

- `@Data` — автоматично генерує гетери, сетери, а також методи `toString`, `equals`, `hashCode`;
- `@NoArgsConstructor`, `@AllArgsConstructor` — генерують відповідні конструктори.

Основні поля:

- `id` — первинний ключ пацієнта, автоматично генерується (`@Id`, `@GeneratedValue(strategy = GenerationType.IDENTITY)`);
- `fullName` — обов'язкове текстове поле для збереження повного імені пацієнта (`@Column(name = "full_name", nullable = false)`);
- `birthDate` — дата народження пацієнта (`LocalDate`), зберігається у відповідному стовпці таблиці (`@Column(name = "birth_date")`);
- `phone` — номер телефону пацієнта (`@Column(name = "phone")`);
- `email` — електронна пошта пацієнта (`@Column(name = "email")`).

Сутність Specialty

Сутність `Specialty` моделює медичні спеціальності лікарів у системі управління прийомами та відповідає таблиці `specialty` у базі даних. Вона позначена анотацією `@Entity` і зіставлена з відповідною таблицею через `@Table(name = "specialty")`.

Для автоматизації створення службового коду використано анотації Lombok:

- `@Data` — генерує гетери, сетери, методи `toString`, `equals`, `hashCode`;
- `@NoArgsConstructor`, `@AllArgsConstructor` — генерують конструктори без і з усіма полями відповідно.

Основні поля:

- `id` — первинний ключ спеціальності, що генерується автоматично (`@Id`, `@GeneratedValue(strategy = GenerationType.IDENTITY)`);
- `name` — назва спеціальності (наприклад, *кардіолог*, *терапевт*), обов'язкове текстове поле (`@Column(name = "name", nullable = false)`).

Сутність `Specialty` пов'язана з лікарями (`Doctor`) через зв'язок "один до багатьох", де одна спеціальність може бути призначена кільком лікарям. Вона забезпечує категоризацію медичного персоналу, використовується при створенні та фільтрації прийомів, а також у сервісному та репозиторному рівнях (`SpecialtyService`, `SpecialtyRepository`) для управління медичними напрямками.

Сутність User

Сутність User моделює дані користувачів системи та відповідає таблиці users у базі даних. Вона позначена анотацією @Entity і зіставлена з таблицею через @Table(name = "users").

Для мінімізації шаблонного коду застосовано анотації Lombok:

- @Data — автоматично генерує гетери, сетери, методи toString, equals, hashCode;
- @NoArgsConstructor, @AllArgsConstructor — створюють конструктори без параметрів і з усіма полями відповідно.

Основні поля:

- id — унікальний ідентифікатор користувача, первинний ключ із автоматичною генерацією (@Id, @GeneratedValue(strategy = GenerationType.IDENTITY));
- username — унікальне ім'я користувача, обов'язкове поле (@Column(nullable = false, unique = true));
- password — обов'язковий пароль користувача (@Column(nullable = false));
- email — унікальна, але необов'язкова електронна пошта (@Column(unique = true));
- provider — необов'язкове поле, що зберігає провайдера автентифікації (наприклад, "google"), використовується для OAuth2.

DTO-класи для автентифікації та реєстрації

У вебзастосунках, зокрема під час реалізації механізмів автентифікації та реєстрації, використовуються DTO (Data Transfer Object) — об'єкти, призначені виключно для передачі даних між клієнтом і сервером. Вони не містять бізнес-логіки та не є частиною доменної моделі (Entity), що забезпечує ізоляцію внутрішньої структури застосунку та покращує безпеку.

AuthRequest

Цей DTO-клас призначений для прийому даних автентифікації з боку клієнта. Він містить два обов'язкові поля: ім'я користувача (username) та пароль (password). Зазвичай використовується в методах контролера, які обробляють POST-запити на вхід до системи. Завдяки простій структурі, клас зручно інтегрується з механізмами Spring Security або власними сервісами авторизації.

AuthResponse

Цей клас використовується для формування відповіді на запит автентифікації. Містить єдине поле — токен авторизації (зазвичай JWT), який клієнт використовує для подальших запитів до захищених ресурсів. DTO дозволяє чітко відокремити інформацію, яка повертається клієнту, не розкриваючи внутрішню реалізацію механізмів безпеки.

RegisterRequest

DTO-клас, призначений для обробки запитів на реєстрацію нових користувачів. Як правило, містить такі поля, як ім'я користувача, пароль і адреса електронної пошти. Застосовується на рівні контролера для зчитування вхідних даних та їх подальшої обробки сервісом реєстрації. Цей клас сприяє спрощенню валідації й обробки запитів, забезпечуючи необхідний мінімум інформації для створення нового облікового запису.

2. Репозиторії (інтерфейси доступу до даних)

Repository — це інтерфейс, який відповідає за доступ до бази даних. Він розширює інтерфейси Spring Data JPA, наприклад JpaRepository, і дозволяє виконувати CRUD-операції без написання SQL. Spring автоматично створює реалізацію цього інтерфейсу під час запуску програми.

Основне призначення репозиторію в Spring Boot:

- Виконання CRUD-операцій (створення, читання, оновлення, видалення)
- Автоматичне генерування SQL-запитів за іменами методів
- Робота з об'єктами сутностей без необхідності вручну писати SQL

Інтерфейс AppointmentRepository

Інтерфейс AppointmentRepository забезпечує доступ до даних про медичні прийоми та відповідає за взаємодію з таблицею прийомів у базі даних. Він розширює JpaRepository<Appointment, Long>, що надає повний набір стандартних CRUD-операцій.

Для реалізації більш гнучких запитів у цьому інтерфейсі визначено два методи з використанням анотації @Query:

- `findByDate` — повертає список прийомів, які точно відповідають заданій даті й часу. Запит використовує параметр `dateTime` типу `LocalDateTime`.
- `getFilteredAppointments` — здійснює фільтрацію прийомів за наступними критеріями:
 - `doctorId` — ідентифікатор лікаря;
 - `patientId` — ідентифікатор пацієнта;
 - `future` — булевий параметр, що вказує на необхідність відображення лише майбутніх прийомів.

Інтерфейс DoctorRepository

Інтерфейс `DoctorRepository` відповідає за доступ до даних про лікарів у системі та реалізує розширення стандартного репозиторію `JpaRepository<Doctor, Long>`, що забезпечує базові CRUD-операції над сутністю `Doctor`.

Для розширення функціональності та реалізації спеціалізованих запитів до бази даних у межах цього інтерфейсу визначено три методи з анотацією `@Query`:

- `getPatientsByDoctorId` — повертає список пацієнтів, які мають запис на прийом до лікаря з вказаним ідентифікатором. Запит ґрунтується на вибірці з таблиці прийомів, що дозволяє отримати лише тих пацієнтів, які взаємодіяли з конкретним лікарем.
- `getDoctorsBySpecialtyId` — здійснює пошук лікарів за вказаною спеціалізацією. Метод корисний для групування або відбору лікарів за напрямками медичної практики.
- `countPatientsByDoctorId` — виконує підрахунок кількості пацієнтів, що мали прийоми у конкретного лікаря. Це дозволяє аналізувати рівень завантаженості лікаря та статистику його прийомів.

Інтерфейс `MedicalRecordRepository`

Інтерфейс `MedicalRecordRepository` забезпечує доступ до медичних записів пацієнтів, реалізуючи типову функціональність репозиторію через розширення `JpaRepository<MedicalRecord, Long>`. Це дозволяє виконувати базові операції створення, читання, оновлення та видалення записів типу `MedicalRecord`.

Основний користувацький метод:

- `findMedicalRecordByPatient` — виконує вибірку медичних карток за ідентифікатором пацієнта. Запит надає доступ до повної історії

хвороб конкретного пацієнта, що є критично важливим для подальшого надання медичної допомоги та супроводу лікування.

Інтерфейс PatientRepository

Інтерфейс PatientRepository відповідає за зберігання та доступ до даних про пацієнтів у медичній інформаційній системі. Він реалізує розширення стандартного інтерфейсу JpaRepository<Patient, Long>, що надає повний набір CRUD-операцій — створення, читання, оновлення та видалення записів про пацієнтів.

На поточному етапі PatientRepository не містить спеціалізованих запитів або додаткових методів, однак забезпечує достатню функціональність для базової роботи з даними пацієнтів. За потреби, інтерфейс може бути доповнений методами.

Інтерфейс SpecialtyRepository

Інтерфейс SpecialtyRepository відповідає за роботу з даними про медичні спеціальності та реалізує стандартний набір CRUD-операцій через розширення JpaRepository<Specialty, Long>. Він дозволяє зберігати, оновлювати й вилучати інформацію про спеціалізації лікарів, а також виконує роль аналітичного інструменту.

Інтерфейс містить два користувацькі методи з використанням анотації @Query:

- countBySpecialty — підраховує кількість прийомів, які стосуються певної спеціальності, за її ідентифікатором. Метод базується на зв'язку лікаря з прийомами та спеціалізацією, що дозволяє оцінити попит на конкретний напрям медицини.
- countAllAppointments — повертає загальну кількість прийомів у системі. Цей метод може застосовуватись для обчислення відносних

показників, зокрема частки прийомів за спеціальністю у загальному обсязі.

Інтерфейс UserRepository

Інтерфейс UserRepository виконує функцію доступу до облікових записів користувачів системи та реалізує стандартні CRUD-операції шляхом розширення JpaRepository<User, Long>. Його основне призначення — підтримка механізмів автентифікації та авторизації у вебзастосунку.

Інтерфейс містить два спеціалізовані методи:

- findByUsername — здійснює пошук користувача за іменем користувача (username). Повертає об'єкт Optional<User>, що дозволяє безпечно обробляти результат наявності або відсутності відповідного запису.
- findByEmail — виконує аналогічний пошук за електронною поштою (email). Також повертає Optional<User> для зручності перевірки.

3. Сервіси (бізнес-логіка)

Service — це клас, який містить бізнес-логіку додатку. Позначається анотацією @Service. Використовується для обробки даних, перевірок, трансформацій тощо.

Основне призначення: інкапсуляція бізнес-логіки та координація між контролером і репозиторієм.

Сервіс AppointmentService

Клас AppointmentService є сервісним компонентом, що реалізує бізнес-логіку для управління призначеннями пацієнтів у медичній системі. Він позначений анотацією @Service, що вказує на його роль у контексті Spring Framework як класу, відповідального за логіку прикладного рівня.

Основні завдання AppointmentService полягають в інкапсуляції доступу до бази даних через AppointmentRepository, а також у наданні зручного інтерфейсу для контролерів, що керують запитами від користувачів.

Функціональність класу охоплює:

- Отримання списку прийомів за певною датою або з урахуванням фільтрів (ідентифікатор лікаря, пацієнта, чи є прийом майбутнім).
- Пошук прийому за ідентифікатором, з генерацією винятку у випадку його відсутності.
- Створення нового прийому на основі переданого об'єкта.
- Оновлення наявного запису, що включає зміну дати та приміток.
- Видалення прийому за ідентифікатором.

Сервіс DoctorService

Клас DoctorService є компонентом рівня сервісів, що реалізує бізнес-логіку для роботи з лікарями. Він позначений анотацією @Service, що вказує на його включення до контексту Spring як сервісного шару, відповідального за обробку запитів, пов'язаних з лікарями.

Основною функцією DoctorService є інкапсуляція взаємодії з DoctorRepository для забезпечення таких операцій:

- Отримання повного списку лікарів, що містяться у системі.
- Пошук лікарів за спеціальністю через ідентифікатор спеціальності.
- Пошук конкретного лікаря за його ідентифікатором, з генерацією винятку у разі його відсутності.
- Отримання списку пацієнтів лікаря, що записані на прийом до нього.

- Підрахунок кількості пацієнтів, яких обслуговує конкретний лікар.
- Додавання нового лікаря до системи.
- Оновлення наявного запису лікаря, з можливістю змінити ім'я, електронну пошту та спеціальність.
- Видалення лікаря за його унікальним ідентифікатором.

Сервіс MedicalRecordService

Клас MedicalRecordService виконує роль сервісного компонента для управління медичними записами пацієнтів. Він позначений анотацією @Service, що свідчить про його призначення як частини сервісного шару програми.

Основне завдання цього класу — інкапсуляція бізнес-логіки, пов'язаної з медичними картками пацієнтів, і делегування базових операцій доступу до даних репозиторію MedicalRecordRepository.

Функціональні можливості включають:

- Отримання повного списку медичних записів, що містяться у системі.
- Пошук записів за ідентифікатором пацієнта, що дозволяє відобразити його історію лікування.
- Отримання конкретного запису за його унікальним ідентифікатором, із винятковим повідомленням у разі відсутності.
- Створення нового медичного запису шляхом збереження нового об'єкта.
- Оновлення наявного запису, з можливістю зміни діагнозу, лікування та дати створення.
- Видалення запису за ідентифікатором, що дає змогу підтримувати актуальність бази.

Сервіс PatientService

Клас PatientService є сервісним компонентом, відповідальним за управління інформацією про пацієнтів у системі. Позначений анотацією @Service, він інкапсулює бізнес-логіку, пов'язану з операціями над сутністю пацієнта.

Основні функції сервісу охоплюють:

- Отримання повного списку пацієнтів із бази даних.
- Пошук пацієнта за унікальним ідентифікатором із винятковою обробкою випадку відсутності.
- Додавання нового пацієнта шляхом збереження нового запису.
- Оновлення існуючих даних пацієнта, включно з ім'ям, датою народження, електронною поштою та телефоном.
- Видалення пацієнта за ідентифікатором.

Сервіс SpecialtyService

Клас SpecialtyService є сервісним компонентом, який реалізує бізнес-логіку для роботи зі спеціальностями лікарів. Він анотований як @Service, що дозволяє Spring автоматично визначати його як компонент для ін'єкції залежностей у відповідні місця програми.

Основні функції класу:

- Отримання повного списку спеціальностей, що містяться у системі.
- Додавання нової спеціальності до бази даних.
- Обчислення середнього співвідношення кількості прийомів до лікарів певної спеціальності відносно загальної кількості всіх прийомів.

Для цього використовується два методи з репозиторію

SpecialtyRepository: countBySpecialty, який підраховує кількість записів

для спеціальності за її ідентифікатором, і `countAllAppointments`, який повертає загальну кількість записів.

Сервіс JwtService

Клас `JwtService` — сервісний компонент, що реалізує механізми створення та валідації JWT (JSON Web Token), які використовуються для автентифікації та авторизації користувачів у системі. Анотований як `@Service`, клас забезпечує централізовану обробку токенів у Spring Boot-застосунку.

Функціональність:

- Метод `generateToken(String username)` створює підписаний JWT-токен, у якому записується ім'я користувача як суб'єкт (`subject`). Токен має термін дії 1 добу (24 години) і підписується за допомогою алгоритму HMAC SHA-256 з використанням секретного ключа `secret`.
- Метод `extractUsername(String token)` розшифровує переданий токен, перевіряє його підпис і витягує значення поля `subject`, тобто ім'я користувача, яке використовувалось під час генерації токена.

Клас використовує бібліотеку `io.jsonwebtoken (JJWT)` для роботи з токенами. Він є критично важливим елементом системи безпеки, зокрема в компонентах `JwtFilter` та `AuthController`, де перевіряється автентичність запитів користувача.

4. Контролери (обробка HTTP-запитів)

Controller — це клас, який обробляє HTTP-запити користувачів. Позначається анотацією @RestController або @Controller. Приймає вхідні дані, викликає методи сервісу та повертає результат.

Основне призначення: забезпечення взаємодії між клієнтом (фронтендом) і серверною логікою (сервісами).

Контролер AppointmentController

AppointmentController — REST-контролер, який відповідає за обробку HTTP-запитів, пов'язаних із записами на медичні прийоми. Анотований як @RestController та прив'язаний до шляху /appointments, він забезпечує доступ до CRUD-операцій через RESTful API.

Контролер взаємодіє з сервісним рівнем через ін'єкцію залежності AppointmentService, делегуючи йому всю бізнес-логіку.

Основні ендпоінти:

- GET /appointments — повертає список записів, відфільтрованих за лікарем (doctorId), пацієнтом (patientId) або майбутніми датами (future). Параметри є необов'язковими.
- GET /appointments/{dateTime} — повертає список записів на конкретну дату та час. Значення dateTime передається як змінна шляху (@PathVariable) у форматі LocalDateTime.
- POST /appointments — створює новий запис, приймаючи об'єкт типу Appointment у тілі запиту (@RequestBody).
- PUT /appointments/{id} — оновлює наявний запис за переданим ідентифікатором. Дані запису передаються в тілі запиту, а ідентифікатор — через URL.
- DELETE /appointments/{id} — видаляє запис за його унікальним ідентифікатором.

Контролер AuthController

AuthController — REST-контролер, відповідальний за обробку запитів, пов'язаних із автентифікацією та реєстрацією користувачів. Анотований як @RestController і прив'язаний до префіксу маршруту /api/auth, він забезпечує два ключові ендпоінти для управління доступом до системи.

Контролер використовує ін'єкції залежностей для взаємодії з такими компонентами:

- UserRepository — для доступу до даних користувачів;
- JwtService — для генерації JWT-токенів;
- PasswordEncoder — для хешування та перевірки паролів.

Основні ендпоінти:



- POST /api/auth/register — реєструє нового користувача на основі об'єкта RegisterRequest, що містить ім'я, пароль і email. Якщо ім'я користувача вже існує (перевірка через findByUsername), повертає статус 409 CONFLICT з повідомленням "Username already exists". У іншому випадку створює нового користувача із хешованим паролем, зберігає його в базі даних та повертає статус 200 OK з повідомленням "User registered successfully".
- POST /api/auth/login — обробляє вхід користувача на основі AuthRequest, що включає ім'я користувача та пароль. Якщо користувач не знайдений або пароль не відповідає (перевірка через passwordEncoder.matches), повертає 401 UNAUTHORIZED з повідомленням "Invalid credentials". У разі успішної автентифікації створює JWT-токен за допомогою jwtService.generateToken та повертає його у вигляді AuthResponse.

Контролер AuthTestController

AuthTestController — допоміжний REST-контролер, призначений для перевірки автентифікації користувачів після входу в систему. Анотований як @RestController і прив'язаний до маршруту /api/test, він забезпечує два GET-ендпоінти для тестування доступу через різні механізми автентифікації.

Контролер використовує об'єкт Authentication, автоматично наданий Spring Security, який містить інформацію про поточного автентифікованого користувача.

Основні ендпоінти:

- GET /api/test/jwt — перевіряє наявність і дійсність JWT-токена. Метод testJwt() приймає об'єкт Authentication та повертає HTTP-відповідь зі статусом 200 OK і повідомленням у форматі "  Access granted via JWT! Hello, <username>", де <username> — ім'я користувача, отримане з authentication.getName().
- GET /api/test/oauth — перевіряє автентифікацію через зовнішній OAuth2-провайдер (наприклад, GitHub). Метод testOAuth() також приймає об'єкт Authentication і повертає HTTP-відповідь зі статусом 200 OK та повідомленням "  Access granted via OAuth2! Welcome, <username>".

Контролер DoctorController

DoctorController — це REST-контролер, який реалізує інтерфейс доступу до інформації про лікарів та пов'язані з ними дані. Анотований як @RestController і прив'язаний до URL-префіксу /doctors, контролер взаємодіє з шаром сервісів через ін'єкцію залежності DoctorService, що забезпечує відповідну бізнес-логіку.

Основні ендпоінти:

- GET /doctors — метод `getAllDoctors()` повертає повний список лікарів, отриманий з `doctorService.getAllDoctors()`.
- GET /doctors/specialty?id=... — метод `getDoctorsBySpecialty(@RequestParam long id)` повертає список лікарів, що мають вказану спеціальність, передану через параметр `id`. Логіку забезпечує `doctorService.getDoctorsBySpecialty(id)`.
- GET /doctors/patients?id=... — метод `getPatientsById(@RequestParam long id)` повертає список пацієнтів, прикріплених до лікаря з вказаним ідентифікатором. Делегується у `doctorService.getPatientsByDoctorId(id)`.
- GET /doctors/patients_count?id=... — метод `getPatientsCountById(@RequestParam long id)` повертає числове значення кількості пацієнтів, прикріплених до лікаря. Реалізується через `doctorService.countPatients(id)`.
- POST /doctors — метод `addDoctor(@RequestBody Doctor doctor)` додає нового лікаря, дані якого передаються в тілі запиту, і повертає створений об'єкт. Відповідає виклику `doctorService.addDoctor(doctor)`.
- PUT /doctors/{id} — метод `updateDoctor(@PathVariable long id, @RequestBody Doctor doctor)` оновлює інформацію про лікаря за вказаним ID, використовуючи нові дані з тіла запиту. Викликає `doctorService.updateDoctor(id, doctor)`.
- DELETE /doctors/{id} — метод `deleteDoctor(@PathVariable long id)` видаляє лікаря з бази за переданим ідентифікатором, викликаючи `doctorService.deleteDoctor(id)`.

Контролер **MedicalRecordController**

MedicalRecordController — це REST-контролер, який забезпечує інтерфейс доступу до медичних записів пацієнтів. Анотований як

@RestController та прив'язаний до префіксу шляху /records, він делегує всю бізнес-логіку компоненту MedicalRecordService, ін'єктованому через конструктор.

Основні ендпоінти:

- GET /records — метод getAllRecords() повертає список усіх медичних записів у системі, викликаючи medicalRecordService.getAllMedicalRecords().
- GET /records/{id} — метод getMedicalRecord(@PathVariable long id) повертає окремий медичний запис за його ідентифікатором через medicalRecordService.getMedicalRecordById(id).
- GET /records/patient?id=... — метод getMedicalRecordByPatient(@RequestParam long id) повертає список медичних записів, які належать пацієнту з переданим ID. Реалізується через medicalRecordService.getMedicalRecordByPatient(id).
- POST /records — метод addMedicalRecord(@RequestBody MedicalRecord medicalRecord) додає новий медичний запис, дані якого надаються у тілі запиту, і повертає створений об'єкт. Делегує виклик medicalRecordService.addMedicalRecord(medicalRecord).
- PUT /records/{id} — метод updateMedicalRecord(...) оновлює медичний запис за переданим ID, приймаючи оновлені дані у тілі запиту.
- DELETE /records/{id} — метод deleteMedicalRecord(@PathVariable long id) видаляє медичний запис за вказаним ідентифікатором, викликаючи medicalRecordService.deleteMedicalRecord(id).

Контролер PatientController

PatientController — це REST-контролер, який відповідає за керування сутністю пацієнтів у медичній системі. Анотований як @RestController, він

обробляє HTTP-запити на шляху з префіксом /patients та делегує бізнес-логіку сервісному шару PatientService, який ін'єктується через конструктор.

Основні методи контролера:

GET /patients — метод getAllPatients() повертає список усіх пацієнтів.
Викликає метод patientService.getAllPatients().

POST /patients — метод addPatient(@RequestBody Patient patient) дозволяє створити нового пацієнта, передаючи дані у форматі JSON в тілі запиту.
Повертає створений об'єкт після виклику patientService.addPatient(patient).

PUT /patients/{id} — метод updatePatient(...) відповідає за оновлення даних пацієнта за вказаним ID. Однак у поточній реалізації є помилка: параметр patient не має анотації @RequestBody, через що Spring Boot не зможе автоматично десеріалізувати JSON-запит. Правильна реалізація:

DELETE /patients/{id} — метод deletePatient(@PathVariable long id) видаляє пацієнта за заданим ідентифікатором, викликаючи patientService.deletePatient(id).

Контролер SpecialtyController

SpecialtyController — це REST-контролер, який забезпечує доступ до даних про лікарські спеціальності. Анотований як @RestController та прив'язаний до URL-префіксу /specialties, він делегує логіку обробки запитів сервісному шару SpecialtyService, ін'єктованому через конструктор.

Реалізовані методи контролера:

- GET /specialties — метод getAllSpecialties() повертає список усіх спеціальностей, використовуючи specialtyService.getAllSpecialties().
- GET /specialties/average?id=... — метод getSpecialtyAverage(@RequestParam long id) обчислює середнє значення (наприклад, кількість прийомів чи інший узагальнений показник) для

спеціальності з вказаним ID. Використовує

`specialtyService.getSpecialtyAverage(id)`.

- `POST /specialties` — метод `addSpecialty(@RequestBody Specialty specialty)` створює нову спеціальність, передаючи дані у форматі JSON. Повертає створену спеціальність після виклику `specialtyService.addSpecialty(specialty)`.

5. Конфігурація

Клас `JwtFilter`

`JwtFilter` — це компонент Spring (`@Component`), який розширює клас `OncePerRequestFilter` та реалізує механізм автентифікації на основі JWT-токенів. Він автоматично перехоплює кожен HTTP-запит до системи, перевіряючи, чи запит містить дійсний токен авторизації.

Основна логіка роботи:

1. Перевірка заголовка `Authorization`

Метод `doFilterInternal(...)` перевіряє, чи в запиті наявний заголовок `Authorization` та чи починається він із префікса `"Bearer "`. Якщо так — із заголовка вилучається токен.

2. Отримання імені користувача

Із витягнутого токена за допомогою `JwtService.extractUsername(...)` визначається ім'я користувача. Якщо воно присутнє і ще не встановлена автентифікація у `SecurityContextHolder`, відбувається пошук користувача в `UserRepository`.

3. Створення об'єкта автентифікації

Якщо користувач знайдений, створюється об'єкт `UsernamePasswordAuthenticationToken` (з іменем користувача, відсутнім паролем і порожнім списком прав доступу). Цей об'єкт встановлюється в контекст безпеки `SecurityContextHolder`, авторизуючи користувача.

4. Продовження ланцюга фільтрації

Незалежно від результату автентифікації, метод викликає `filterChain.doFilter(request, response)` для передачі запиту далі по ланцюгу фільтрів.

Використані залежності:

- `JwtService` — сервіс для роботи з JWT (витяг імені користувача).
- `UserRepository` — репозиторій для пошуку користувача за ім'ям.
- `SecurityContextHolder` — глобальний контекст безпеки для поточного потоку.
- `UsernamePasswordAuthenticationToken` — об'єкт для зберігання інформації про автентифікованого користувача.

Клас OAuth2LoginSuccessHandler

`OAuth2LoginSuccessHandler` — це компонент Spring (`@Component`), який реалізує інтерфейс `AuthenticationSuccessHandler` і відповідає за обробку успішної автентифікації користувача через OAuth2 (наприклад, Google, Facebook).

Основна логіка роботи:

1. Отримання інформації про користувача OAuth2

Після успішної автентифікації в методі `onAuthenticationSuccess(...)` отримується об'єкт `OAuth2User` із поточної аутентифікації, з якого за допомогою `getAttribute("email")` вилучається email користувача.

2. Перевірка наявності користувача в базі

Через `UserRepository` виконується пошук користувача за email. Якщо користувач відсутній, створюється новий об'єкт `User` з таким email,

порожнім паролем (оскільки аутентифікація здійснюється через зовнішнього провайдера), і вказується провайдер — "google".

3. Генерація JWT-токена

Для імені користувача (username) генерується JWT-токен за допомогою `JwtService.generateToken(...)`.

4. Перенаправлення користувача

Користувача перенаправляють на URL `/login/success?token=...`, де у параметрі `token` передається створений JWT. Це дозволяє фронтенду отримати токен для подальшої роботи з авторизацією.

Використані залежності:

- `UserRepository` — репозиторій для доступу до даних користувачів.
- `JwtService` — сервіс для створення JWT-токенів.
- `OAuth2User` — представлення користувача, отриманого від OAuth2 провайдера.
- `Authentication` — об'єкт Spring Security, що містить інформацію про поточну аутентифікацію.

Клас `SecurityConfig`

`SecurityConfig` — це конфігураційний клас безпеки для Spring Security, який визначає правила захисту HTTP-запитів у додатку.

Основні елементи конфігурації:

- Анотації:
 - `@Configuration` — вказує, що клас є конфігураційним біном Spring.
 - `@EnableWebSecurity` — активує підтримку Spring Security для веб-додатка.

- SecurityFilterChain (метод filterChain):

Конфігурує основний ланцюжок фільтрів безпеки з такими налаштуваннями:

- Відключає захист від CSRF (Cross-Site Request Forgery) (csrf().disable()), що часто використовується в API.
- Дозволяє вільний доступ (без аутентифікації) до публічних маршрутів:
 - /api/auth/** — маршрути для аутентифікації (реєстрація, вхід).
 - /oauth2/** — маршрути для OAuth2-логіну.
 - /login — сторінка входу.
- Всі інші запити вимагають аутентифікації (anyRequest().authenticated()).
- Налаштовує OAuth2-логін із власним обробником успішного входу (successHandler(oAuth2LoginSuccessHandler)).
- Додає JwtFilter у ланцюжок фільтрів перед UsernamePasswordAuthenticationFilter для перевірки JWT-токена у заголовках запитів.

- PasswordEncoder (метод passwordEncoder):

Визначає бін PasswordEncoder, який використовує алгоритм BCryptPasswordEncoder для безпечного хешування паролів.

Залежності, що інжектуються:

- JwtFilter — фільтр для обробки JWT-токенів у запитах.
- OAuth2LoginSuccessHandler — обробник, що спрацьовує після успішної OAuth2-автентифікації.

5.Тестування та налагодження

Для перевірки працездатності системи було використано ручне тестування основних REST-контролерів та інструмент Postman для моделювання HTTP-запитів до API. Тестування проводилося поетапно після реалізації кожного модуля: автентифікації, роботи з пацієнтами, лікарями, записами на прийом, медичними записами та спеціалізаціями.

За допомогою Postman були перевірені основні запити:

- POST /api/auth/register — реєстрація користувача (успішний сценарій та обробка дублікатів логінів).

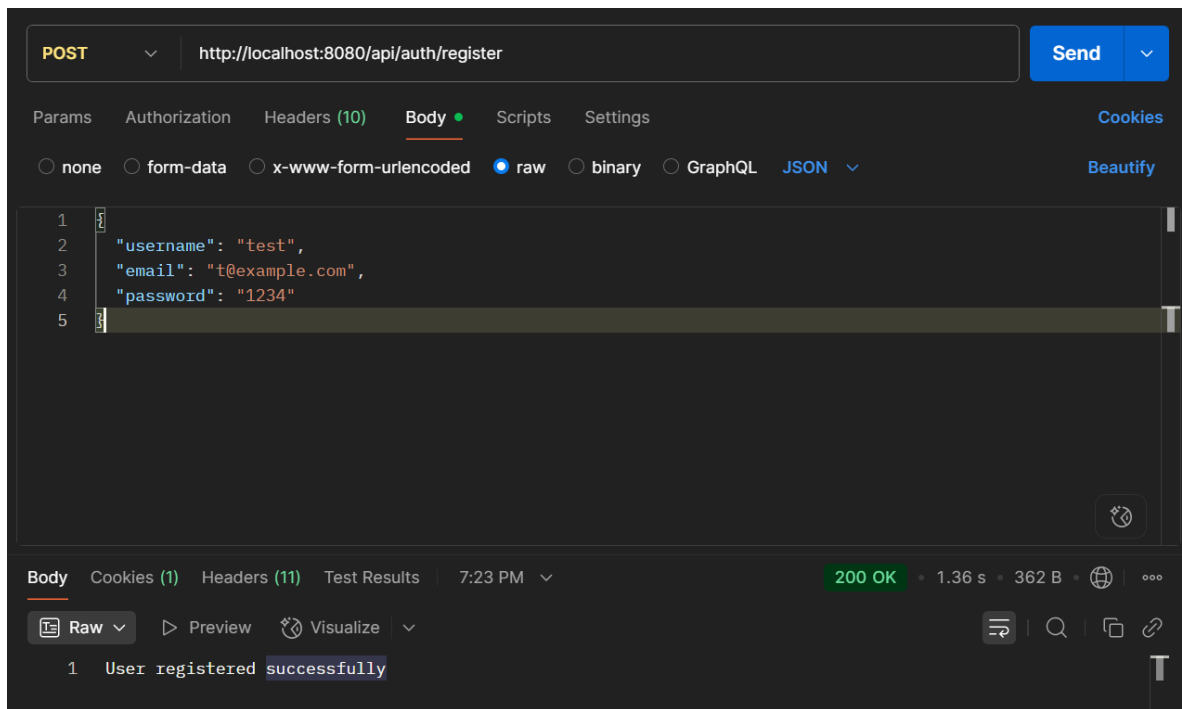


Рисунок 5.1 – Тестування реєстрації користувача

- POST /api/auth/login — автентифікація користувача та отримання JWT.

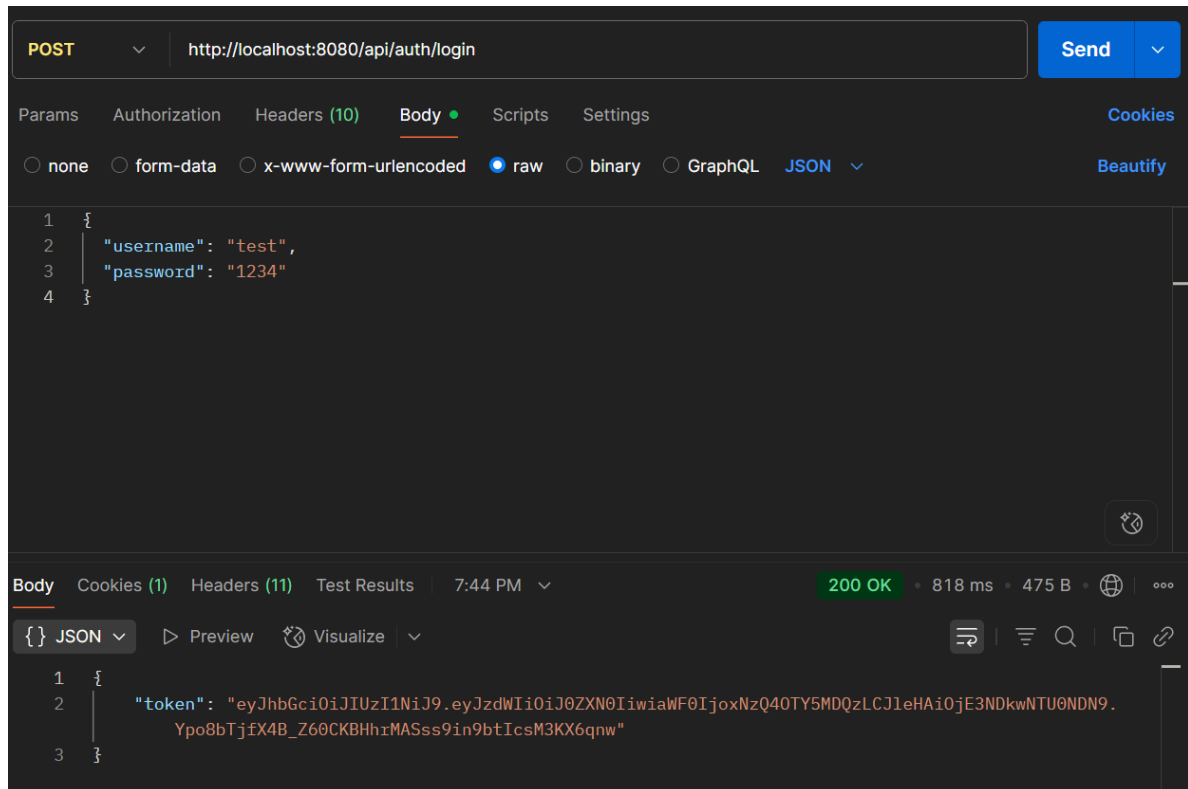


Рисунок 5.2 – Тестування автентифікації користувача

- GET /appointments?doctorId=1 — фільтрація записів за лікарем.

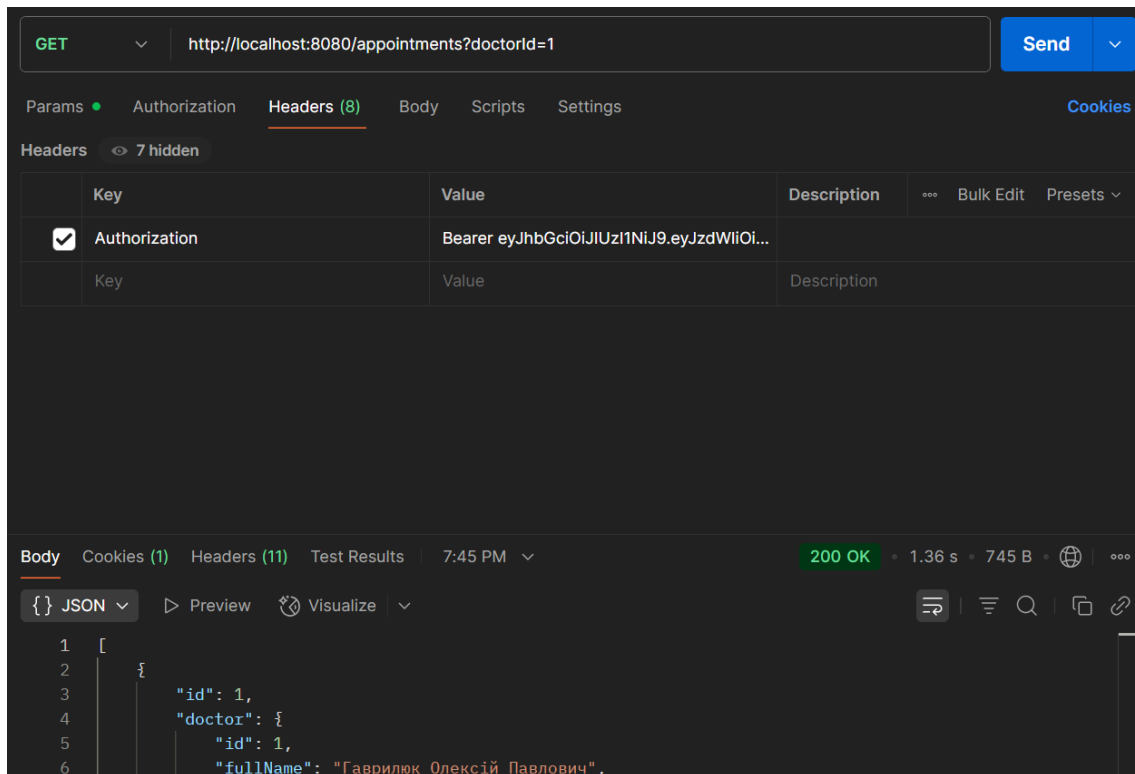


Рисунок 5.3 – Тестування GET-request таблиці appointments

- GET /records/patient?id=3 — отримання медичних записів для пацієнта.

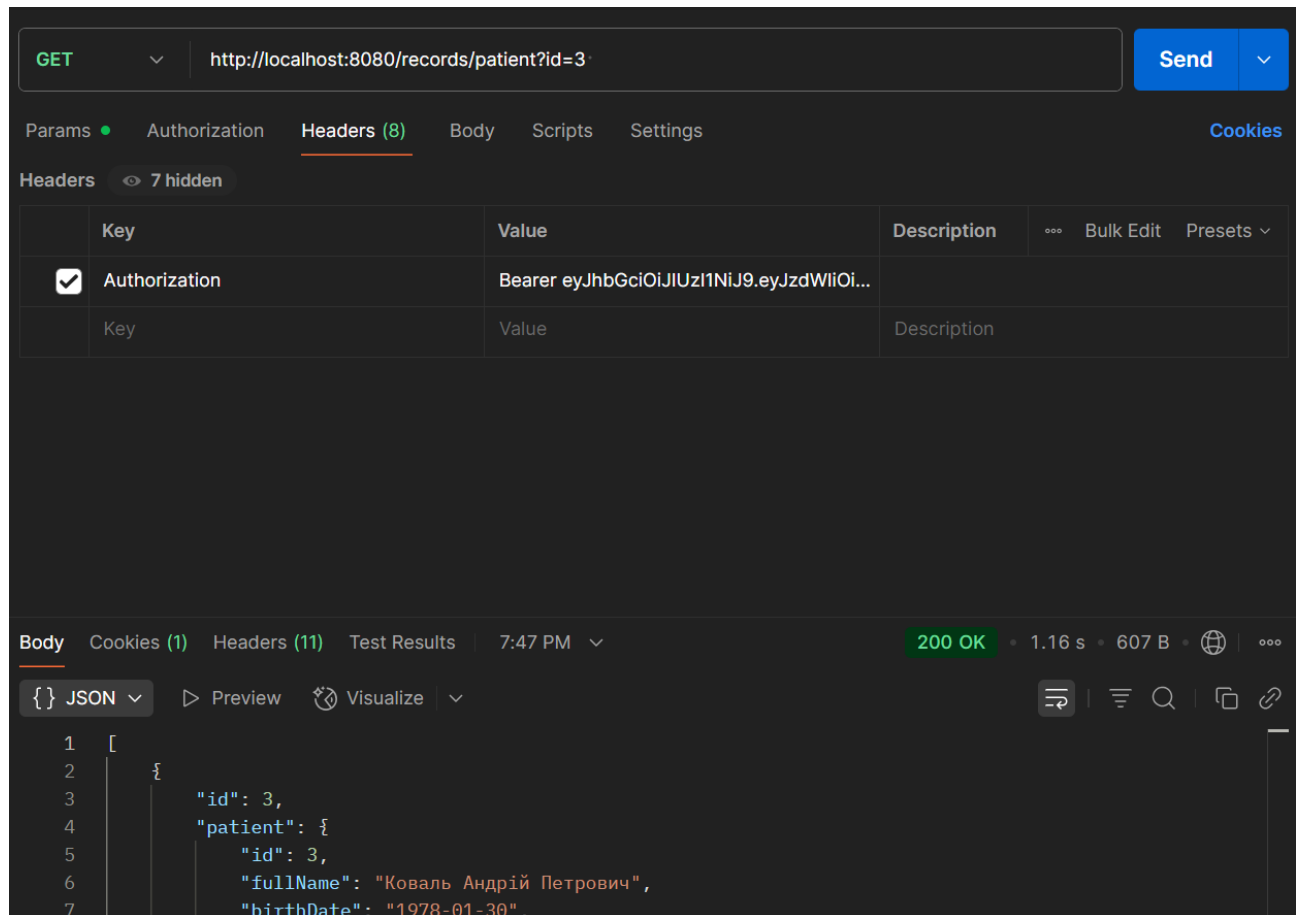


Рисунок 5.4 – Тестування GET-request таблиці records

- GET /api/test/jwt — перевірка доступу до захищених маршрутів за допомогою токена.

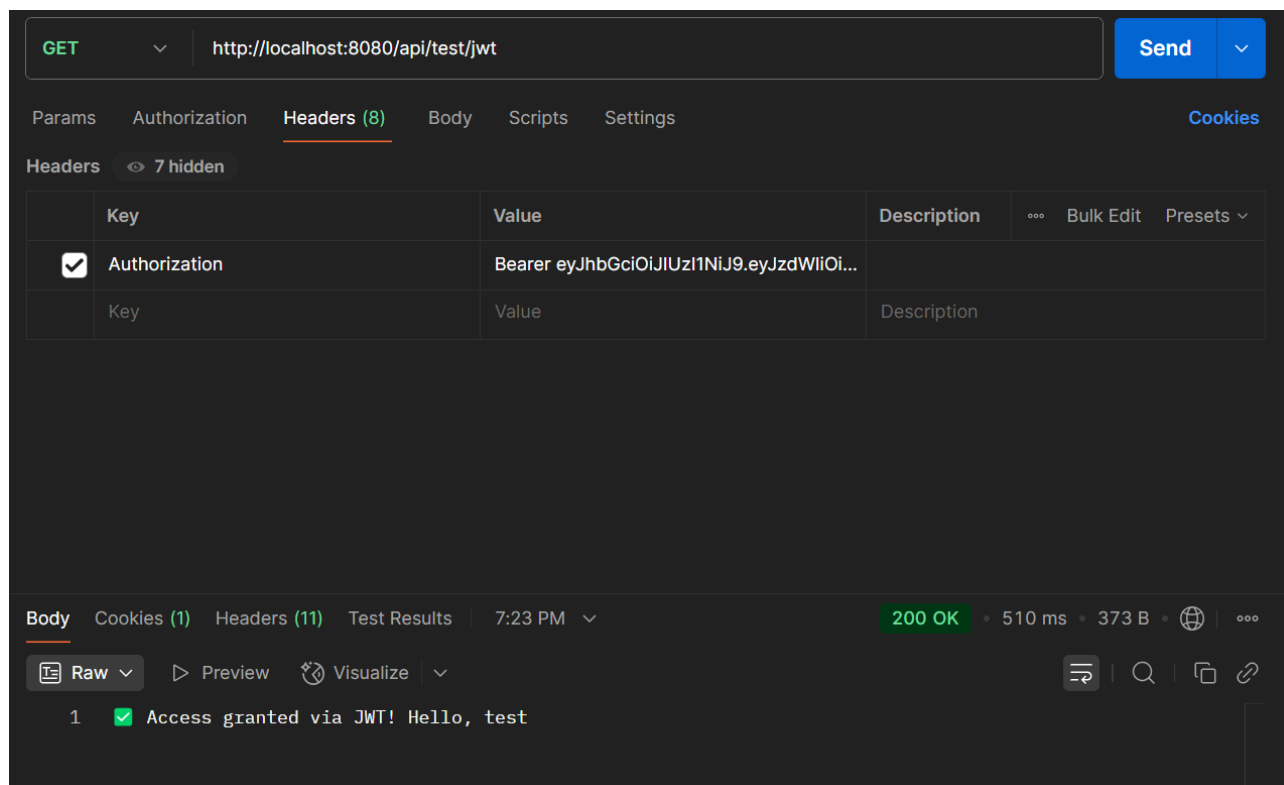


Рисунок 5.5 – Тестування jwt-токену

Висновки

В результаті виконання даної роботи було створено повнофункціональний RESTful API для медичної інформаційної системи, що забезпечує управління пацієнтами, лікарями, спеціальностями, записами на прийом та медичними картками. Особлива увага була приділена впровадженню механізмів безпеки, що базуються на технологіях JWT (JSON Web Token) та OAuth2, що дозволяє ефективно та надійно автентифікувати користувачів як через внутрішню систему, так і за допомогою сторонніх провайдерів (наприклад, Google).

Реалізовано кастомний фільтр для обробки JWT-токенів, який перевіряє кожен запит і встановлює контекст безпеки на основі валідного токена. Крім того, для OAuth2-логіки розроблено SuccessHandler, що створює або оновлює користувача в базі даних після успішної авторизації та генерує JWT для подальшої роботи з API. Ця архітектура забезпечує гнучкість, масштабованість і високу безпеку додатка.

Всі основні функції системи протестовані за допомогою ручного тестування та інструменту Postman, що дозволило впевнитися в коректній роботі ендпоінтів, правильній обробці помилок, а також підтвердити належний рівень захисту ресурсів. Під час тестування виявлені деякі помилки, зокрема некоректне оновлення ресурсів та відсутність обробки деяких винятків, які було усунуто шляхом додавання відповідних перевірок і виключень у сервісному шарі.

Під час реалізації проекту було здобуто цінний практичний досвід роботи з Spring Security, а також принципами побудови безпечних API, роботою з токенами та фільтрами, а також інтеграції зовнішніх OAuth2-провайдерів. Також було освоєно методи організації REST API з використанням анотацій Spring MVC.

Для подальшого розвитку проєкту доцільно розглянути впровадження ролей користувачів і контролю доступу на рівні методів (наприклад, через анотації `@PreAuthorize`), що дозволить точніше керувати правами доступу. Крім того, рекомендується розробити автоматизовані тести для API, які забезпечать стабільність роботи під час подальшого масштабування і змін. Окремо перспективним напрямком є створення клієнтського веб-інтерфейсу або мобільного застосунку, який використовуватиме розроблений API, що суттєво підвищить зручність користування системою.

Отже, поставлені цілі досягнуті, система працює стабільно і безпечно, а набуті навички створюють міцну основу для подальшого вдосконалення та розширення функціоналу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bauer C., King G. Java Persistence with Hibernate. 2nd Edition. Greenwich: Manning Publications, 2015. 608 p. (дата звернення: 11.06.2025).
2. Walls C. Spring in Action. 6th Edition. Shelter Island: Manning Publications, 2022. 656 p. (дата звернення: 11.06.2025).
3. Spring Framework Documentation. URL: <https://docs.spring.io/spring-framework/docs/current/reference/html/> (дата звернення: 11.06.2025).
4. Lombok Project Documentation. URL: <https://projectlombok.org/features/all> (дата звернення: 11.06.2025).
5. Richardson C. Microservices Patterns: With examples in Java. Shelter Island: Manning Publications, 2018. 520 p. (дата звернення: 11.06.2025).
6. Офіційна документація Hibernate ORM. URL: <https://hibernate.org/orm/documentation/> (дата звернення: 11.06.2025).