Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

Комплексна бакалаврська дипломна робота на тему:

«Автоматизована система видавання направлень на обстеження в медичних закладах. Частина 2. Модуль розмежування прав доступу користувачів»

Виконала: студентка 4 курсу групи 1БС-20б
спеціальності 125 Кібербезпека
_____ Вікторія КЛИШ

Керівник: к. т. н., доцент каф. ЗІ
_____ Юрій БАРИШЕВ
«14» червня 2024 р.

Рецензент: к. т. н., доцент каф. ПЗ
_____ Оксана РОМАНЮК
«14» червня 2024 р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«14» червня 2024 р.

Вінниця ВНТУ – 2024 року

Міністерство освіти і науки України
Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти I (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ,
д. т. н., проф.
Володимир ЛУЖЕЦЬКИЙ
«11» березня 2024 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТЦІ

Вікторії КЛИШ

1. Тема роботи: «Автоматизована система видавання направлень на обстеження в медичних закладах. Частина 2. Модуль розмежування прав доступу користувачів»
   керівник роботи: Юрій БАРИШЕВ, к. т. н., доцент кафедри ЗІ,
   затверджені наказом ректора ВНТУ від 11 березня 2024 року №80.
2. Строк подання студентом роботи 14 червня 2024 р.
3. Вихідні дані до роботи:
   - Підтримка браузерів: Firefox, Chrome, TOR.
   - Вид бази даних: Реляційна база даних.
   - Мова програмування: JavaScript.
   - Вид автентифікації користувачів: двофакторна.
4. Зміст текстової частини: Вступ. 1. Аналіз методів розмежування прав доступу. 2. Структура модуля розмежування прав доступу. 3. Алгоритм роботи модуля розмежування прав доступу. 4. Тестування автоматизованої системи. Висновки. Список використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: Порівняльний аналіз моделей розмежування прав доступу (плакат А4), узагальнена архітектура засобу (плакат А4), модель бази даних (плакат А4), структура токена (плакат А4), алгоритм роботи модуля (плакат А4), алгоритм створення токена (плакат А4), результати блокового тестування (плакат А4), результати статичного тестування безпеки (плакат А4).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Ю. БАРИШЕВ, к.т.н., доц. каф.ЗІ	11.03.24	30.05.24
2	Ю. БАРИШЕВ, к.т.н., доц. каф.ЗІ	11.03.24	14.04.24
3	Ю. БАРИШЕВ, к.т.н., доц. каф.ЗІ	11.03.24	28.04.24
4	Ю. БАРИШЕВ, к.т.н., доц. каф.ЗІ	11.03.24	07.06.24

7. Дата видачі завдання 11 березня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітк
1	Аналіз завдання. Вступ	12.03.24 – 14.03.24	
2	Аналіз інформаційних джерел за напрямком бакалаврської дипломної роботи	15.03.24 – 31.03.24	
3	Розробка рішень, моделей, алгоритмів	01.04.24 – 15.04.24	
4	Практична реалізація, моделювання, експериментування, результати	16.04.24 – 01.05.24	
5	Оформлення пояснювальної записки	02.05.24 – 10.05.24	
6	Попередній захист БДР	30.05.24 – 07.06.24	
7	Виправлення зауважень, підготовка ілюстративного матеріалу	10.06.24 – 12.06.24	
8	Перевірка на наявність текстових запозичень	12.06.24 – 13.06.24	
9	Представлення БДР до захисту, рецензування	13.06.24 – 17.06.24	
10	Захист БДР	18.06.24 – 21.06.24	

Студентка _____ Вікторія КЛИ

Керівник роботи _____ Юрій БАРИШ

АНОТАЦІЯ

Комплексна бакалаврська дипломна робота складається з 57 сторінок формату А4, на яких є 23 рисунки, 3 таблиці, список використаних джерел містить 35 найменувань.

Комплексна бакалаврська дипломна робота присвячена розробці автоматизованої системи видавання направлень на обстеження в медичних закладах, а саме розробці модуля розмежування прав доступу. Завдяки аналізу моделей розмежування прав доступу, було з’ясовано, що жодна модель самостійно не в змозі задовольнити високий рівень безпеки для медичних інформаційних систем. У зв'язку з цим було запропоновано комбінований підхід, що поєднує різні моделі розмежування прав доступу, який забезпечує підвищений рівень захисту даних у системі. Запропонована система використовує двофакторну автентифікацію, що дало змогу реалізувати комбінований підхід. Наведено алгоритми, що реалізують основні елементи системи. Виконано тестування, яке дозволило підтвердити безпеку та коректність роботи.

Ключові слова: кібербезпека, захист даних, персональні дані, база даних, автоматизована система, чутливі дані, медицина, розмежування прав доступу, RBAC, ABAC, токени, JWT, двофакторна автентифікація, клієнт-серверна архітектура.

ABSTRACT

A comprehensive bachelor's work consists of 57 A4 pages, containing 23 figures, 3 tables, and a list of 35 references.

The comprehensive bachelor's thesis is devoted to the development of an automated system for issuing electronic referrals for examinations in medical institutions, specifically the development of user access control module. Through the analysis of access control models, it was found that no single model is able to meet the high level of security for medical information systems. In this regard, a combined approach that utilizes different access control models was proposed, which provides an increased level of data protection in the system The proposed system performs two-factor authentication, which allowed to implemente the combined approach. The algorithms those implement the main elements of the system are presented. Testing has been carried out to confirm the security and correction of the system.

Keywords: cybersecurity, data protection, personal data, database, automated system, sensitive data, medicine, access control, RBAC, ABAC, tokens, JWT, two-factor authentication, client-server architecture.

ЗМІСТ

ВСТУП 4
1 АНАЛІЗ МЕТОДІВ ЗАБЕЗПЕЧЕННЯ КОНФІДЕНЦІЙНОСТІ ДАНИХ КОРИСТУВАЧІВ 6
1.1 Моделі розмежування прав доступу 6
1.2 Аналіз поширених методів автентифікації 13
1.3 Огляд JWT за стандартом RFC 7519 15
1.4 Постановка задачі 17
1.5 Висновки з розділу 18
2 СТРУКТУРА МОДУЛЯ РОЗМЕЖУВАННЯ ПРАВ ДОСТУПУ 19
2.1 Узагальнена архітектура засобу 19
2.2 Проєктування структури бази даних 21
2.3 Структура токена 25
2.4 Архітектура програмного коду 27
2.5 Висновки з розділу 28
3 АЛГОРИТМ РОБОТИ МОДУЛЯ РОЗМЕЖУВАННЯ ПРАВ ДОСТУПУ 30
3.1 Узагальнений алгоритм роботи засобу 30
3.2 Алгоритм роботи модуля клієнта 31
3.3 Алгоритм роботи модуля сервера 32
3.4 Алгоритм роботи модуля взаємодії з базою даних 34
3.5 Алгоритм створення токена 35
3.6 Алгоритм автентифікації 36
3.7 Висновки з розділу 38
4 ТЕСТУВАННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ 39
4.1 Обґрунтування виборів засобів розробки 39
4.2 Блокове тестування 42
4.3 Інтеграційне тестування 43
4.4 Статичне тестування безпеки 49
4.5 Висновки з розділу 50
ВИСНОВКИ 52

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 54
ДОДАТКИ 57
Додаток А. Текст застосунку. Модуль клієнта 58
Додаток Б. Текст застосунку. Модуль сервера 65
Додаток В. Результати інтеграційного тестування 71
Додаток Г. Протокол перевірки бакалаврської дипломної роботи на наявність текстових запозичень 80
Додаток Д. Ілюстративна частина 81

ВСТУП

Медичний заклад – це установа, де обробляється великий обсяг конфіденційної інформації про пацієнтів. Ця інформація містить особисті та медичні дані, такі як діагнози, результати тестів, лікування та інші приватні відомості. З огляду на важливість захисту цієї інформації від несанкціонованого доступу, використання розмежування прав доступу є важливим у медичній галузі.

Тому постає актуальне завдання забезпечити високим рівнем захисту чутливу інформацію в медичних закладах. Один із ключових аспектів цього – розмежування прав доступу до системи. Це включає реєстрацію користувачів з наданням їм відповідних ролей і атрибутів, що дозволяють точно визначити їхні права доступу та обмеження.

Водночас в медичних закладах потрібно реалізовувати принцип мінімуму привілеїв. Це дозволить запобігти надмірному доступу до конфіденційної інформації та підвищеному ризику її втрати чи пошкодження. Відомі методи захисту можуть стати застарілими в умовах цифрового середовища. Нестача гнучкості у налаштуванні та адаптації може обмежувати здатність системи управляти доступом користувачів.

Враховуючи недоліки сучасних медичних інформаційних систем, зміни у підходах до розмежування прав доступу потребують суттєвої зміни всієї системи. Для досягнення цього система має бути одночасно жорсткою у мінімізації привілеїв для користувачів та гнучкою, щоб адаптуватися до нових вимог і умов. Тому в медичних інформаційних системах актуально реєструвати користувачів і надавати їм визначені ролі та атрибути. Це дозволить здійснити контроль доступу до системи за допомогою атрибутів робочої станції, а також встановлювати обмеження за часом. При розробці атрибутів важливо врахувати необхідність створення токена, який забезпечить функціональність з використанням цих атрибутів.

Об’єктом роботи є процес розмежування прав доступу працівників в медичних системах.

Предметом дослідження є методи та засоби розмежування прав доступу при видаванні направлень.

Мета роботи – покращення автентифікації користувачів, задіяних в медичних системах в процесі видавання направлень. Основні завдання, які необхідно виконати в рамках цієї роботи, включають:

- впровадження двофакторної автентифікації;
- створення бази даних для зберігання інформації про користувачів;
- створення ролей для користувачів;
- генерування токенів;
- розробка додаткових атрибутів часу та робочої станції для токена;
- розробка інтерфейсу програмного засобу.

Результати роботи були апробовані на 3 конференціях з публікацією тез та матеріалів доповідей:

- аналіз можливостей некласичних моделей розмежування прав доступу для захисту медичних даних – LIII Всеукраїнська науково-технічна конференція підрозділів ВНТУ (2024) [1];
- метод захищеного зберігання медичних даних за допомогою розмежування прав доступу та блокчейну – The 13th International Scientific Conference «ITSec» (2024) [2];
- модуль видавання електронних направлень в медичних закладах – Theoretical and Applied Cybersecurity «TACS-2024» (2024) [3].

1 АНАЛІЗ МЕТОДІВ ЗАБЕЗПЕЧЕННЯ КОНФІДЕНЦІЙНОСТІ ДАНИХ КОРИСТУВАЧІВ

1.1 Моделі розмежування прав доступу

Розмежування прав доступу [4] (англ. access control) – це механізм контролю доступу, що визначає, які користувачі мають доступ до яких ресурсів у системі. Це означає, що тільки авторизовані користувачі, такі як лікарі, медичні сестри та адміністратори, можуть отримувати доступ до конфіденційних даних про пацієнтів, у той час як інші мають обмежений доступ або жодного доступу до цієї інформації.

Для отримання доступу до конфіденційної інформації, користувач повинен пройти процедуру ідентифікації в системі, що включає в себе проходження автентифікації. Цей процес передбачає введення облікових даних, таких як пароль, PIN-код або токен, а також може включати біометричне сканування. Після успішної автентифікації система відповідно до встановлених політик контролю доступу користувачеві надаються відповідні дозволи для використання системи з урахуванням його прав та обов'язків.

Існує чотири основні типи розмежування правд доступу, кожен з яких керує доступом до конфіденційної інформації унікальним способом.

Дискреційне розмежування прав доступу (DAC) [5]. Механізм дискреційного розмежування прав передбачає, що власники даних можуть визначати права доступу для конкретних користувачів або груп користувачів**.** Отже, користувач, обладнаний певними привілеями до об'єкта, має можливість делегувати ці самі привілеї іншому користувачеві, і, відповідно, останній може передати ці ж привілеї далі.

Дозволи на доступ до кожної частини даних зберігаються в списку контролю доступу (access control list). ACL включає користувачів і групи, які можуть отримати доступ до даних, а також рівні доступу, які вони мають. Таким чином ACL діє як політика безпеки, і звичайні користувачі не можуть його редагувати або видалити.

DAC легкий у користування та зручний, однак, оскільки облікові дані можуть вільно передаватися між співробітниками, модель DAC, як відомо, несе певні ризики для безпеки, тому не є придатною для організацій, які вимагають високого рівня безпеки своїх даних, наприклад, медицина, фінанси, військові, державні установи тощо.

Переваги DAC:

– дружній до користувачів;
– користувач легко може делегувати права доступу до власних або чужих ресурсів іншим користувачам.

Недоліки:

– важко відслідковувати хто кому надає привілеї, тому може бути витік інформації до когось за межами організації;
– не централізована система тому адміністрації важко відслідковувати дії користувачів, єдиний спосіб ACL.

До моделі дискреційного розмежування прав доступу належать такі моделі [5]:

– Access Control Matrix;
– Модель Гаррісона-Руззо-Ульмана (HRU);
– Take-Grant.

Access Control Matrix є загальною моделлю, яка визначає права доступу між об'єктами та суб'єктами у вигляді таблиці, яка містить детальний перелік прав доступу та дій, які можуть виконувати користувачі.

Модель Гаррісона-Руззо-Ульмана (HRU) – модель безпеки стану захисту в комп'ютерних системах, яку можна представити у вигляді матриці, з одним рядком для кожного суб'єкта і одним стовпчиком для кожного суб'єкта і об'єкта (всі суб'єкти також є об'єктами). У таблиці наведено перелік прав доступу, об'єктів та суб'єктів, за яким можна легко визначити, які конкретні права доступу мають суб'єкти до об'єктів. Таким чином, HRU можна вважати розширеною версією або вдосконаленою версією Access Control Matrix, яка більш гнучко визначає права доступу та управляє безпекою в системах комп'ютерної безпеки.

Модель Take-Grant представляє всю систему як орієнтований граф, вершинами якого є суб'єкти та об'єкти комп'ютерної системи, а дугами – набір прав доступу суб'єктів до об'єктів. Ця модель вводить два нових права – «брати» (take) і «дарувати» (grant). Право «брати» копіює будь-яке право, що належить об'єкту B, на об'єкт A, тоді як право «дарувати» копіює право, що належить об'єкту A, на об'єкт B. Використовується коли потрібно відстежити багато взаємозв'язків між суб'єктами та об'єктами в системі.

Мандатне розмежування прав доступу (MAC) [6] надає користувачам доступ спираючись на рівень конфіденційності даних та рівень доступу користувача. Користувачам потрібно довести потребу в інформації перед тим, як отримати доступ.

Кожному користувачеві призначається рівень безпеки або допуску. Отримати доступ до об’єкта чи ресурсу можна лише якщо їхній рівень допуску дорівнює або перевищує рівень класифікації ресурсу. Коли особа намагається отримати доступ до конкретного ресурсу, операційна система або ядро безпеки перевіряють облікові дані об'єкта, щоб визначити, чи буде надано доступ.

Адміністратор відіграє ключову роль у встановленні та підтримці MAC, налаштовуючи всі дозволи користувача та контролюючи доступ до ресурсів. Через таке жорстке адміністрування звичайні користувачі не можуть налаштувати власні дозволи.

Переваги мандатного розмежування прав доступу:

– захист від атак – користувачі не мають можливості розкрити або передати конфіденційні дані;
– високий рівень захисту даних – адміністратор визначає доступ до об'єктів, і користувачі не можуть змінювати ці налаштування.

Недоліки:

– користувачі повинні постійно запитувати доступ до ресурсу;
– не масштабований – дана модель потребує постійного оновлення.

До моделі мандатного розмежування прав доступу належать такі моделі [6]:

– Белл-ЛаПадули (Bell-LaPadula;

– Біба (Biba).

Модель Белла-ЛаПадули розроблена для забезпечення суворого захисту конфіденційності критично важливої інформації та використовується для підтримки конфіденційності.

Принципи моделі Белл-ЛаПадули:

– «The Simple Security» – суб'єкт може читати інформацію з об'єкта тоді і тільки тоді, коли його рівень доступу дорівнює або перевищує рівень секретності об’єкта;
– «Star Property» – суб'єкт може писати інформацію у об’єкт на тому самому та верхньому рівні секретності, але не на нижньому рівні секретності;
– «Strong Star Property» – суб'єкт може читати та писати інформацію у об’єкт лише на тому самому рівні секретності.

Модель Біба використовується для підтримки цілісності безпеки та працює прямо протилежно до моделі Белла-ЛаПадули.

Принципи моделі Біба:

– «Simple integrity rule» – суб'єкт може лише читати файли на тому самому рівні секретності та верхньому рівні секретності, але не на нижньому рівні секретності;
– «Star integrity rule» – суб'єкт може лише записувати файли на тому самому рівні секретності та нижньому рівні секретності, але не на верхньому рівні секретності;
– «Strong Star integrity rule» – суб'єкт може читати та писати інформацію у об’єкт лише на тому самому рівні секретності.

Рольове розмежування прав доступу (RBAC) [7] забезпечує управління доступом на основі посади (ролі), яку користувач займає. У цій системі, ролі вже мають визначені права доступу, і користувачам призначаються відповідні ролі відповідно до їхніх обов'язків або функцій у організації.

З великою кількістю співробітників безпеку легше підтримувати обмеживши непотрібний доступ до конфіденційної інформації на основі встановленої ролі кожного користувача в організації.

Переваги:

- користувачам не потрібно постійно запитувати адміністраторів на доступ до об’єкту, тому підвищиться ефективність роботи;
- обмеження доступу користувачів лише до необхідних об’єктів, зменшує ризик порушення даних і витоку інформації, та обмежує вплив зловмисних атак.

Недоліки:

- впровадження RBAC може бути складною та громіздкою роботою через потребу у чіткому визначенні ролей, взаємодії між відділами та продуманій реалізації;
- потреби організації постійно змінюються, але дана модель не може підлаштовуватися, тому що їй бракує гнучкості;
- має обмеження через брак гнучкості. Ролі, визначені на початку, можуть більше не відповідати цілям організації, особливо у контексті зміни команд та росту компанії;
- створення детальних і спеціальних ролей лише зробить систему заплутаною та складною для управління.

Розмежування прав доступу на основі атрибутів (ABAC) [8]. Це модель, в якій запити суб'єкта на виконання операцій над об'єктами задовольняються або відхиляються на основі призначених атрибутів суб'єкта, умов середовища та набору політик.

Атрибути спрощують структуру управління, оскільки дозволи можуть залежати від типу користувача, місця розташування, відділу тощо. У такій системі адміністратори мають багато тонкого контролю. Також можна навіть надати одній особі різні дозволи залежно від місця входу або того, що вона намагається зробити в інший день тижня.

В моделі ABAC існують два типи атрибутів: на основі дій та на основі середовища. Атрибути на основі дій визначають доступ користувачів до ресурсів на основі їхніх дій, таких як читання, запис, видалення і т. д. Атрибути на основі середовища враховують такі умови : час доби, місцезнаходження, стан системи і т. д., щоб визначити, чи має користувач доступ до ресурсів у певних умовах.

Переваги:

- гнучкість – адміністратори можуть визначати, покращувати та керувати багатьма змінними, забезпечуючи високий рівень контролю;
- інкапсуляція – використовує інкапсуляцію для приховування технічних деталей прав доступу. Рішення про доступ змінюються лише зміною значень атрибутів зберігаючи при цьому основний зв’язок суб’єкт/об’єкт;
- ускладнює користувачам доступ до ресурсів і важливих служб на невідомих пристроях.

Недоліки:

- багато змінних. Визначення змінних і налаштування правил – це величезна робота, особливо на початку проєкту;
- через велику кількість атрибутів потребує значних ресурсів, включаючи більше обчислювальної потужності та часу.

Модель розмежування прав на основі організації (OrBAC) [9] є варіантом моделі RBAC, в якій сутності, які становлять організацію, є одним із складових моделювання. Кожен користувач має одну або декілька ролей залежно від того, до яких груп належить.

OrBAC визначає структуру організації через активні сутності (entities) та їх ролі, а також використовує такі поняття:

- активність (Activity) – група одного або кількох дій, які виконуються;
- вид (View) – група одного або кількох об'єктів;
- контекст (Context) – конкретна ситуація, яка обумовлює дійсність правила.

Однією з головних переваг OrBAC є те, що вона може автоматично отримувати конкретні правила з абстрактних правил. Також підтримує концепцію

успадкування, яка забезпечує дуже ефективний спосіб структурування політики безпеки, та концепцію делегування, яка дозволяє користувачам самостійно керувати деякими своїми правами доступу та надавати їх іншим користувачам за потреби. Це розподіляє відповідальності та зменшує навантаження на адміністраторів.

Розглянуті моделі доступу представляють різні підходи до управління доступом в інформаційних системах. У таблиці 1.1 наведено основні характеристики кожної моделі, які можна використовувати для аналізу та вибору найбільш адекватної моделі в залежності від вимог конкретного проєкту або системи. Конфіденційність вказує на рівень захисту конфіденційної інформації від несанкціонованого доступу. Цілісність проти випадкового внесення інформації вказує на здатність системи захищати дані від непередбачуваних змін. Протидія проти навмисного внесення дезінформації означає захист від умисних спроб зміни або спотворення інформації. Гнучкість вказує на здатність системи адаптуватися до змін у вимогах до доступу і ресурсів. Оцінка кожної моделі відображається через високий, середній або низький рівень для кожного з цих критеріїв.

Таблиця 1.1 – Порівняння моделей розмежування прав доступу

Вимоги	HRU	Bell-LaPadula	Biba	RBAC	ABAC	OrBAC
Конфіденційність	Висока	Висока	Низька	Середня	Висока	Середня
Цілісність	Середня	Середня	Висока	Середня	Середня	Середня
Гнучкість	Середня	Низька	Низька	Висока	Висока	Висока
Протидія внесенню дезінформації	Середня	Висока	Середня	Висока	Висока	Середня

Згідно з результатами порівняння, кожна з моделей має свої переваги та обмеження. Враховуючи зазначені обмеження, важливо обрати рішення для забезпечення високого рівня захисту чутливих даних, а також для забезпечення швидкості та зручності управління правами доступу. Тому варто розглянути комбіновані моделі, які поєднують переваги різних підходів та уникають недоліків окремих моделей.

1.2 Аналіз поширених методів автентифікації

Автентифікація – це процес перевірки ідентичності користувача перед наданням доступу до системи або послуг. Вибір методу автентифікації важливий не лише для захисту даних користувачів, але і для забезпечення їхнього комфорту та впевненості в безпеці при використанні онлайн-сервісів. Метод автентифікації повинен враховувати різні фактори, включаючи рівень конфіденційності даних, зручність для користувачів, ефективність та можливість інтеграції з іншими системами [10].

Поширені такі методи автентифікації [11]:

- традиційні логін і пароль. Цей метод використовується широко і є стандартним для багатьох систем. Він включає в себе введення ідентифікаційних даних, таких як ім'я користувача та пароль, для підтвердження ідентичності користувача;
- біометричні дані. Використовує біометричні дані, такі як відбитки пальців, голос, сітківка ока та риси обличчя, для ідентифікації користувача;
- автентифікація на основі одноразових паролів (OTP). Використовує одноразові паролі, які генеруються і надсилаються користувачам через SMS, електронну пошту або спеціальні програми для автентифікації;
- автентифікація на основі токенів. Цей підхід використовує токени, що надаються користувачам, як підтвердження їхньої ідентичності;
- апаратна автентифікація базується на використанні спеціального фізичного пристрою, такого як USB-ключ або смарт-карта, який належить авторизованому користувачеві.

Традиційні логін і пароль вважається не дуже стійким способом захисту даних. Навіть «сильні паролі» марні проти таких атак, як фішинг і клавіатурні шпигуни або перебір паролів. Такі атаки не намагаються вгадати пароль, як атака грубої сили, а викрадають його безпосередньо у користувача.

Біометричні дані користувача є унікальними, що робить їх надзвичайно складним для зловмисників дублювати, оскільки сайт даного засобу буде в робочих комп’ютерах – реалізувати даний метод автентифікації буде непросто і не вигідно.

Апаратна автентифікація надає високий рівень безпеки та зручності використання. Проте, серед недоліків варто зазначити ризик крадіжки або втрати пристрою а також вартість таких пристроїв.

Одноразові паролі зручні і легкі у використанні, але поступаються перед токенами тим, що дають всі права до системи. Якщо зловмиснику вдасться отримати доступ до одноразового пароля, він матиме повний доступ до системи, і це може створити серйозний ризик для безпеки.

У порівнянні з цим, токени, мають більшу гнучкість та можливість регулювати рівні доступу. Серед найбільш популярних токенів виділяються такі типи: Opaque, Phantom, Split, Tokens Handler та JSON Web Tokens (JWT).

Opaque токени самі по собі не містять жодної інформації про користувача або права доступу. Вони зазвичай є випадковими рядками, які використовуються для отримання даних з сервера автентифікації. Проте залежні від сервера для перевірки кожного токена та зумовлюють високе навантаження на сервери автентифікації при масштабуванні. Phantom токени поєднують в собі JWT та Opaque токени. Коли користувач надсилає phantom token, сервер перетворює його на реальний токен з корисною інформацією. Проте, складно реалізувати та управляти. Split токени розділяються на дві частини, одна з яких зберігається на клієнті, а інша - на сервері, що підвищує безпеку, але ускладнює управління і збільшує затримки через необхідність з'єднання з сервером. Tokens Handler є централізованою службою, яка керує створенням, зберіганням та перевіркою токенів, проте проблеми з сервером можуть вплинути на всю систему. JSON Web Tokens (JWT) є відкритим стандартом, який визначає компактний спосіб передачі інформації між сторонами у вигляді JSON-об'єкта. За допомогою JWT, можна встановити різні права доступу для різних користувачів, і це дозволяє строго контролювати їхні дії в системі. Також, токени можуть бути встановлені на певний термін дії, після чого вони автоматично втрачають чинність, зменшуючи ризик несанкціонованого доступу.

Порівняння методів автентифікації важливо для визначення оптимальних засобів захисту доступу до системи. Таблиця 1.2 допоможе прийняти обґрунтоване рішення щодо вибору методу автентифікації, який найкраще відповідає потребам конкретної системи чи проєкту.

Таблиця 1.2 – Порівняння методів автентифікації

Методи автентифікації	Вартість реалізації	Складність реалізації	Вразливість до атак	Ймовірність втрати	Потреба в додатковому обладнанні
Логін і пароль	Низька	Низька	Висока	Висока	Ні
Біометричні дані	Висока	Висока	Середня	Низька	Так
Одноразові паролі (OTP)	Середня	Середня	Середня	Відсутня	Ні
Токени	Низька	Середня	Низька	Низька	Ні
Апаратний метод	Висока	Висока	Середня	Висока	Так

Порівняльний аналіз методів автентифікації допомагає зрозуміти переваги та недоліки кожного методу. Обираючи метод автентифікації для конкретного проєкту, важливо враховувати рівень захисту, зручність використання та вартість реалізації.

1.3 Огляд JWT за стандартом RFC 7519

У контексті даної досліджуваної системи обрано використання JSON Web Tokens (JWT) як методу автентифікації. JWT – метод безпечної передачі інформації між сторонами як об’єкт JSON [12].

JWT складається з трьох частин: заголовка (Header), тіла (Payload) і підпису (Signature).

Заголовок описує криптографічні операції, застосовані до JWT, і, за бажанням, додаткові властивості JWT:

– параметр «typ» (type) використовується для оголошення типу JWT. Рекомендується писати цей параметр великими літерами. Використання «typ» є необов'язковим та зазвичай ігнорується реалізаціями JWT;

– параметр заголовка "cty" (content type) використовується для передачі структурної інформації про JWT. Необов’язковий коли не використовується вкладений підпис або шифрування. Рекомендується завжди писати "JWT" за допомогою великих літер для сумісності зі старішими реалізаціями.

Тіло містить параметри (атрибути користувача) і додаткові дані. Існує набір зареєстрованих параметрів:

– параметр "iss" (Issuer) ідентифікує суб'єкта, який видав JWT. Значення "iss" є чутливою до регістру рядком, що містить значення StringOrURI. Використання цього параметра є необов'язковим;
– параметр "sub" (Subject) ідентифікує суб'єкта, який є темою JWT (для чого призначається). Чутливий до регістру;
– параметр "aud" (Audience) ідентифікує одержувачів, для яких призначений JWT. Чутливий до регістру;
– параметр "exp" (Expiration Time) вказує час закінчення терміну дії після якого JWT більше не приймається для обробки. Значення повинно бути числом, що містить значення NumericDate;
– параметр "nbf" (Not Before) вказує на термін до якого токен не дійсний. Значення повинно бути числом, що містить значення NumericDate;
– параметр "iat" (Issued At) вказує час, коли був виданий JWT. Значення повинно бути числом, що містить значення NumericDate;
– параметр "jti" (JWT ID) надає унікальний ідентифікатор для JWT надає унікальний ідентифікатор для JWT. Його можна використовувати для запобігання повторному використанню токена. Чутливий до регістру;

Ці параметри не є обов'язковими, але їхнє використання не за призначенням може призвести до колізій.

Підпис грає важливу роль у забезпеченні цілісності і автентичності токена. Алгоритм, який використовується для створення підпису, є тим самим алгоритмом,

згаданим у розділі заголовка JWT. Заголовок і тіло JWT завжди кодуються Base64 [13].

Основні переваги JWT включають простоту використання, швидкість та здатність підтримувати розмежування прав доступу для користувачів. Також важливою перевагою є той факт, що JWT може використовуватися в різних середовищах, включаючи мобільні застосунки та веб-сайти.

1.4 Постановка задачі

Під час аналізу відомих рішень у сфері розмежування прав доступу для медичних установ, було виявлено декілька недоліків, які становлять серйозні проблеми для безпеки чутливих даних.

Незважаючи на те, що модель RBAC вже активно використовується в медичних інформаційних системах, вона має певні обмеження. Перший недолік – складність управління ролями. У великих медичних установах кількість ролей може бути дуже великою, що ускладнює їхнє управління та налаштування. Другий недолік – недостатня гнучкість моделей. Статичні ролі у RBAC часто не враховують всі можливі контексти доступу до даних, такі як час, місце або конкретні умови, що може призвести до надмірного або недостатнього доступу.

Усі ці недоліки підкреслюють необхідність розробки нових підходів та моделей для строгого розмежування прав доступу в медичних установах.

Автентифікація є важливою складовою забезпечення безпеки в медичних інформаційних системах. Існуючі методи автентифікації, такі як логін та пароль, мають свої обмеження, оскільки вони можуть бути вразливими до атак типу «фішинг» та «перебір». Для підвищення рівня безпеки необхідно впровадити двофакторну автентифікацію, яка поєднує два різних методи ідентифікації користувача: пароль та токен.

Головною задачею є розробка модуля розмежування прав доступу, який усуне недоліки, виявлені під час аналізу відомих рішень для медичних установ. Розроблений модуль повинен забезпечити високий захист чутливих даних і

відповідати сучасним вимогам безпеки, забезпечуючи тим самим безпечну та швидку роботу медичних установ.

1.5 Висновки з розділу

Розглянуто класичні моделі розмежування прав доступу, їх переваги та недоліки та принципи за якими вони працюють.

Аналіз моделей розмежування прав доступу показав, що для застосування в автоматизованій системі видавання направлень найбільш доцільне комбіноване поєднання моделей ABAC та RBAC. RBAC управляє правами доступу в залежності від обов'язків користувачів, а ABAC впроваджує такі додаткові атрибути, як робоча станція та час, коли можна отримати доступ.

Аналіз факторів автентифікації дозволив обґрунтувати використання паролю та JWT, який відповідає стандарту RFC 7519. JWT підтримує вбудовану можливість встановлення терміну дії токенів, що знижує ризик несанкціонованого доступу після закінчення їхнього терміну дії. Також дасть можливість впровадити робочу станцію та час коли можна здійснювати вхід в систему. Це робить JWT більш безпечним та зручним у використанні у порівнянні з іншими типами токенів.

Завдяки комплексному підходу, що поєднує багатофакторну автентифікацію та розмежування прав доступу за допомогою RBAC та ABAC, система може досягти високого рівня безпеки без впровадження додаткового обладнання. Це захистить чутливі медичні дані пацієнтів, дозволяючи доступ до них лише авторизованим користувачам з відповідними правами доступу.

2 СТРУКТУРА МОДУЛЯ РОЗМЕЖУВАННЯ ПРАВ ДОСТУПУ

2.1 Узагальнена архітектура засобу

Архітектура програмного забезпечення – структура системи, яка включає програмні елементи та їх зв’язки. Основними структурними елементами є база даних (БД), сервер та клієнт.

Користувачі будуть знаходитися на різних пристроях, тому для програмного засобу необхідно застосувати клієнт-серверний підхід [14] (рис. 2.1). Це дозволяє забезпечити доступність системи з будь-якого місця та будь-якого пристрою.

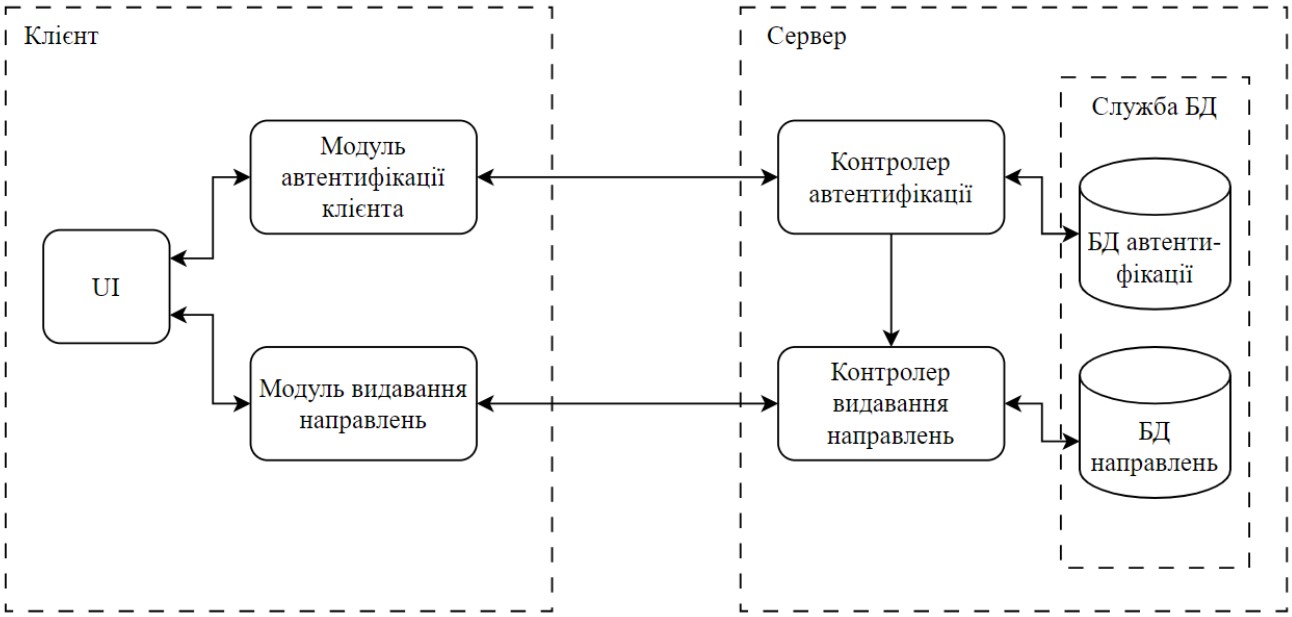


Рисунок 2.1 – Узагальнена архітектура засобу

По-перше, розподіл функціональності між клієнтом і сервером дозволяє розподілити навантаження на систему, що значно підвищує її продуктивність та здатність обробляти більшу кількість запитів.

По-друге, завдяки структурному розділенню компонентів досягається високий рівень безпеки. Навіть якщо клієнтська частина буде скомпрометована зловмисниками, сервер із чутливими даними та логікою залишається захищеним від несанкціонованого доступу.

По-третє, основна логіка і дані зосереджені на сервері, тому їх набагато легше підтримувати, змінювати та оновлювати з одного місця.

В клієнті міститься модуль для автентифікації клієнта, модуль видавання направлень та користувацький інтерфейс для взаємодії з серверною архітектурою. На сервері обробляють запити до двох баз даних. База даних автентифікації пов’язана з модулем автентифікації, а база даних з направленнями – з модулем видавання направлень.

Розробка політики розмежування прав доступу на сервері забезпечить захист даних і дозволить управляти доступом до системи на основі ролей користувачів. Серверна сторона має бути з’єднана з базою даних, оскільки у базі даних зберігаються всі дані, необхідні для функціонування системи.

База даних централізовано управляє всіма даними і забезпечує високу швидкість доступу до них. База даних забезпечить захист цілісності, доступності та структурованості даних. У ній будуть зберігатися користувачі, їх ролі та токени.

На сервері обробляються запити від клієнтів і він взаємодіє з базою даних. Сервер відповідає за автентифікацію та авторизацію користувачів, зберігання та обробку даних. Сервер реалізує політику розмежування прав доступу, забезпечуючи доступ для користувачів залежно від їх ролей. Сервер забезпечує цілісність даних і виконує всі необхідні перевірки перед збереженням або зміною даних у базі даних.

Клієнт надсилає запити на сервер і отримує відповіді, завдяки чому користувач взаємодіє з системою через зручний інтерфейс.

Схема на рисунку 2.2 зображує архітектуру клієнт-серверної взаємодії з базою даних. Клієнт (користувач) надсилає запити на сервер. Сервер обробляє ці запити та взаємодіє з базою даних для отримання або збереження необхідної інформації. Після обробки запитів сервер відправляє відповіді назад до клієнта. Таким чином, сервер виступає посередником між клієнтом і базою даних, забезпечуючи обмін даними.

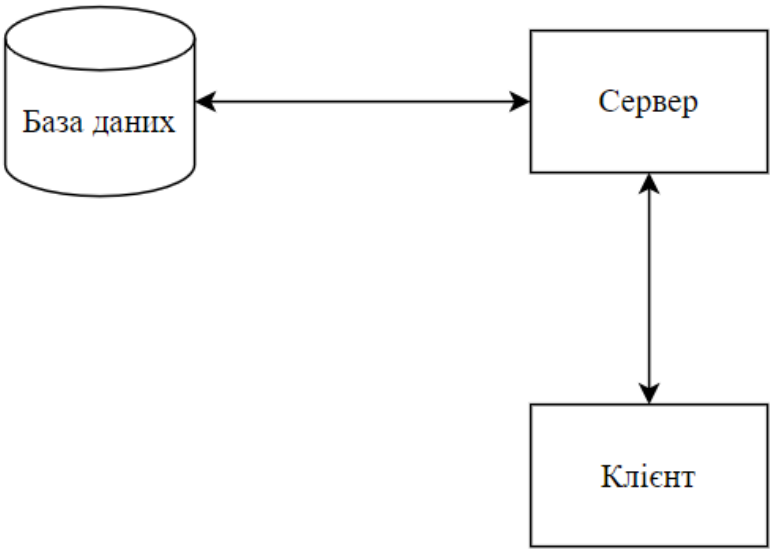


Рисунок 2.2 – Схема архітектури програми

Наступний етап – проєктування структури бази даних, яка дозволить розмежовувати права доступу.

2.2 Проєктування структури бази даних

В контексті автентифікації користувачів, реляційна база даних (БД) є важливою складовою для збереження та управління даними користувачів. Реляційна база даних заснована на реляційній моделі, яка в свою чергу представлена у вигляді таблиць. Кожна таблиця у БД має унікальний ключ, який ідентифікує окремі записи. Зв’язки між таблицями у БД встановлюються за допомогою зовнішніх ключів, що вказують на спільні дані між таблицями. Завдяки цьому система автентифікації перевіряє введені користувачем дані та забезпечує відповідність між введеними даними та записами у базі даних для надання або відмови у доступі.

Далі потрібно створити схему бази даних, яка необхідна для системи управління доступом. Це включає таблиці для користувачів, токенів і ролей, а також таблицю для встановлення зв'язку між користувачами та їх ролями.

Таблиця «Users» повинна містити дані про користувачів, такі як їхні логіни, паролі, ПІБ, організацію, посаду та номер ліцензії. Таблиця «Tokens» повинна

зберігати інформацію про видані токени, включаючи геш, робочу станцію та дату закінчення терміну дії токенів. Таблиця «Roles» повинна містити різні ролі, які можуть бути надані користувачам, разом з їхніми правами доступу. Нарешті, таблиця «UsersAndRoles» повинна встановлювати зв'язок між користувачами і їхніми ролями, надаючи користувачам відповідні права доступу.

Нижче наведена схема бази даних на рисунку 2.3:

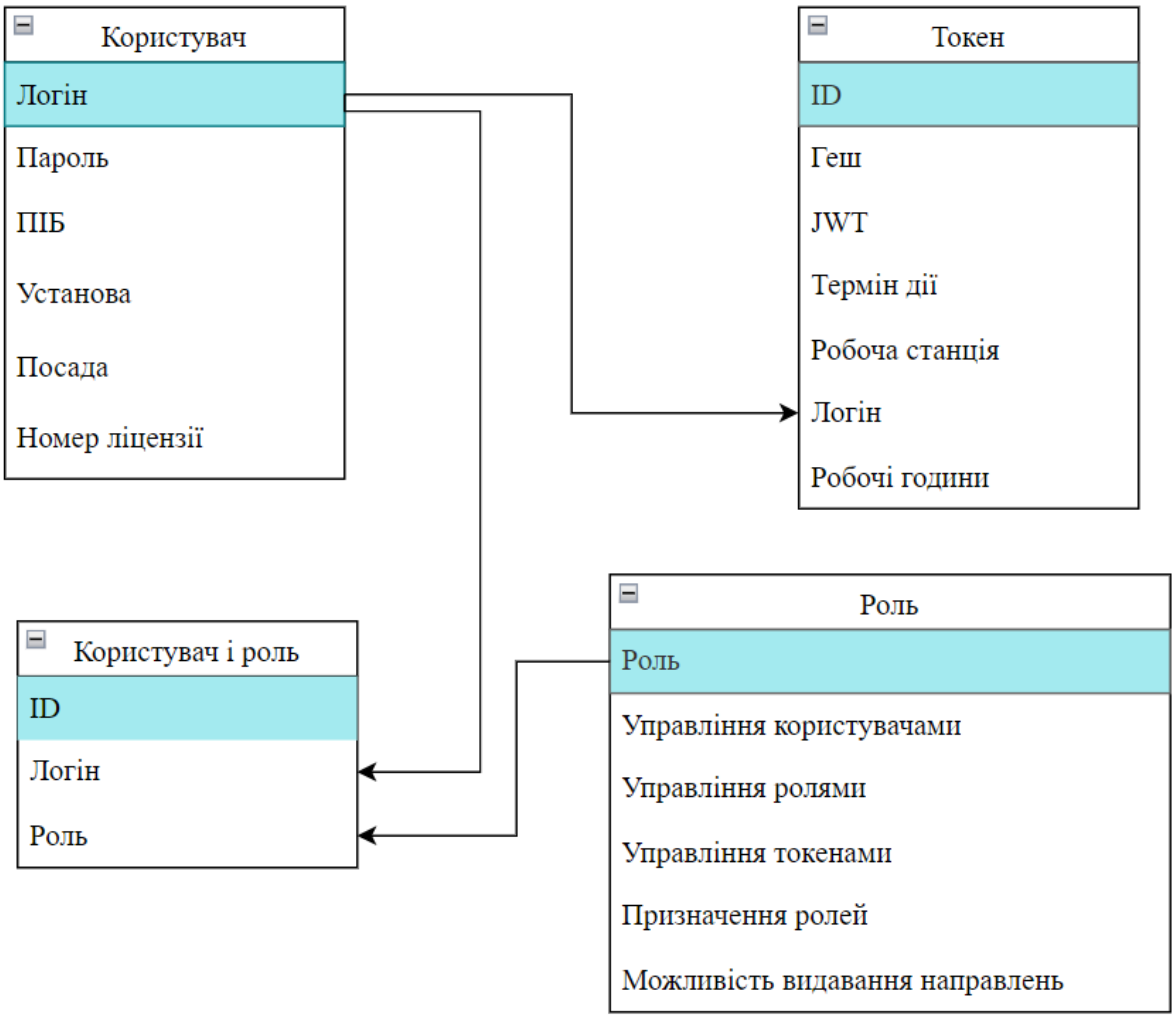


Рисунок 2.3 – Структура бази даних

Тепер необхідно проаналізувати в якій формі знаходиться база даних. Нормалізація – це процес організації полів і таблиць реляційної бази даних з метою мінімізації надлишкових даних та запобігання аномаліям при оновленні даних.

База даних знаходиться в першій нормальній формі (1NF) [15], якщо всі її атрибути атомарні, тобто кожне поле містить тільки одне значення, і кожне

значення в таблиці є унікальним. Всі таблиці в цій базі даних відповідають вимогам 1NF:

– кожне поле містить тільки одне значення;

– кожний рядок є унікальним завдяки первинним ключам.

База даних знаходиться у другій нормальній формі (2NF) [14], якщо вона відповідає вимогам 1NF і всі неключові атрибути повністю залежні від первинного ключа. Всі таблиці в базі даних відповідають вимогам 2NF, оскільки відповідають вимогам 1NF та атрибути кожної таблиці залежать від первинного ключа відповідної таблиці.

База даних знаходиться в третій нормальній формі (3NF) [14], якщо вона відповідає вимогам 2NF і жоден неключовий атрибут не залежить транзитивно від первинного ключа, тобто неключові атрибути не повинні залежати один від одного. Всі таблиці в базі даних відповідають вимогам 3NF, оскільки відсутні транзитивні залежності, а всі атрибути залежать лише від первинних ключів відповідних таблиць.

База даних знаходиться у нормальній формі Бойса-Кодда (BCNF) [14], якщо для кожного функціонального залежного (A → B) A є суперключем. Це більш строга версія 3NF.

База даних знаходиться в нормалізованому стані, відповідає третій нормальній формі (3NF) та нормальній формі Бойса-Кодда (BCNF). Це вважається найкращою практикою [14] з точки зору продуктивності та захисту даних. Нормалізація забезпечує мінімізацію надлишкових даних та забезпечує цілісність даних, що є важливим аспектом для систем, що потребують високого рівня безпеки та ефективності управління даними.

У системі важливо мати гнучкий механізм контролю над діями, які можуть виконувати користувачі. Для цього використовуються стани дозволів, які визначають, чи має користувач право виконувати певну дію. Кожному дозволу може бути присвоєно один із трьох станів:

– дозволено (1), визначає, що користувач має право виконувати певну дію в системі;
– заборонено (0), визначає, що користувач не має права виконувати певну дію в системі;
– невизначено (null), визначає відсутність специфікованого дозволу або заборони для користувача на виконання певної дії.

Ці правила встановлюють чіткі критерії для управління доступом у системі. У разі, якщо користувач має декілька ролей, застосовуються такі принципи:

– якщо хоча б один дозвіл має «-», тоді, дія заборонена;
– якщо дозволи мають «null», тоді дія заборонена;
– якщо дозволи мають «+» та «null», тоді дія дозволена.

Завдяки цьому підходу можна запобігти несанкціонованому доступу та непередбаченим ситуаціям, коли користувач має доступ до інформації, до якої він не повинен був мати доступ.

Далі потрібно сформувати ролі. Вони необхідні для забезпечення прозорості та контролю над тим, які дії можуть виконувати різні користувачі. Кожній ролі надається певний набір дозволів, тобто, які дії користувач з цією роллю може виконувати в системі.

Сформовано такі ролі:

– адміністратор. Передбачається повний доступ до управління користувачами та їх даними, ролей та токенів, окрім доступу до видавання направлень;
– завідувач. Передбачається доступ до управління користувачами, ролями та призначенням ролей, окрім управління токенами та можливістю видавання направлень;
– сімейний лікар. Передбачається доступ до видавань направлень;
– медсестра. Передбачається доступ до видавань направлень;
– інтерн. Не передбачається жодного доступу.

На таблиці 2.1 детальніше розписано про дозволи ролей.

Таблиця 2.1 – Ролі та дозволи

Дія	Адміністратор	Завідувач	Сімейний лікар	Медсестра	Інтерн
Управління користувачами	1	1	null	0	0
Управління ролями	1	0	null	0	0
Управління токенами	1	0	null	0	0
Призначення ролей	1	1	null	0	0
Можливість видавання направлень	null	null	1	1	null

Ця схема бази даних дозволить виконувати автентифікацію та авторизацію користувачів, керувати їхніми правами доступу та забезпечувати безпеку системи управління доступом. Використання різних станів доступу дозволить системі адаптуватися до потреб організації та гнучко налаштовувати права доступу для різних користувачів. Такий підхід також допомагає забезпечити високий рівень безпеки, запобігаючи несанкціонованому доступу та збільшуючи контроль над користувачами та інформацією до якої вони мають доступ.

2.3 Структура токена

При проєктуванні систем управління доступом, ключовою складовою є структура та формат токенів, які використовуються для автентифікації користувачів. Структура токена має важливе значення для забезпечення безпеки та цілісності даних, а також для забезпечення ефективного управління доступом до ресурсів.

Заголовок токена містить ключові поля, за якими можна визначити алгоритм підпису («alg») та тип токена («typ»).

У тілі токена зберігаються різноманітні атрибути, що визначають користувача та його доступ до системи:

– «id» – ідентифікатор користувача;

– «fullname» – повне ім’я користувача;

– «exp» – дата терміну токена;

– «startwork» – початок роботи, тобто о котрій користувач може автентифікуватися;
– «endwork» – закінчення роботи, тобто о котрій користувач більше не може користуватись системою;
– «time» – скільки годин працює токен;
– «jobtitle» – посада користувача;
– «device» – робоча станція з якого здійснюється вхід в систему.

Нижче на рисунку 2.4 зображено структуру токена.

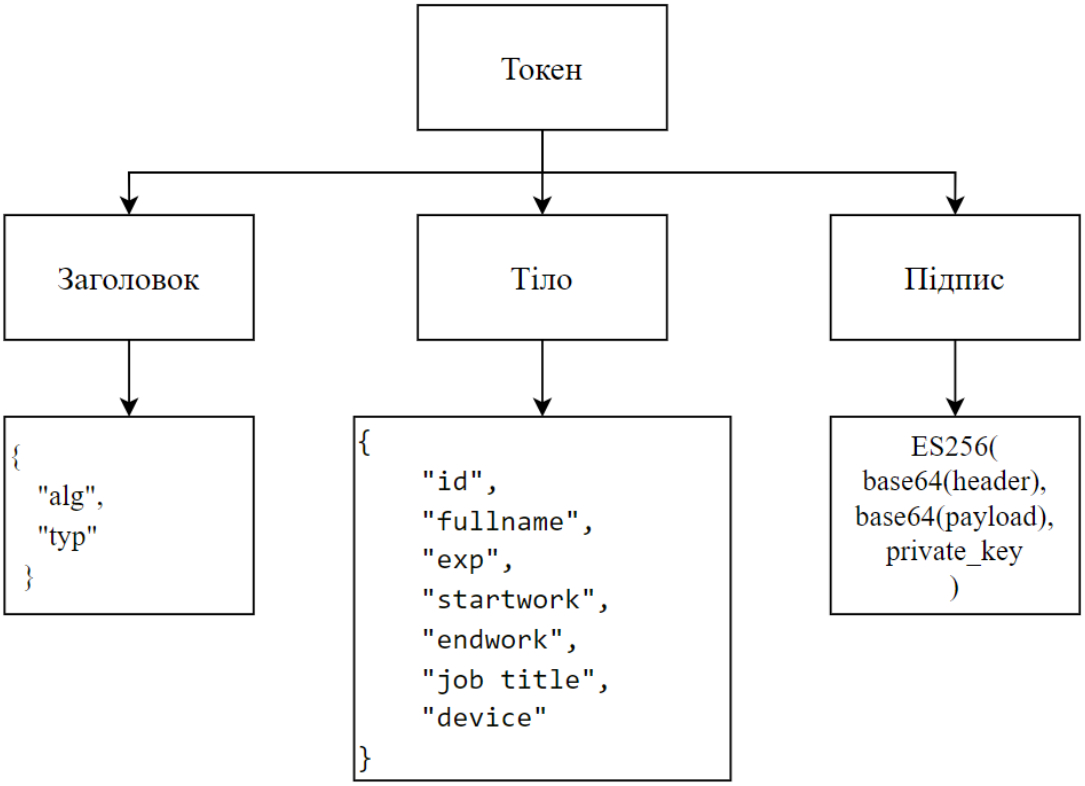


Рисунок 2.4 – Структура токена

Інформація в тілі токена є необхідною для ідентифікації користувача та його дій у системі, а також для встановлення обмежень та контролю за доступом до системи. Перед формуванням підпису за допомогою криптографічного алгоритму, заголовок та тіло токена кодуються в форматі base64 для забезпечення конфіденційності та цілісності даних. Закритий ключ, використаний для генерації

підпису, забезпечує недоступність вмісту токена для зловмисників та підтверджує автентичність даних.

Отже, структура токена містить додаткову інформацію про користувача та його доступ до системи, а також механізми безпеки, такі як підпис та кодування даних. За допомогою цих полів, здійснюється ідентифікація та автентифікація користувачів у системі.

2.4 Архітектура програмного коду

Основний функціонал програми спрямований на забезпечення автентифікації та авторизації користувачів, керування їхніми правами доступу та інтеграцію з базою даних для зберігання інформації про користувачів та їх ролі.

Система має забезпечувати гнучкий контроль над правами доступу користувачів, щоб адміністратори могли налаштовувати, які дії можуть виконувати користувачі залежно від їхніх ролей та привілеїв.

Для взаємодії користувача з системою потрібен веб-інтерфейс. Всього буде 5 форм:

- для автентифікації;
- для введення даних про користувача;
- для ведення даних токена;
- для ведення даних про роль;
- для з’єднування користувача та ролі.

Для кожної форми необхідний скрипт щоб обробити дані та надіслати їх до серверу.

Серверна частина програми буде відповідальною за обробку запитів користувачів та взаємодію з базою даних. Функції сервера будуть включати не лише запис, редагування та видалення даних з бази даних, а й отримання даних для подальшого відображення на веб-інтерфейсі.

Однією з ключових функцій системи буде реалізація двофакторної автентифікації. Для цього буде розроблено механізми шифрування паролю,

створення та верифікації токенів, що дозволить забезпечити додатковий рівень безпеки при вході користувачів до системи та кращий контроль за тим, де і коли здійснюється вхід в систему.

Створено UML діаграму, яка відображає взаємозв'язки між складовими програмного коду:

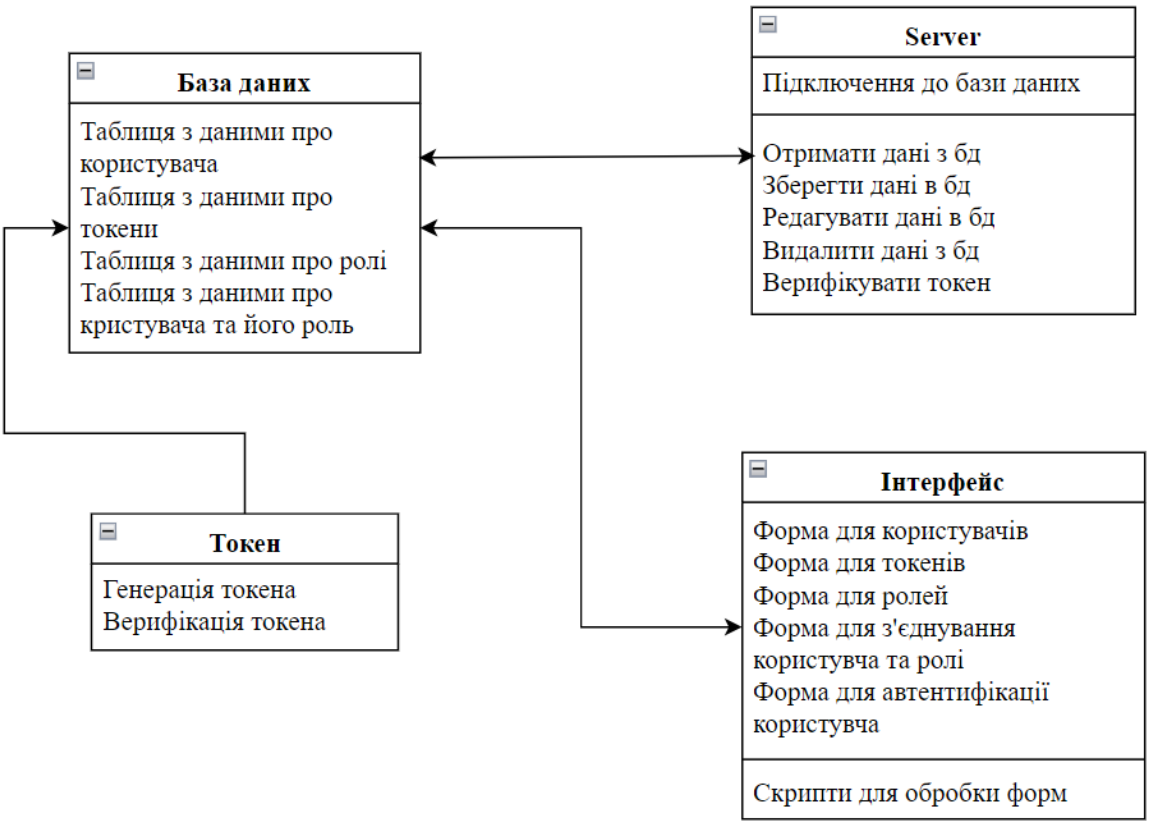


Рисунок 2.5 – UML діаграма програмного коду

Програмний код системи буде спроєктований з урахуванням потреб системи: автентифікації та авторизації користувачів, керування їхніми правами доступу та взаємодії з базою даних.

2.5 Висновки з розділу

У другому розділі було представлено ключові аспекти для розробки програмного засобу. Розглянуто структуру бази даних, формат токенів та програмний код системи, що забезпечують безпеку та функціональність системи.

База даних була спроєктована таким чином, щоб уникнути аномалій при оновленні. Таблиці для користувачів, токенів, ролей та їх зв'язків були створені для управління даними та контролю над доступом.

Формат та структура токенів дозволяють ідентифікувати користувача та його доступ до системи, що забезпечує високий рівень захисту інформації. Використання двофакторної автентифікації додатково підвищує рівень безпеки, зменшуючи ризик несанкціонованого доступу до системи.

Впроваджено механізми гешування паролів та створення і верифікації токенів, що забезпечує додатковий рівень безпеки при вході користувачів до системи. Це також сприяє кращому контролю за тим, де і коли здійснюється вхід у систему.

3 АЛГОРИТМ РОБОТИ МОДУЛЯ РОЗМЕЖУВАННЯ ПРАВ ДОСТУПУ

3.1 Узагальнений алгоритм роботи засобу

Оскільки в попередньому розділі обрано клієнт-серверну архітектуру, тому варто врахувати це при розробці алгоритму роботи засобу.

Результати розробки узагальненого алгоритму роботи засобу зображено на рисунку 3.1.

Рисунок 3.1 – Узагальнений алгоритм роботи застосунку

На першому етапі здійснюється автентифікація користувача де він вводить логін пароль та токен, якщо роль користувача має необхідні дозволи тоді після

успішної автентифікації користувач заповнює необхідну форму даними, наприклад, реєстрація нового користувача чи створення нової ролі. Далі сервер отримує дані від клієнта, зчитує їх і починає процес обробки.

Після успішної обробки сервер визначає, яку саме операцію потрібно виконати, наприклад створення, оновлення чи видалення користувача. Після обробки запиту до бази даних сервер надсилає відповідь назад клієнту з результатами операції.

3.2 Алгоритм роботи модуля клієнта

Модуль клієнта – це користувацький інтерфейс, який відповідає за взаємодію сервера з користувачем.

Першим етапом йде автентифікація користувача, який повинен ввести логін, пароль та файл з токеном на формі з автентифікацією. На стороні сервера правдиться перевірка користувача в базі даних, якщо користувач існує, тоді з бази даних витягаються його ролі та дозволи. Перед кожною функцією перевіряється чи має користувач доступ. Після успішної автентифікації та перевірки дозволів користувач може заповнювати різні форми в залежності від своїх прав доступу.

Всі форми в системі:

- форма для створення, видалення та оновлення користувача;
- форма для створення та видалення токена;
- форма для створення, видалення та оновлення ролі;
- форма для пов’язування користувача та ролі.

Крім того, розроблений модуль повинен надавати модулю, що розробляється в іншій частині комплексної бакалаврської дипломної роботи інформацію про можливість доступу користувача до системи видавання направлень. Це дозволить виконувати розмежування прав доступу користувачів до системи загалом, в той час як форми покликані надати адміністративний доступ до розмежування прав.

Алгоритм роботи модуля клієнта зображено на рисунку 3.2.

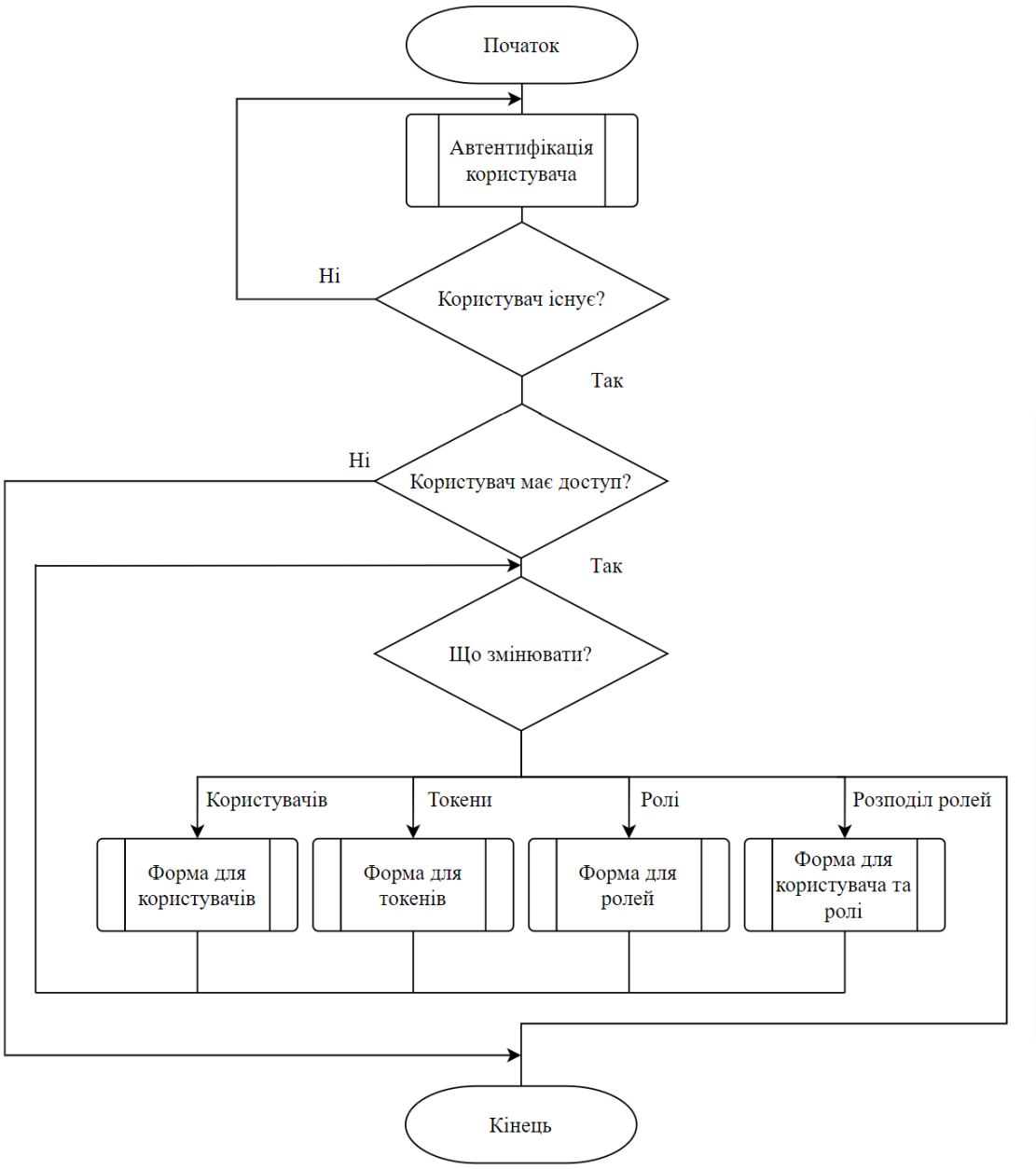


Рисунок 3.2 –Алгоритм роботи модуля клієнта

Після заповнення, дані з форми надсилаються для подальшої обробки. Далі на стороні клієнта зчитуються введені користувачем дані і готуються для відправки на сервер.

3.3 Алгоритм роботи модуля сервера

Алгоритм роботи модуля сервера складається з декількох ключових етапів, які забезпечують взаємодію з клієнтами та базою даних. Алгоритм зображено на рисунку 3.3.

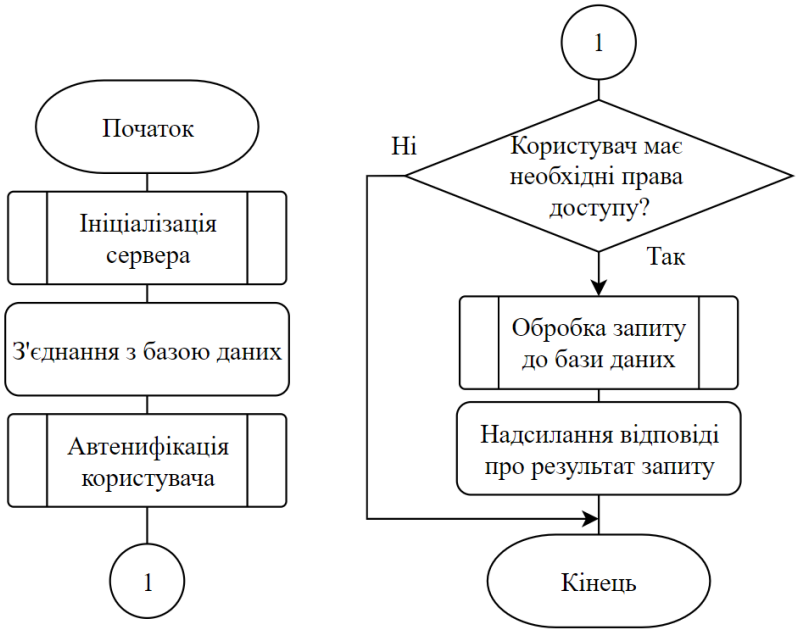


Рисунок 3.3 – Алгоритм роботи модуля сервера

На першому етапі здійснюється запуск сервера. Ініціалізуються всі необхідні компоненти та налаштування, які забезпечують роботу сервера.

На другому етапі встановлюється з’єднання з базою даних. Сервер здійснює перевірку доступності бази даних та обробляє можливі помилки з'єднання.

Далі здійснюється автентифікація користувача. З модуля клієнта на модуль сервера надходять дані з форми автентифікації, які вводить користувач. Отримані дані проходять перевірку на стороні сервера, що включає перевірку на наявність спеціальних символів, форматування та довжину. Після цього сервер порівнює отриманий геш пароля з гешем, збереженим у базі даних, для підтвердження автентичності користувача. Далі перевіряється токен на відповідність робочій станції, часу та іншим параметрам. Також визначаються права доступу користувача.

Після успішної автентифікації сервер переходить до обробки запиту до бази даних. В залежності від форми та типу, сервер надсилає відповідний SQL-запит з параметрами до бази даних, якщо користувач має необхідні права доступу, якщо так, то далі сервер формує відповідь для клієнта на основі результатів виконання

запиту до бази даних. Відповідь може бути про успішне виконання операції або повідомлення про помилку. Після цього відповідь надсилається клієнту.

3.4 Алгоритм роботи модуля взаємодії з базою даних

Основні етапи алгоритму роботи модуля взаємодії з базою даних включають підключення сервера до бази даних, виконання SQL-запитів, обробку результату запиту.

Алгоритм роботи модуля взаємодії з базою даних зображено на рисунку 3.4.

На першому етапі сервер підключається до бази даних. Далі сервер отримує від клієнта дані з форми, після чого сервер в залежності від типу операції, формує відповідний SQL-запит. Для захисту від SQL-ін'єкцій, параметри запиту передаються окремо від SQL-інструкцій.

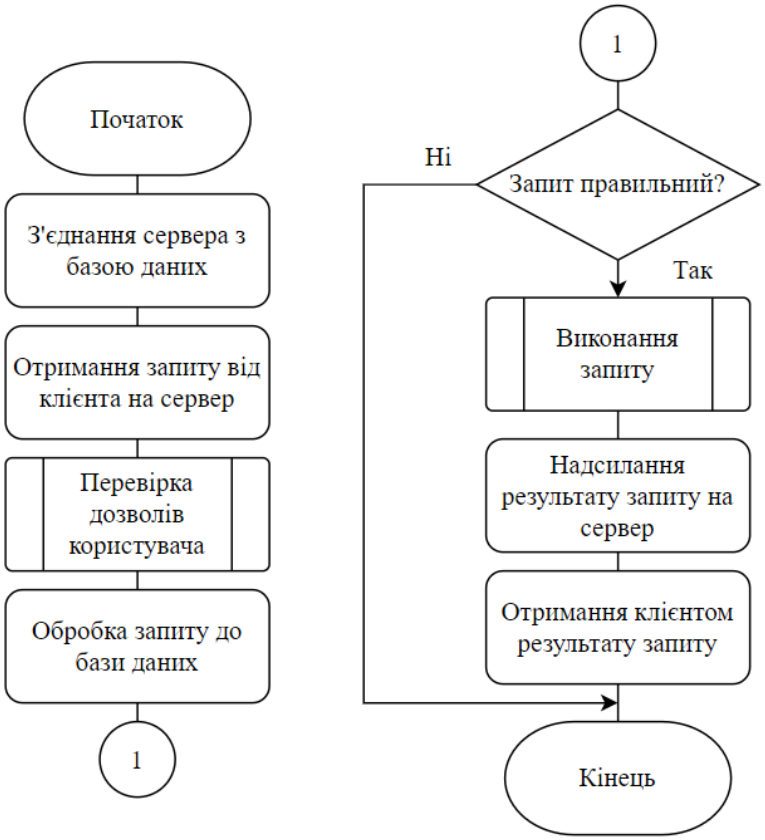


Рисунок 3.4 – Алгоритм роботи модуля взаємодії з базою даних

Якщо у користувача є необхідні дозволи для надсилання запитів до бази даних, тоді виконується SQL-запит. Модуль виконує запит до бази даних, використовуючи встановлене з'єднання.

Після виконання запиту користувач отримує результати. Якщо запит був успішним, модуль отримує результати запиту та обробляє їх відповідно до потреб програмного засобу. Наприклад, якщо запит був на отримання даних з бази даних, то результати будуть відображені на формі.

3.5 Алгоритм створення токена

Для підвищення безпеки доступу до медичної інформації використовується технологія JWT як метод автентифікації. Алгоритм створення токена зображено на рисунку 3.5.

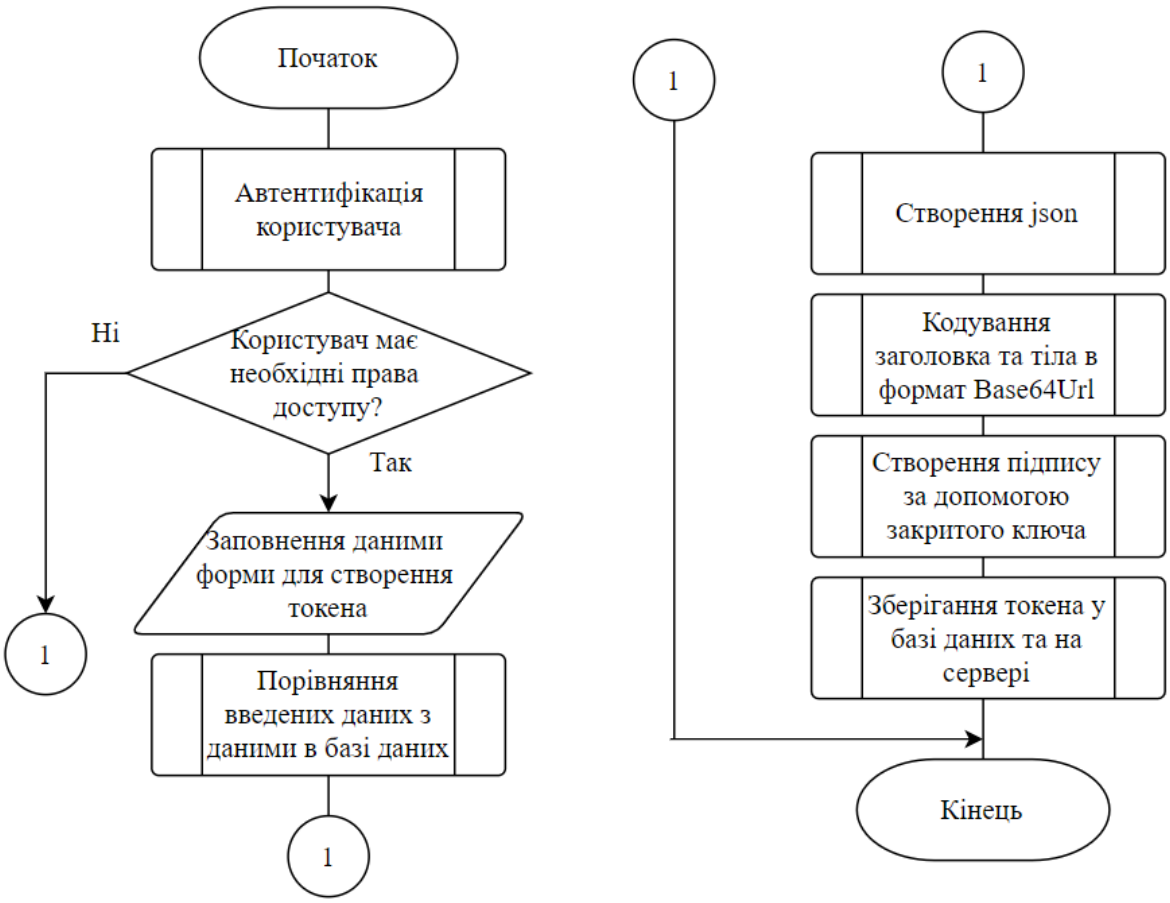


Рисунок 3.5 – Алгоритм створення токена

Процес створення токена починається з автентифікації, якщо у користувача є права для цього, тоді він може заповнити відповідну форму даними. Після цього система приймає введені користувачем дані для подальшої обробки, де відбувається їх перевірка. Сервер порівнює логін з тим, що зберігається в базі даних. Якщо все правильно, сервер переходить до наступного кроку, а в разі неправильних даних система повертає помилку.

Далі для генерації токена створюється json в якому є заголовок токена та тіло, яке містить основну інформацію про користувача та параметри токена. Ці частини кодуються в формат Base64Url. Потім використовується закритий ключ і алгоритм ES256 для створення криптографічного підпису токена.

Закодовані заголовок, тіло та підпис з’єднуються через крапки, формуючи повний токен у форматі: `<header>.<payload>.<signature>`. Готовий токен зберігається у базі даних та на сервері.

3.6 Алгоритм автентифікації

Алгоритм автентифікації, як складова системи забезпечення безпеки, пропонує чітко визначений алгоритм взаємодії користувача і програмного засобу.

Алгоритм роботи автентифікації зображено на рисунку 3.6.

На першому етапі, система перевіряє наявність користувача в своїй базі даних. Відсутність запису про користувача призводить до відмови в доступі до програмного засобу. Щоб отримати доступ до ресурсів системи користувач повинен бути зареєстрованим користувачем з необхідними правами доступу.

Далі, користувач має ввести свій логін, пароль і токен, отриманий від іншого користувача з відповідними правами доступу. Сервер перевіряє чи існує логін в системі, далі перевіряється геш пароля.

Після цього система аналізує токен. Важливою частиною є перевірка підпису токена, терміну його дії, а також інших параметрів, таких як час та робоча станція, що використовується для входу.

Рисунок 3.6 – Алгоритм автентифікації користувача

Таким чином, схема забезпечує перевірку автентичності для захисту системи від несанкціонованого доступу.

3.7 Висновки з розділу

У третьому розділі було розроблено алгоритм роботи засобу побудований на основі клієнт-серверної архітектури, який надає чітку послідовність дій для автентифікації користувача, обробки його запитів та надсилання відповідей.

Алгоритми роботи модуля клієнта та сервера надають структурований підхід до взаємодії між користувачем і системою через клієнтсько-серверну архітектуру. Модуль клієнта передбачає автентифікацію користувача та його можливість заповнення різних форм, що визначаються правами доступу. Серверний модуль ініціалізує всі необхідні компоненти, встановлює з’єднання з базою даних та проводить автентифікацію користувача.

Алгоритм роботи модуля взаємодії з базою даних забезпечує обробку запитів від клієнтів до бази даних.

Розроблено алгоритм автентифікації, яка включає чітко визначений алгоритм взаємодії користувача і програмного засобу. Важливою частиною процесу є аналіз токена, включаючи перевірку підпису, терміну дії токена та інших параметрів.

4 ТЕСТУВАННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ

4.1 Обґрунтування виборів засобів розробки

Для реалізації програмного засобу обрано мову програмування – JavaScript [16]. Це рішення прийнято на основі порівняння з іншими відомими мовами програмування, такими як Python [17], Java [18] та C# [19].

JavaScript, завдяки своєму асинхронному характеру та моделі подій, швидко обробляє численні одночасні запити. Це особливо важливо для веб-застосунків, де необхідно швидко реагувати на дії користувачів та взаємодію з сервером. На відміну від Python, який зазвичай вимагає додаткових бібліотек для асинхронної обробки (наприклад, asyncio [20]), JavaScript має вбудовану підтримку асинхронних функцій.

JavaScript – мова, яка може використовуватися як на клієнтській (в браузері), так і на серверній стороні (з Node.js [21]). Це спрощує розробку, завдяки чому можна використовувати одну мову для всього стеку технологій. Java та C# також підтримують серверну розробку, але їх використання на клієнтській стороні обмежене, що ускладнює узгодження та підтримку коду.

JavaScript має одну з найбільших спільнот розробників та безліч бібліотек і фреймворків (наприклад, React [22], Angular [22], Vue.js [22] для фронтенду та Express [23] для бекенду). Python також має потужну екосистему, але для веб-розробки він зазвичай використовує Django [24] або Flask [24], що може бути складнішим у налаштуванні та інтеграції з фронтендом.

JavaScript є однією з найпопулярніших мов програмування, що спрощує пошук схожих рішень та вирішення схожих проблем. Хоча Python також є популярною мовою, він більше підходить для наукових досліджень та аналізу даних, а не для високопродуктивних веб-застосунків.

Node.js був обраний як серверна платформа. Node.js ідеально підходить для створення мікросервісів, що дає можливість розробляти застосунки з гнучкою

архітектурою, де кожен компонент може бути незалежно масштабований та оновлюваний.

Для забезпечення безпеки та управління правами доступу була використана бібліотека jsonwebtoken [25], яка створює, підписує та перевіряє токени, що необхідно для реалізації багатофакторної автентифікації.

Розробка інтерфейсу програмного засобу здійснюється з використанням HTML [26], CSS [26] та JavaScript. HTML використовується для створення структури веб-сторінок. За допомогою HTML визначаються основні елементи інтерфейсу, такі як форми, кнопки, таблиці, заголовки та інші компоненти, що відображаються на сторінці. HTML забезпечує базову розмітку, яка є основою для подальшого стилювання та додавання функціональності.

CSS відповідає за стилізацію HTML-елементів, налаштовуючи зовнішній вигляд веб-сторінок. Використання CSS забезпечує узгоджений та привабливий інтерфейс для користувачів, полегшуючи сприйняття та взаємодію з веб-застосунком.

JavaScript використовується для додавання динамічної поведінки та інтерактивності до веб-сторінок. За допомогою JavaScript реалізовано обробку подій, таких як натискання кнопок, введення даних у форми, зміни вмісту сторінки без її перезавантаження та інші взаємодії з користувачем.

Visual Studio Code (VS Code) [27] – це потужний, безкоштовний та відкритий редактор коду, розроблений компанією Microsoft. Він є одним із найпопулярніших інструментів серед розробників завдяки своїй гнучкості, розширюваності та підтримці різних мов програмування.

VS Code підтримує JavaScript та багато інших мов програмування. Завдяки цьому можна працювати в одному середовищі без необхідності перемикатися між різними редакторами для різних завдань. Також там присутні безліч розширень, які можна легко встановити для додаткових функцій, таких як підтримка фреймворків, бібліотек, інструментів для налагодження, інтеграція з системами контролю версій та багато іншого.

Використання розширень для живого перегляду (Live Server [28]) дозволяє бачити зміни в коді в режимі реального часу в браузері. Це прискорює процес розробки, оскільки немає потреби постійно перезавантажувати сторінку для перевірки змін.

Для тестування використовується Jest [29], який є потужним фреймворком для тестування у JavaScript. Він забезпечує широкий набір можливостей для написання та виконання тестів, включаючи автоматичне спостереження за змінами файлів, масштабованість тестових наборів та інші. Окрім Jest, існують інші фреймворки, такі як Mocha [29] та Jasmine [29], але вони менш зручні: Mocha вимагає додаткових налаштувань, а Jasmine має менш активну спільноту/

Серед різних систем управління базами даних (СУБД) обрано SQL Server Management Studio [30]. Цей вибір обґрунтовано високою функціональністю, продуктивністю та надійністю у роботі з базами даних Microsoft SQL Server, що робить його ідеальним інструментом для розробки, моделювання та адміністрування великих баз даних у великих проєктах. Також SQL Server відомий своєю здатністю управляти великим обсягом даних та швидким виконанням операцій завдяки своїм оптимізованим механізмам обробки запитів. Це робить його ідеальним вибором для проєктів будь-якого розміру, від невеликих до великих корпоративних застосунків.

Oracle Database [31] також є потужною СУБД, відомою своєю. Використовується у великих корпоративних застосунках, системах управління даними та системах аналітики. Проте, Oracle робить акцент на адміністрування та має складнішу структуру налаштування, що може ускладнити роботу з ним для розробників. Це робить Oracle менш практичним для проєктів, де необхідна швидкість і простота розробки та обслуговування. Т

Для обробки та виконання запитів в базу даних обрано mssql [32], який має високу популярність та широкий функціонал. Його вибір обумовлений відмінною сумісністю з SQL Server, що дозволяє виконувати операції з базою даних. Важливою перевагою є також широкий спектр функцій та можливостей для

оптимізації роботи з базою даних, що сприяє покращенню продуктивності та швидкості розробки.

4.2 Блокове тестування

Блокове тестування полягає у тестуванні конкретного блоку програми, який може бути класом, методом або навіть невеликою програмою в цілому. Головна мета полягає в перевірці правильності роботи цього блоку, його функцій або методів, без прив'язки до інших частин системи.

Завдяки блоковому тестуванні можна розглядати окремі частини програми як самостійні одиниці, які можна тестувати і перевіряти на коректність роботи без необхідності включення інших компонентів або системи в цілому.

Для тестування обрано бібліотеку Jest, яка є зручною та доступною для використання.

Тестувались основні функції на клієнті:

- функції, які призначені для отримання даних з форми на веб-сторінку;
- функція, яка призначена для підпису JWT-токенів на основі переданого тіла.

Було написано тести, які полягали в наступному:

- вхід: порожній об'єкт, очікування: виникне помилка;
- вхід: неправильний формат номеру ліцензії «abc», очікування: виникне помилка «номер ліцензії повинен містити тільки цифри.»;
- вхід: неправильний формат номеру ліцензії з пробілами спереду та ззаду « 123v », очікування: виникне помилка «номер ліцензії повинен містити тільки цифри.»;
- вхід: порожнє поле логіну, очікування: виникне помилка «логін користувача не може бути порожнім.»;
- вхід: відсутність елементу ролі, очікування: null;
- вхід: відсутність елементів дозволів, очікування: null;
- вхід: неправильні значення дозволів, очікування: null;

– вхід: неправильний об'єкт payload, очікування: виникне помилка;
– вхід: неправильний формат номеру ліцензії «123v», очікування: виникне помилка «номер ліцензії повинен містити тільки цифри.»;
– вхід: порожнє поле логіну, очікування: виникне помилка «логін користувача не може бути порожнім.»;
– вхід: відсутність елементу ролі, очікування: null;
– вхід: відсутність елементів дозволів, очікування: null;
– вхід: неправильні значення дозволів, очікування: null.

Всі тести пройшли успішну перевірку, результат успішного тестування зображено на рисунку 4.1.

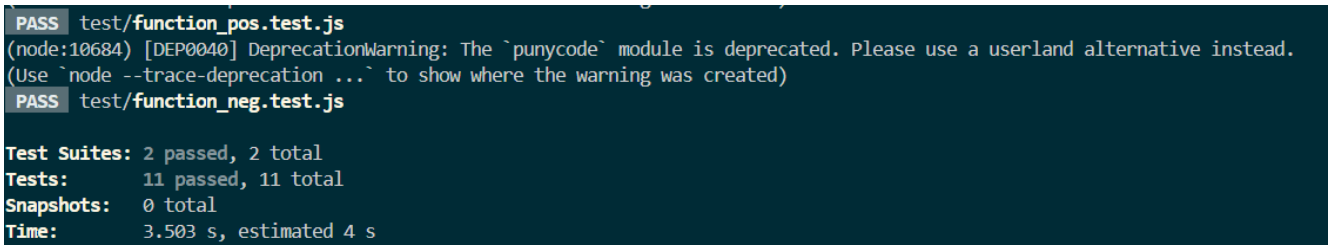


Рисунок 4.1 – Результати позитивного та негативного тестування

Завдяки цьому підходу можна виявляти та виправляти помилки в конкретних частинах програми, що спрощує процес відлагодження коду і поліпшує роботу програмного засобу. Блокове тестування також може бути легко автоматизовано та регулярно застосоване під час розробки та підтримки програми.

4.3 Інтеграційне тестування

Інтеграційне тестування – це тип тестування програмного забезпечення, який спрямований на перевірку взаємодії між різними компонентами або модулями системи як єдиного функціонального блоку. Основна мета інтеграційного тестування - переконатися, що окремі компоненти програми працюють правильно разом, відповідно до вимог.

Для початку необхідно протестувати перший модуль – додавання, видалення або оновлення записів в БД за допомогою користувацького інтерфейсу.

У ролі користувача виступає адміністратор, який має майже всі дозволи в системі. Дані, отримані з форми, надсилаються на сервер, де йде обробка даних для запису в БД. Спершу необхідно запустити сервер та перейти за посиланням «http://127.0.0.1:3000/index.html».

На прикладі, користувач обирає форму для реєстрації іншого користувача в систему:

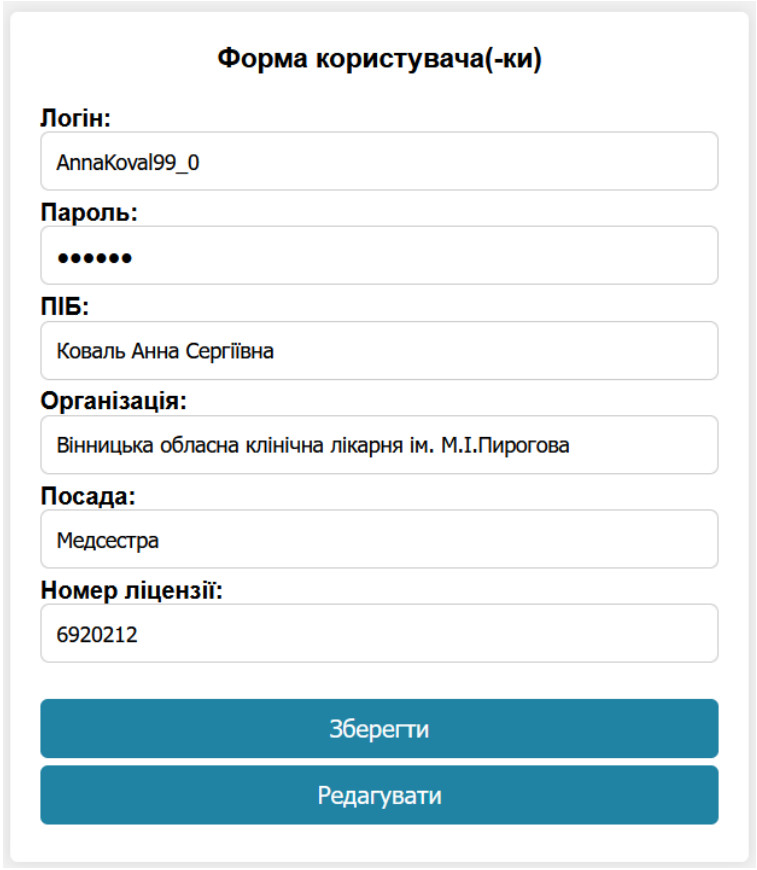


Рисунок 4.2 – Заповнена форма з додаванням користувачки

Користувач заповнює форму для реєстрації нової користувачки в системі, вводячи необхідні особисті дані. Необхідні ролі відповідно можна призначити у формі для призначення ролей.

Результат успішного запису даних в БД наведена на рисунку 4.3.

	Login	FullName	Organization	Position	NumberLicense	Password
1	AndriiMelnyk87_0	Андрій Мельник	Вінницька обласна клінічна лікарня ім. М.І.Пирог...	Сімейний лікар	142252	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3
2	AnnaKoval99_0	Коваль Анна Сергіївна	Вінницька обласна клінічна лікарня ім. М.І.Пирог...	Медсестра	6920212	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3

Рисунок 4.3 – Результат доданої інформації в БД

Якщо потрібно видалити або оновити дані користувачки, тоді необхідно натиснути на кнопку «Редагувати» та заповнити поля. Після натискання кнопки «Редагувати» відкриється форма, де при виборі користувача, інші поля заповнюються інформацією з бази даних, яка відповідно відповідає логіну в таблиці «Users». Також з’являться дві кнопки «Зберегти зміни» та «Видалити».

Приклад оновлення даних користувача наведено на рисунку 4.4.

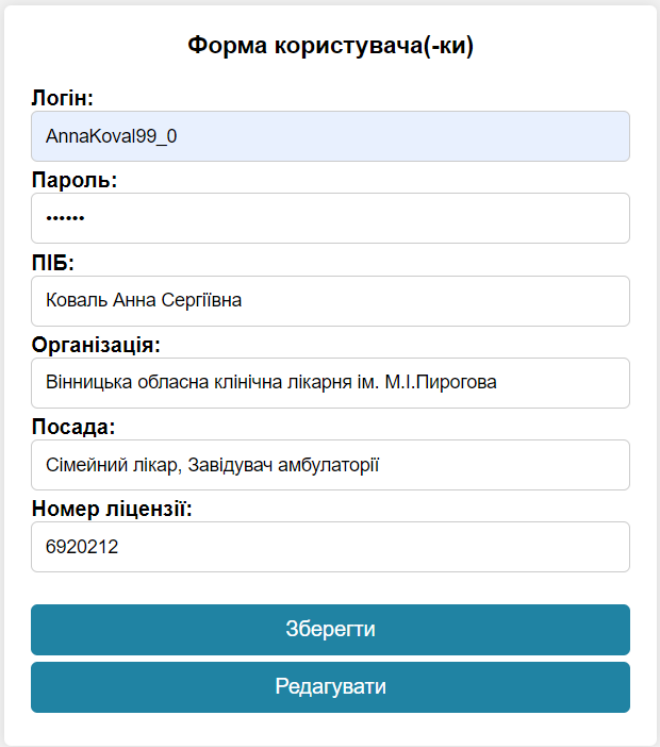


Рисунок 4.4 – Заповнена форма з оновленням даних користувачки

Після заповнення форми, потрібно натиснути на кнопку «Зберегти зміни». Результат наведено на рисунку 4.5.

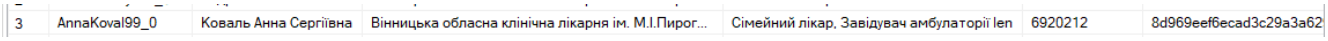


Рисунок 4.5 – Результат оновлених даних в БД

Після натискання кнопки «Зберегти зміни», система оновлює дані користувачки в базі даних, враховуючи всі введені дані.

Наступним етапом є створення токена для користувачки, що є ключовим елементом забезпечення безпеки та функціональності в системі. Цей токен

ідентифікує користувача, керує його доступом до ресурсів системи з урахуванням встановлених правил.

Приклад створення токена зображено на рисунку 4.6.

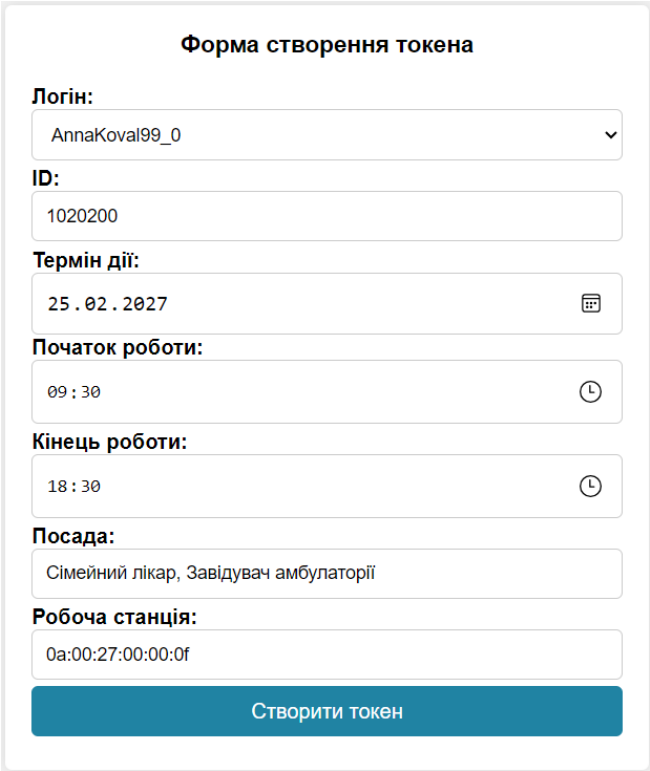


Рисунок 4.6 – Заповнена форма для створення токена

Результати збереження токена в БД наведено на рисунку 4.7.

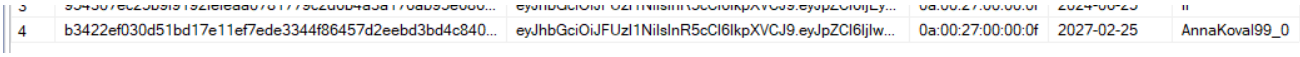


Рисунок 4.7 – Результат запису в БД створеного токена

Токен не лише записується в базу даних, а також створюється текстовий файл на сервері, який можна надіслати користувачу.

Після цього користувач може здійснити вхід в систему (рис. 4.8).

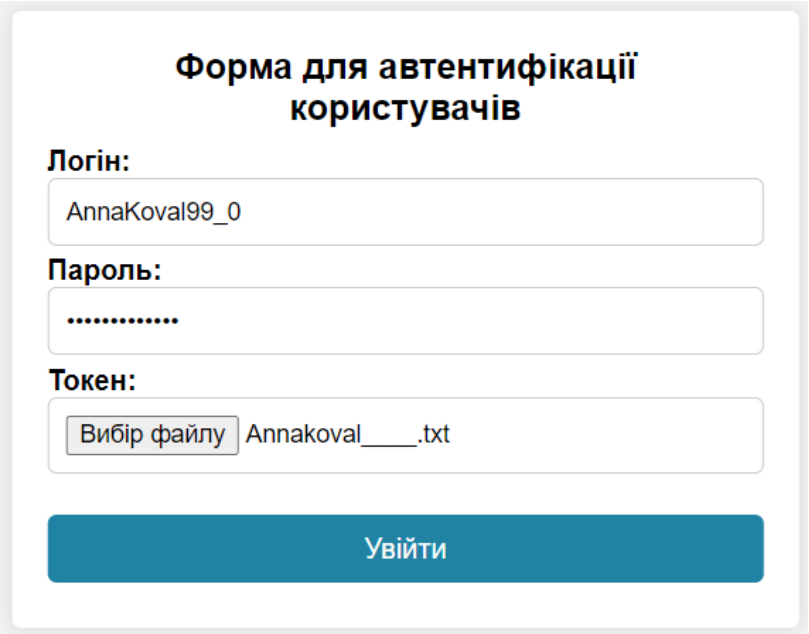


Рисунок 4.8 – Вхід в систему

Далі, після автентифікації, на приклади користувача з роллю «Адміністратор», користувач бачить головну сторінку (рис. 4.9).

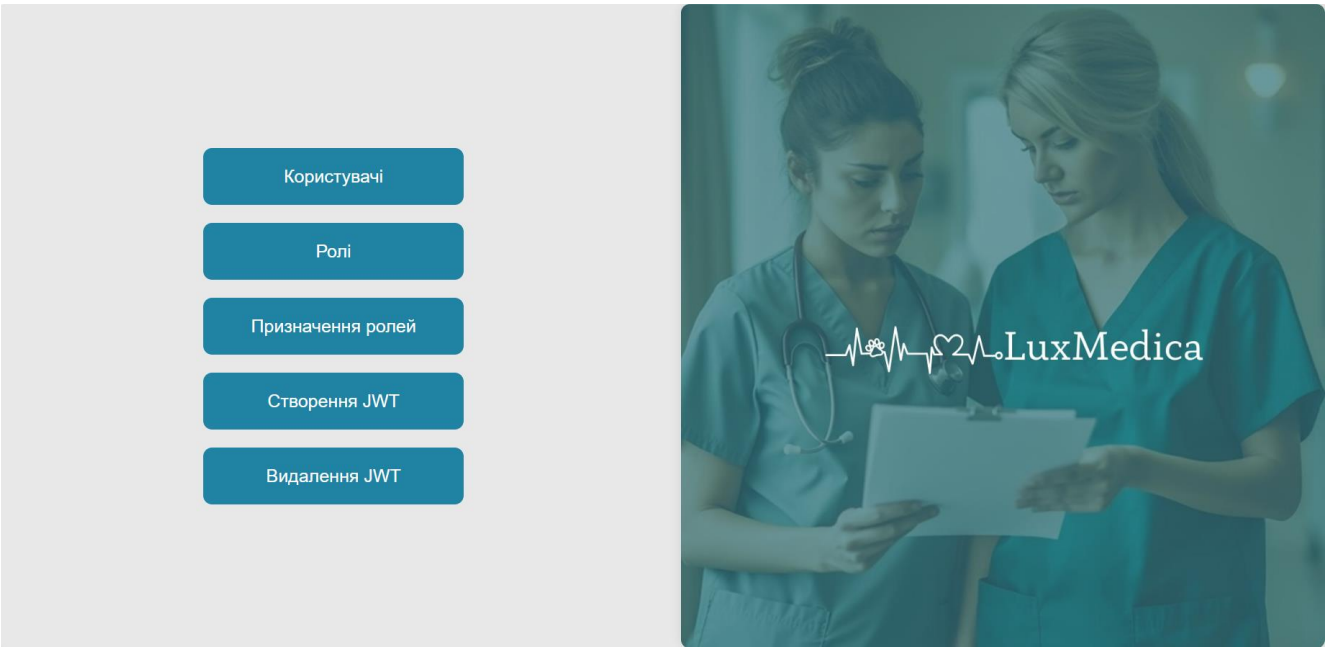


Рисунок 4.9 – Головна сторінка

Натиснувши на кнопку «Ролі», користувач перейде на нову веб-сторінку, де побачить форму для створення, оновлення та видалення ролей. Якщо користувач хоче створити роль, тоді заповнює форму (рис. В.1), після чого натискає на кнопку

«Зберегти», результат виконання запиту можна перевірити у базі даних (рис. В.2), а також відображається на сервері (рис. В.3). Якщо необхідно оновити дані, тоді потрібно натиснути на кнопку «Редагувати». З’являться дві кнопки: «Зберегти зміни» та «Видалити». Для того, щоб переконатись, що запити виконано успішно, потрібно перевірити базу даних (рис. В.4-В.7).

Для того, щоб призначити ролі користувачу необхідно обрати «Призначення ролей» на головній сторінці. На формі є лише два поля, для користувача та ролі. Кнопки відповідно працюють як і в попередній формі. Результати виконання успішного запиту можна побачити в базі даних (рис. В.8-В.9). Для відкликання ролі потрібно натиснути на кнопку «Редагувати», з’являться кнопки для відкликання ролі та відображення інформації, які ролі є в користувача (рис. В.10-В.12)

Для видалення токена потрібно перейти на відповідну форму, та обрати логін користувача (рис. В.13), після чого натиснути на кнопку «Видалити». Результати успішного запиту можна перевірити на сервері та в БД (рис. В.14-В.15).

Далі необхідно зробити перевірку доступу користувачів, якщо буде більше однієї ролі. Для цього потрібно обрати користувача з БД на формі (рис. В.16), призначити йому ролі (рис. В.17) та створити токен (рис. В.18). Потім здійснюється автентифікація, яка переносить користувача на головну сторінку, де можна побачити які форми доступні (рис. В.19-В.20). У разі, якщо користувач знайде спосіб потрапити до форми, до якої у нього немає доступу, тоді з’явиться діалогове вікно з повідомлення про відсутність прав доступу для цих дій (рис. В.21). з

При необхідності видалення користувача з системи, потрібно на формі для користувачів натиснути «Редагувати», обрати користувача та натиснути «Видалити», після цього з бази даних видалиться всі дані про нього відповідно з інших таблиць теж, де є його токени та ролі

Якщо необхідно видалити користувача, тоді потрібно обрати кнопку «Видалити». При натисканні на цю кнопку дані користувача видаляються не лише з таблиці «Users» в базі даних, але і інші пов’язанні дані з логіном користувачки (рис. В.22-В.28)

Отже, після інтеграційного тестування, можна зробити висновок, що програмний засіб працює коректно.

4.4 Статичне тестування безпеки

Статичне тестування – це метод тестування програмного засобу, який проводиться без запуску програми. Його основна мета – аналізувати програмний код на предмет виявлення помилок, аномалій та потенційних проблем без його виконання.

Для статичного тестування програмного коду використано SonarLint [33], який допомагає знаходити та виправляти помилки в коді на ранніх стадіях розробки.

Результати тестування за допомогою SonarLint наведені на рисунку 4.10.

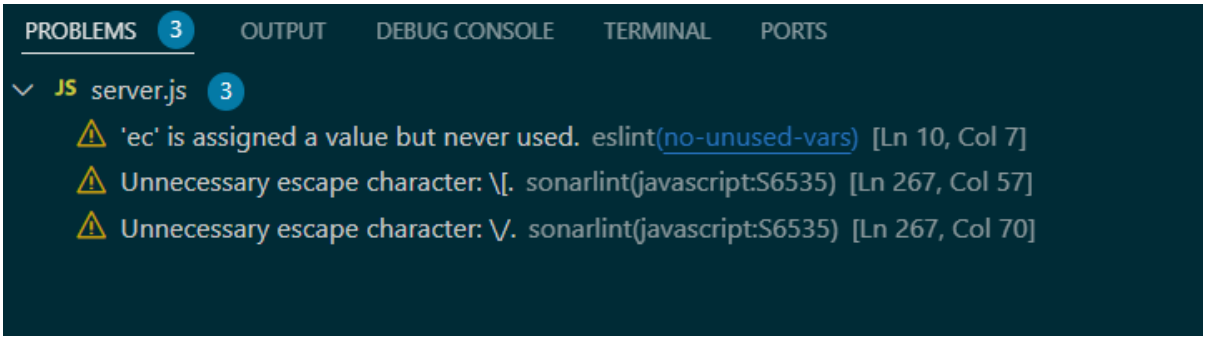


Рисунок 4.10 – Результати тестування за допомогою SonarLint

Аналіз статичного тестування коду показав, що було виявлено декілька незначних помилок та можливостей для покращення коду.

Для виявлення помилок у HTML-коді використано HTMLHint [34], який допомагає забезпечити правильну структуру та семантику документів HTML.

Результати тестування за допомогою HTMLHint наведені на рисунку 4.11.

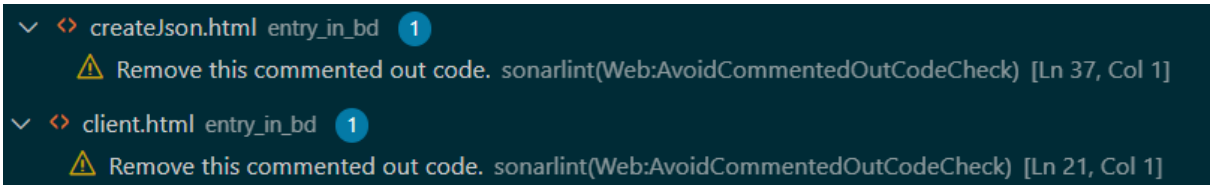


Рисунок 4.11 – Результати тестування за допомогою HTMLHint

Результати показали, що було виявлено лише незначні проблеми та які легко виправлені.

Ще одним інструментом для статичного тестування обрано ESLint [35], який використовується для виявлення та виправлення проблем у JavaScript. Він функціонує як інструмент статичного аналізу, швидко перевіряючи код на можливі проблем. Кожна функція була ретельно протестована, а детальні результати представлені на рисунку 4.11.

```
PS E:\Projects\Node_Js\AccessControl_diploma> npm run lint

> lint
> eslint . --config eslint.config.mjs --format ./eslint-reporter.mjs

Checked: E:\Projects\Node_Js\AccessControl_diploma\FormAuthentication.js - No issues found
Checked: E:\Projects\Node_Js\AccessControl_diploma\entry_in_bd\script\RolesForm.js - No issues found
Checked: E:\Projects\Node_Js\AccessControl_diploma\entry_in_bd\script\TokenForm.js - No issues found
Checked: E:\Projects\Node_Js\AccessControl_diploma\entry_in_bd\script\UsersAndRoles.js - No issues found
Checked: E:\Projects\Node_Js\AccessControl_diploma\entry_in_bd\script\client.js - No issues found
Checked: E:\Projects\Node_Js\AccessControl_diploma\server.js - No issues found

All files passed linting successfully!
```

Рисунок 4.12 – Результати тестування за допомогою ESLint

Застосування плагінів для статичного тестування дозволило виявити та усунути незначні помилки та потенційні проблеми в JS та HTML-коді, що сприяло підвищенню якості та надійності розробленого програмного засобу.

4.5 Висновки з розділу

Для реалізації програмного засобу обрано мову програмування JavaScript. Це рішення було прийнято після порівняння з іншими відомими мовами програмування, такими як Python, Java та C#. JavaScript відзначається своїм асинхронним характером та моделлю подій, швидко обробляє численні одночасні запити. Це особливо важливо для веб-застосунків, де потрібно швидко реагувати на дії користувачів та взаємодіяти з сервером.

Далі було проведено блокове тестування. Цей тип тестування дозволив перевірити правильність роботи окремих блоків програми, без прив'язки до інших компонентів системи. Використання бібліотеки Jest для тестування дозволило ефективно перевірити функціональні можливості програмного засобу на клієнтській стороні.

Наступним етапом було інтеграційне тестування, що спрямоване на перевірку взаємодії різних компонентів програмного засобу як єдиного функціонального блоку. Цей тип тестування дозволив переконатися, що окремі компоненти програми працюють правильно разом, відповідно до вимог. Інтеграційне тестування було успішно проведено, що свідчить про коректну роботу програмного засобу під час взаємодії різних його компонентів.

Статичне тестування безпеки, яке проводилося без запуску програми, було важливою частиною розробки. Застосування SonarLint та ESLint для аналізу програмного коду дозволило виявити деякі незначні помилки та потенційні проблеми, які були виправлені на ранніх стадіях розробки. Також, використання HTMLHint допомогло забезпечити правильну структуру та семантику документів HTML, що підвищило якість програмного засобу.

Таким чином, тестування допомогло забезпечити високу якість розробленого програмного засобу, зменшити кількість помилок та забезпечити його стабільну та швидку роботу.

ВИСНОВКИ

Проведено аналіз моделей розмежування прав доступу, який дав зрозуміти, що застосування лише однієї моделі у медичних інформаційних системах значно погіршить безпеку чутливих даних. Тому вирішено обрати комбіноване поєднання моделей, які б усунули недоліки та доповнили один одного.

Використання RBAC дозволило визначати доступ до ресурсів на основі ролей, що спрощує управління правами користувачів. Однак, вона може призвести до надмірності системи через велику кількість ролей, що ускладнює управління системою та може викликати проблеми з безпекою.

ABAC, з іншого боку, ґрунтується на атрибутах користувачів, ресурсів та контексті, що забезпечує гнучкість налаштовування доступу до даних в залежності від конкретного контексту. Проте, використання лише ABAC може бути складним для налаштування та громіздким.

Завдяки комбінуванню цих двох моделей в комплексній бакалаврській роботі поєднано переваги обох підходів, уникаючи їхніх недоліків та створюючи швидку, ненадмірну та безпечну систему управління доступом. RBAC може бути використаний для загального керування ролями та спрощення структури, тоді як ABAC дасть можливість точніше налаштувати доступ до ресурсів на основі конкретних атрибутів користувачів.

Проаналізовано методи автентифікації та обрано пароль та JWT. Використання двофакторної автентифікації на основі пароля та JWT значно підвищує рівень безпеки, захищаючи систему від несанкціонованого доступу.

Використання пароля є загальноприйнятим та зручним методом, оскільки він знайомий користувачам, легко підтверджує свою ідентичність шляхом введення логіну та пароля. Пароль зазвичай є частиною звичайного процесу автентифікації і вже знайомий більшості користувачів.

Використання JWT дає можливість включати додаткові атрибути у токен, такі як час та робоча станція. Це забезпечує додатковий рівень безпеки та контролю над доступом, оскільки JWT може містити інформацію про контекст

автентифікації, завдяки чому перевіряються додаткові параметри перед наданням доступу.

Описана структура автентифікації включає алгоритм взаємодії користувача з програмним засобом. Цей процес включає в себе перевірку логіна та пароля та перевірку токена. Після перевірки система визначає чи може користувач здійснити вхід в систему.

Розроблено алгоритми на основі клієнт-серверної архітектури. Ці алгоритми описують послідовність операцій, які виконуються при взаємодії з програмним засобом. Клієнтська частина відповідає за взаємодію з користувачем, серверна частина обробляє запити від клієнтів та взаємодіє з базою даних, а модуль взаємодії з базою даних виконує запити до бази. Ці алгоритми описують основні етапи роботи та забезпечують їх коректність.

Описано алгоритм створення токена, який забезпечує створення правильного токена, який використовується для автентифікації. Крім того, цифровий підпис перевіряє цілісність токена та його автентичність.

Проведене тестування підтвердило, що програмний засіб не мав критичних вразливостей. Було перевірено алгоритми клієнтського та серверного модулів, а також взаємодії з базою даних. Результати тестування показали високий рівень захисту даних та стабільну роботу системи.

Таким чином, запропонований в комплексній бакалаврській дипломній роботі підхід, сприятиме покращенню захисту чутливих даних та строгому розмежуванню прав доступу, що дозволить захистити систему від несанкціонованого доступу.

При деякій модифікації розроблений модуль розмежування прав доступу може бути застосований в інших інформаційних системах де обробляють чутливі дані, зокрема в митниці, банках та транспорті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Клиш В. М., Баришев Ю. В. Аналіз можливостей некласичних моделей розмежування прав доступу для захисту медичних даних. *LIII науково-технічна конференція підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2024)* : зб. доповідей. Вінниця : ВНТУ, 2024. С. 331-333.
2. Клиш В. М., Ланова В. С., Баришев Ю. В. Метод захищеного зберігання медичних даних за допомогою розмежування прав доступу та блокчейну. *ITSec: Безпека інформаційних технологій* : матеріали XIII Міжнар. наук.-техн. конф., м. Львів, 9-11 трав. 2024 р. Львів : ЛНУ ім. І. Франка, 2024, С. 115-116.
3. Клиш В. М., Баришев Ю. В. Модуль видавання електронних направлень в медичних закладах. *Theoretical and Applied Cybersecurity*: тези доп., 30-31 трав. 2024 р., Київ : КПІ ім. І. Сікорського, 2024.
4. Kashmar N. et al. A review of access control metamodels. *Procedia computer science*. 2021. Vol. 184. P. 445–452. DOI: 10.1016/j.procs.2021.03.056 (accessed: 20.05.2024).
5. Li, N. Discretionary Access Control. Encyclopedia of Cryptography and Security. Springer, Boston, 2011. P. 353–356 DOI: 10.1007/978-1-4419-5906-5_798 (accessed: 20.05.2024).
6. Easttom, W. Network defense and countermeasures: principles and practices. Pearson Education, Limited, 2018. 544 p.
7. Zhang, Y., Joshi, J.B. Role-Based Access Control. *Encyclopedia of Database Systems*. Springer, New York, 2018. P. 3252–3258 DOI: 10.1007/978-1-4614-8265-9_320 (accessed: 20.05.2024).
8. Vincent C. et al. Guide to Attribute Based Access Control (ABAC) Definition and Considerations. *National Institute of Standards and Technology*, 2019. P. 47. DOI: 10.6028/NIST.SP.800-162 (accessed: 20.05.2024).
9. Ameziane El Hassani A., Abou El Kalam A., Ait Ouahman A. Integrity-Organization based access control for critical infrastructure systems. *Critical infrastructure*

protection VI. Berlin, Heidelberg, 2012. P. 31–42. DOI: 10.1007/978-3-642-35764-0_3 (accessed: 20.05.2024).

10. Authentication – OWASP cheat sheet series. *Introduction – OWASP Cheat Sheet Series*. URL: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html (accessed: 20.05.2024).
11. Syed Idrus S. Z et al. A Review on Authentication Methods. *Australian Journal of Basic and Applied Sciences*, 2013, 7 (5), P. 95-107.
12. miniOrange. What is JWT (JSON Web Token)? How does JWT Authentication work? 2023. URL: https://www.miniorange.com/blog/what-is-jwt-json-web-token-how-does-jwt-authentication-work// (accessed: 20.05.2024).
13. RFC 7519: JSON web token (JWT). *IETF Datatracker*. 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519#autoid-8 (accessed: 20.05.2024).
14. Client-Server Model. URL: https://www.geeksforgeeks.org/client-server-model/ (accessed: 23.05.2024).
15. Date C. J. Database in depth: relational theory for practitioners. Beijing : O'Reilly, 2005. 230 p.
16. Сучасний підручник з JavaScript. Сучасний підручник з JavaScript. URL: https://uk.javascript.info/ (дата звернення: 11.06.2024).
17. W3Schools.com. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/python/ (accessed: 11.06.2024).
18. Burd B. Java For Dummies. Wiley, 2022. 512 p.
19. Price M. J. C# 8.0 and .NET core 3.0 – modern cross-platform development: build applications with C#, .NET core, entity framework core, ASP.NET core, and ML.NET using visual studio code, 4th edition. Packt Publishing, 2019. 818 p.
20. Asyncio – asynchronous I/O. Python documentation. URL: https://docs.python.org/3/library/asyncio.html (accessed: 11.06.2024).
21. Node.js – introduction to node.js. *Node.js – Run JavaScript Everywhere*. URL: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs (accessed: 11.06.2024).

22. Mohit Joshi. Angular vs react vs vue: core differences | browserstack. BrowserStack. 2023 URL: https://www.browserstack.com/guide/angular-vs-react-vs-vue (accessed: 11.06.2024).
23. Express 5.x - API reference. Express - Node.js web application framework. URL: https://expressjs.com/en/5x/api.html (accessed: 11.06.2024).
24. Flask vs django - python tutorial. Learn Python Programming - Python Tutorial. URL: https://pythonbasics.org/flask-vs-django/ (accessed: 11.06.2024).
25. Jsonwebtoken. npm. URL: https://www.npmjs.com/package/jsonwebtoken (accessed: 11.06.2024).
26. Український веб-довідник. Український веб-довідник. URL: https://css.in.ua/ (дата звернення: 11.06.2024).
27. Visual Studio Code. URL: https://code.visualstudio.com/docs (accessed: 11.06.2024).
28. How to enable live server on visual studio code?. GeeksforGeeks. 2022 URL: https://www.geeksforgeeks.org/how-to-enable-live-server-on-visual-studio-code/ (accessed: 11.06.2024).
29. Jest vs mocha vs jasmine: comparing the top 3 javascript testing. *LambdaTest*. URL: https://www.lambdatest.com/blog/jest-vs-mocha-vs-jasmine/ (accessed: 12.06.2024).
30. Syverson B., Murach J. Murach's SQL Server 2022 for Developers. Mike Murach & Associates, 2023. 656 p.
31. Hansen K. B. Practical oracle SQL: mastering the full power of oracle database. Apress, 2020. 460 p.
32. Mssql. npm. URL: https://www.npmjs.com/package/mssql (accessed: 12.06.2024).
33. SonarLint for visual studio. SonarQube 10.5. URL: https://docs.sonarsource.com/sonarlint/visual-studio/ (accessed: 12.06.2024).
34. Hello from HTMLHint | HTMLHint. URL: https://htmlhint.com/ (accessed: 12.06.2024).
35. Find and fix problems in your JavaScript code. ESLint – Pluggable JavaScript Linter. URL: https://eslint.org/ (accessed: 12.06.2024).

ДОДАТКИ

Додаток А

Текст застосунку. Модуль клієнта

FormAuthetication.html

```
<!DOCTYPE html>
<html lang="uk">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style2.css" />
    <script src="FormAuthentication.js" type="text/javascript"></script>
    <title>Форма для автентифікації</title>
  </head>
  <body>
    <div class="menu">
      <h1>LuxMedica</h1>
    </div>

    <div class="container_user_form">
      <h2>Форма для автентифікації користувачів</h2>
      <form id="userForm" enctype="multipart/form-data">
        <label for="login">Логін:</label>
        <input type="text" id="login" name="login" required /><br />

        <label for="pass">Пароль:</label>
        <input type="password" id="pass" name="pass" /><br />

        <label for="token">Токен:</label>
        <input type="file" id="token" name="token" />
        <br /><br />
        <button type="button" id="submitButton" onclick="submitButt()">
          Увійти
        </button>
      </form>
    </div>
  </body>
</html>
```

FormAuthentication.js

```
async function submitButt() {
  let login = document.getElementById("login").value;
  let pass = document.getElementById("pass").value;
  let tokenFile = document.getElementById("token").files[0];
  if (!tokenFile) {
    alert("Будь ласка, оберіть файл");
    return;
  }
  if (!tokenFile) {
    alert("Будь ласка, оберіть файл");
    return;
  }

  let reader = new FileReader();
  reader.readAsText(tokenFile, "UTF-8");
  reader.onload = async function (event) {
    let token = event.target.result;

    try {
      let response = await fetch("http://localhost:5500/authenticate", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({ login, pass, token }),
```

```
      });
      let data = await response.json();
      console.log(data);
      if (data.authenticated) {
        window.location.href = `entry_in_bd/index.html?login=${encodeURIComponent(
          login
        )}`;
      } else {
        alert("Неправильний логін, пароль або токен");
      }
    } catch (error) {
      console.error("Помилка при відправці запиту:", error);
      alert("Помилка при автентифікації. Будь ласка, спробуйте ще раз.");
    }
  };
}
```

client.html

```
<!DOCTYPE html>
<html lang="uk">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style\style1.css" />
    <script src="script\client.js" type="text/javascript"></script>
    <title>Форма користувача</title>
  </head>
  <body>
    <div class="container_user_form">
      <h2>Форма користувача(-ки)</h2>
      <form id="userForm">
        <label for="login">Логін:</label>
        <select id="user" name="user" style="display: none"></select>
        <input type="text" id="login" name="login" required /><br />

        <label for="pass">Пароль:</label>
        <input type="password" id="pass" name="pass" /><br />

        <label for="fullname">ПІБ:</label>
        <input type="text" id="fullname" name="fullname" required /><br />

        <label for="organization">Організація:</label>
        <input
          type="text"
          id="organization"
          name="organization"
          required
        /><br />

        <label for="position">Посада:</label>
        <input type="text" id="position" name="position" required /><br />

        <label for="numberlicense">Номер ліцензії:</label>
        <input
          type="text"
          id="numberlicense"
          name="numberlicense"
          pattern="\d*"
          inputmode="numeric"
        /><br />
        <br />
        <button type="button" id="submitButton" onclick="submitUserForm()">
          Зберегти
        </button>
        <button type="button" onclick="editButtForm()" id="editButton">
          Редагувати
        </button>
        <button
          type="button"
          onclick="editDatainBD()"
```

```
          id="edit_data_BD"
          style="display: none"
        >
          Зберегти зміни
        </button>
        <button
          type="button"
          onclick="delButtUser()"
          id="delButton"
          style="display: none"
        >
          Видалити
        </button>
      </form>
    </div>
  </body>
</html>
```

client.js

```
function data_from_form() {
  let numberlicenseInput = document.getElementById("numberlicense");
  let numberlicense = numberlicenseInput.value.trim();
  if (numberlicense !== "" && !/^\d+$/.test(numberlicense)) {
    alert("Номер ліцензії повинен містити тільки цифри.");
    return;
  }
  let login = document.getElementById("login").value;
  let password = document.getElementById("pass").value;
  let fullname = document.getElementById("fullname").value;
  let organization = document.getElementById("organization").value;
  let position = document.getElementById("position").value;
  let data = {
    login: login,
    password: password,
    fullname: fullname,
    organization: organization,
    position: position,
    numberlicense: numberlicense,
  };
  return data;
}

function submitUserForm() {
  let data = data_from_form();
  sendData("http://localhost:5500/InsertUpdateDeleteUsers", {
    action: "submit",
    ...data,
  });
  document.getElementById("editButton").style.display = "inline-block";
}

function editButtForm() {
  document.getElementById("login").style.display = "none";
  document.getElementById("submitButton").style.display = "none";
  document.getElementById("editButton").style.display = "none";
  document.getElementById("edit_data_BD").style.display = "inline-block";
  document.getElementById("user").style.display = "inline-block";
  document.getElementById("delButton").style.display = "inline-block";
  fetch("http://localhost:5500/getUserData")
    .then((response) => {
      if (response.status == 403) {
        alert("Доступ відхилено: недостатньо прав доступу.");
        throw new Error("Доступ відхилено: недостатньо прав доступу.");
      }
      return response.json();
    })
    .then((userData) => {
      let userSelect = document.getElementById("user");
      userSelect.innerHTML = "";
```

```
      let chooseOption = document.createElement("option");
      chooseOption.value = "";
      chooseOption.textContent = "--Обрати користувача(-ку)--";
      userSelect.appendChild(chooseOption);

      userData.forEach((user) => {
        let option = document.createElement("option");
        option.value = user.Login;
        option.textContent = user.Login;
        userSelect.appendChild(option);
      });
    })
    .catch((error) => {
      console.error("Помилка при отриманні даних користувача:", error);
      if (error.message !== "Доступ відхилено: недостатньо прав доступу.") {
        alert("Виникла помилка при отриманні даних користувача.");
      }
    });

  document.getElementById("user").addEventListener("change", function () {
    let selectedLogin = this.value;
    if (selectedLogin) {
      fetchUserData(selectedLogin);
    } else {
      resetUserData();
    }
  });

  function fetchUserData(selectedLogin) {
    fetch("http://localhost:5500/getUserData")
      .then((response) => {
        if (response.status === 403) {
          alert("Доступ відхилено: недостатньо прав доступу.");
          throw new Error("Доступ відхилено: недостатньо прав доступу.");
        }
        return response.json();
      })
      .then((userData) => {
        let selectedUser = userData.find(
          (user) => user.Login === selectedLogin
        );
        if (selectedUser) {
          document.getElementById("login").value = selectedUser.Login;
          document.getElementById("fullname").value = selectedUser.FullName;
          document.getElementById("organization").value =
            selectedUser.Organization;
          document.getElementById("position").value = selectedUser.Position;
          document.getElementById("numberlicense").value =
            selectedUser.NumberLicense;
        }
      })
      .catch((error) => {
        console.error("Помилка при отриманні даних користувача:", error);
        if (error.message !== "Доступ відхилено: недостатньо прав доступу.") {
          alert("Виникла помилка при отриманні даних користувача.");
        }
      });
  }

  function resetUserData() {
    document.getElementById("login").value = "";
    document.getElementById("fullname").value = "";
    document.getElementById("organization").value = "";
    document.getElementById("position").value = "";
    document.getElementById("numberlicense").value = "";
  }
}

function editDatainBD() {
  let data = data_from_form();
```

```
  sendData("http://localhost:5500/InsertUpdateDeleteUsers", {
    action: "edit",
    ...data,
  });
}

function delButtUser() {
  let data = data_from_form();
  sendData("http://localhost:5500/InsertUpdateDeleteUsers", {
    action: "delete",
    ...data,
  });
  document.getElementById("editButton").style.display = "none";
  document.getElementById("submitButton").style.display = "none";
  document.getElementById("userForm").reset();
}

function sendData(url, data) {
  fetch(url, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(data),
  })
    .then((response) => {
      if (!response.ok) {
        throw new Error("Помилка при відправці запиту.");
      }
      return response.text();
    })
    .then((message) => {
      console.log(message);
      alert(message);
      document.getElementById("userForm").reset();
    })
    .catch((error) => {
      console.error("Виникла помилка:", error);
      alert("Виникла помилка при відправці запиту.");
    });
}
```

createJson.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style/style1.css" />
    <title>Форма створення токена</title>
  </head>
  <body>
    <div class="container_token_form">
      <h2>Форма створення токена</h2>
      <form id="tokenForm">
        <label for="login">Логін:</label>
        <select id="login" name="login" required></select>

        <label for="id">ID:</label>
        <input type="text" id="id" name="id" required />

        <label for="exp">Термін дії:</label>
        <input type="date" id="exp" name="exp" required />

        <label for="startwork">Початок роботи:</label>
        <input type="time" id="startwork" name="startwork" required />

        <label for="endwork">Кінець роботи:</label>
        <input type="time" id="endwork" name="endwork" required />
```

```
    <label for="jobtitle">Посада:</label>
    <input type="text" id="jobtitle" name="jobtitle" required />

    <label for="workstation">Робоча станція:</label>
    <input type="text" id="workstation" name="workstation" required />

    <button
      type="button"
      id="submitTokenButton"
      onclick="submitTokenForm()"
    >
      Створити токен
    </button>
  </form>
</div>
<script>
  document.addEventListener("DOMContentLoaded", function () {
    fetch("http://localhost:5500/getUserData")
      .then((response) => {
        if (!response.ok) {
          alert("Доступ відхилено: недостатньо прав доступу.");
          throw new Error("Доступ відхилено: недостатньо прав доступу.");
        }
        return response.json();
      })
      .then((userData) => {
        let userSelect = document.getElementById("login");
        userSelect.innerHTML = "";

        let chooseOption = document.createElement("option");
        chooseOption.value = "";
        chooseOption.textContent = "--Обрати користувача(-ку)--";
        userSelect.appendChild(chooseOption);

        userData.forEach((user) => {
          let option = document.createElement("option");
          option.value = user.Login;
          option.textContent = user.Login;
          userSelect.appendChild(option);
        });
      })
      .catch((error) => {
        console.error("Помилка при отриманні даних користувача:", error);
      });
  });

  async function submitTokenForm() {
    let id = document.getElementById("id").value;
    let login = document.getElementById("login").value;
    let workstation = document.getElementById("workstation").value;
    let exp = document.getElementById("exp").value;
    let startwork = document.getElementById("startwork").value;
    let endwork = document.getElementById("endwork").value;

    let jobtitle = document.getElementById("jobtitle").value;
    let data = {
      id: id,
      login: login,
      exp: exp,
      startwork: startwork,
      endwork: endwork,
      jobtitle: jobtitle,
      workstation: workstation,
    };
    let response = await fetch("http://localhost:5500/signToken", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
```

```
          body: JSON.stringify(data),
        });

        if (response.ok) {
          let result = await response.json();
          let token = result.token;

          console.log("Created Token:", token);

          let blob = new Blob([token], { type: "text/plain" });
          let url = URL.createObjectURL(blob);
          let a = document.createElement("a");
          a.href = url;
          a.download = "token.txt";
          a.click();
          URL.revokeObjectURL(url);
        } else {
          console.error("Помилка при створенні токена:", await response.text());
        }
      }
    </script>
  </body>
</html>
```

Додаток Б

Текст застосунку. Модуль сервера

server.js

```
const sql = require("mssql/msnodesqlv8");
const jwt = require("jsonwebtoken");
const express = require("express");
const fs = require("fs");
const crypto = require("crypto");
const cors = require("cors");
const os = require("os");
const app = express();
app.use(express.json());
require("dotenv").config();
app.use(express.json());
app.use(cors());
const validateInput = (input, regex) => regex.test(input);
let globallogin = "";
app.use(
  cors({
    origin: "http://127.0.0.1:5500",
    methods: "POST, GET, OPTIONS, PUT, DELETE",
    allowedHeaders:
      "Origin, application/json, X-Requested-With, Content-Type, Accept, Authorization",
  })
);
let config = {
  server: process.env.DB_SERVER,
  database: process.env.DB_DATABASE,
  driver: process.env.DB_DRIVER,
  options: {
    trustedConnection: true,
    useUTC: true,
  },
  // connectionTimeout: 15000,
};

sql.connect(config, function (err) {
  if (err) {
    console.log("Database connection error:", err);
    return;
  }
  console.log("Connected to the database!");
});
async function getPrivateKey() {
  try {
    const privateKey = fs.readFileSync(process.env.PRIVATE_KEY_PATH, "utf8");
    return privateKey;
  } catch (err) {
    console.error("Error reading private key file:", err);
    throw err;
  }
}
async function getPublicKey() {
  try {
    const publicKey = fs.readFileSync(process.env.PUBLIC_KEY_PATH, "utf8");
    return publicKey;
  } catch (err) {
    console.error("Error reading public key file:", err);
    throw err;
  }
}
async function executeQuery(query, params = []) {
  try {
```

```
    let pool = await sql.connect(config);
    let request = pool.request();
    params.forEach((param) => {
      request.input(param.name, param.type, param.value);
    });
    let result = await request.query(query);
    return result.recordset;
  } catch (err) {
    console.error("Error executing query:", err);
    throw err;
  }
}
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send("Something broke!");
});

async function getPermForTableUsers(login) {
  try {
    let userPermissions = await getUserPermissions(login);

    if (!userPermissions) {
      throw new Error("Permissions not found for 'Head of Department'");
    }
    let roles = Object.keys(userPermissions);
    let hasNull = false;
    let hasAllow = false;
    for (let role of roles) {
      let permissions = userPermissions[role];
      let userManagementAction = permissions
        ? permissions.UserManagementActions
        : null;

      if (userManagementAction == 0) {
        return false;
        заборонена;
      }
      if (userManagementAction == null) {
        hasNull = true;
      }
      if (userManagementAction == 1) {
        hasAllow = true;
      }
    }
    return hasAllow && (!hasNull || hasNull);
  } catch (err) {
    console.error("Error getting permissions for table Users:", err);
    throw new Error("Error getting permissions for table Users");
  }
}
async function saveTokenToDatabase(token, payload) {
  try {
    let hash = await sha256(token);
    let expDate = new Date(payload.exp);
    let workHours = `${payload.startwork}-${payload.endwork}`;
    let params = [
      { name: "id", type: sql.BigInt, value: payload.id },
      { name: "hash", type: sql.VarChar, value: hash },
      { name: "jwt", type: sql.VarChar, value: token },
      { name: "workstation", type: sql.VarChar, value: payload.workstation },
      { name: "exptime", type: sql.Date, value: expDate },
      { name: "login", type: sql.VarChar, value: payload.login },
      { name: "workhours", type: sql.VarChar, value: workHours },
    ];
    await executeQuery(
      "INSERT INTO Tokens (ID, Hash, JWT, WorkStation, ExpirationTime, WorkHours, Login)
VALUES (@id, @hash, @jwt, @workstation, @exptime, @workhours, @login)",
      params
    );
    console.log("Token saved to database:", token);
```

```
  } catch (err) {
    console.error("Error saving token to database:", err);
    throw err;
  }
}
app.post("/signToken", async (req, res) => {
  let payload = req.body;
  let permission = await getPermForTableToken(globallogin);
  if (!permission) {
    return res.status(403).send("Access denied: insufficient permissions.");
  }

  try {
    let expDate = new Date(payload.exp);
    let expSeconds = Math.floor(expDate.getTime() / 1000);

    let newPayload = {
      ...payload,
      exp: expSeconds,
    };
    let privateKey = await getPrivateKey();
    let token = jwt.sign(newPayload, privateKey, { algorithm: "ES256" });
    if (token.length > 8000) {
      throw new Error("Token length exceeds the maximum limit.");
    }
    await saveTokenToDatabase(token, payload);

    res.json({ token });
  } catch (err) {
    console.error("Error signing token:", err);
    res.status(500).json({ error: "Error signing token." });
  }
});

app.post("/InsertUpdateDeleteUsers", async (req, res) => {
  let {
    action,
    login,
    password,
    fullname,
    organization,
    position,
    numberlicense,
  } = req.body;
  let permission = await getPermForTableUsers(globallogin);
  if (!permission) {
    return res.status(403).send("Access denied: insufficient permissions.");
  }
  if (!validateInput(login, /^\w+$/)) {
    res.status(400).send("Invalid login");
    return;
  }

  if (
    password &&
    !validateInput(password, /^[a-zA-Z0-9!@#$%^&*()_+={}\[\]:;"'<>,.?\/\\|-]+$/)
  ) {
    res.status(400).send("Invalid password");
    return;
  }
  if (!validateInput(fullname, /^[a-zA-ZA-Яа-яЁёЇїІіЄєҐґ\s]+$/)) {
    res.status(400).send("Invalid fullname");
    return;
  }
  if (!validateInput(organization, /^[a-zA-Z0-9A-Яа-яЁёЇїІіЄєҐґ\s.,-]+$/)) {
    res.status(400).send("Invalid organization");
    return;
  }
  if (!validateInput(position, /^[a-zA-ZA-Яа-яЁёЇїІіЄєҐґ\s .,-]+$/)) {
    res.status(400).send("Invalid position");
```

```
    return;
  }
  if (numberlicense && !validateInput(numberlicense, /^\d+$/)) {
    res.status(400).send("Invalid numberlicense");
    return;
  }

  let hashedPassword = await sha256(password);
  let params = [
    { name: "login", type: sql.VarChar, value: login },
    { name: "password", type: sql.VarChar, value: hashedPassword },
    { name: "fullname", type: sql.VarChar, value: fullname },
    { name: "organization", type: sql.VarChar, value: organization },
    { name: "position", type: sql.VarChar, value: position },
    { name: "numberlicense", type: sql.BigInt, value: numberlicense },
  ];
  try {
    if (action === "submit") {
      await executeQuery(
        `INSERT INTO Users (Login, FullName, Organization, Position, NumberLicense,
Password) VALUES (@login, @fullname, @organization, @position, @numberlicense, @password)`,
        params
      );
      console.log("Received data for saving to the database:", req.body);
      res.send("Data successfully saved to the database.");
    } else if (action === "edit") {
      await executeQuery(
        `
        UPDATE Users
        SET FullName = @fullname,
            Organization = @organization,
            Position = @position,
            NumberLicense = @numberlicense
        WHERE Login = @login
        `,
        params
      );
      console.log("Data was received to update the database:", req.body);
      res.send("Data successfully updated in the database.");
    } else if (action === "delete") {
      await executeQuery(`DELETE FROM Tokens WHERE login=@login`, params);
      await executeQuery(
        `DELETE FROM UsersAndRoles WHERE login=@login`,
        params
      );
      await executeQuery(`DELETE FROM Users WHERE login=@login`, params);
      console.log("Received data to be deleted in the database:", req.body);
      res.send("Data successfully deleted in the database.");
    } else {
      throw new Error("Invalid action specified.");
    }
  } catch (err) {
    console.error("Error saving/updating/deleting user data:", err);
    res
      .status(500)
      .send("Error saving/updating/deleting user data in the database.");
  }
});
app.post("/delTokeninBD", async (req, res) => {
  let { hash, jwt, workstation, exptime, login } = req.body;
  let params = [
    { name: "hash", type: sql.VarChar, value: hash },
    { name: "jwt", type: sql.VarChar, value: jwt },
    { name: "workstation", type: sql.VarChar, value: workstation },
    { name: "exptime", type: sql.Date, value: exptime },
    { name: "login", type: sql.VarChar, value: login },
  ];
  let permission = await getPermForTableToken(globallogin);
  if (!permission) {
    return res.status(403).send("Access denied: insufficient permissions.");
```

```
  }
  try {
    await executeQuery(`DELETE FROM Tokens WHERE Hash=@hash`, params);
    console.log("Received data to be delete in the database:", req.body);
    res.send("Data successfully delete in the database.");
  } catch (err) {
    console.error("Error deleting token data:", err);
    res.status(500).send("Error deleting token data in the database.");
  }
});
app.post("/authenticate", async (req, res) => {
  let { login, pass, token } = req.body;
  let publicKey = await getPublicKey();
  globallogin = login;

  if (
    !validateInput(login, /^\w+$/) ||
    !validateInput(pass, /^[a-zA-Z0-9!@#$%^&*()_+={}\[\]:;"'<>,.?\/\\|-]+$/)
  ) {
    return res.status(400).send("Invalid login or password");
  }
  if (!token) {
    res.json({
      authenticated: false,
      error: "Токен обов'язковий для автентифікації",
    });
    return;
  }

  try {
    let res_log = await executeQuery(
      "SELECT * FROM Users WHERE Login = @login",
      [{ name: "login", type: sql.VarChar, value: login }]
    );

    if (res_log.length == 0) {
      res.json({ authenticated: false, error: "User not found" });
      return;
    }

    let user = res_log[0];
    // console.log("login :>> ", user.Login);

    let hashedPassword = await sha256(pass);
    if (hashedPassword !== user.Password) {
      res.json({ authenticated: false, error: "Invalid password" });
      return;
    }

    try {
      let req_db_token = await executeQuery(
        "SELECT * FROM Tokens WHERE Login = @login AND JWT = @jwt",
        [
          { name: "login", type: sql.VarChar, value: login },
          { name: "jwt", type: sql.VarChar, value: token },
        ]
      );
      let dbToken = req_db_token[0];

      let decodedToken = jwt.verify(token, publicKey, {
        algorithms: ["ES256"],
      });
      let tokenUser = decodedToken.login;
      let expTime = decodedToken.exp;
      let id = decodedToken.id;
      if (id != dbToken.ID) {
        res.json({ authenticated: false, error: "ID does not match user" });
        return;
      }
```

```
      if (tokenUser !== login) {
        res.json({ authenticated: false, error: "Token does not match user" });
        return;
      }

      let currentTime = Math.floor(Date.now() / 1000);

      if (currentTime > expTime) {
        res.json({ authenticated: false, error: "Token has expired" });
        return;
      }

      let workstation = decodedToken.workstation;
      let dbTokenWorkstation = dbToken.WorkStation;
      let macAddress = getMacAddress();
      if (workstation !== dbTokenWorkstation || workstation !== macAddress) {
        res.json({ authenticated: false, error: "Invalid workstation" });
        return;
      }

      let startWorkHours = parseInt(decodedToken.startwork.split(":")[0]);
      let startWorkMinutes = parseInt(decodedToken.startwork.split(":")[1]);
      let endWorkHours = parseInt(decodedToken.endwork.split(":")[0]);
      let endWorkMinutes = parseInt(decodedToken.endwork.split(":")[1]);

      const startWork = new Date();
      startWork.setHours(startWorkHours, startWorkMinutes, 0);
      const endWork = new Date();
      endWork.setHours(endWorkHours, endWorkMinutes, 0);

      const now = new Date();
      if (now < startWork || now > endWork) {
        res.json({ authenticated: false, error: "Not within work hours" });
        return;
      }
      let hashedToken = await sha256(token);
      if (hashedToken !== dbToken.Hash) {
        res.json({ authenticated: false, error: "Invalid token hash" });
        return;
      }

      res.json({ authenticated: true });
    } catch (err) {
      console.error("Error verifying token:", err);
      res
        .status(500)
        .json({ authenticated: false, error: "Error verifying token" });
    }
  } catch (err) {
    console.error("Error authenticating user:", err);
    res
      .status(500)
      .json({ authenticated: false, error: "Error authenticating user" });
  }
});
const PORT = process.env.PORT;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

Додаток В

Результати інтеграційного тестування

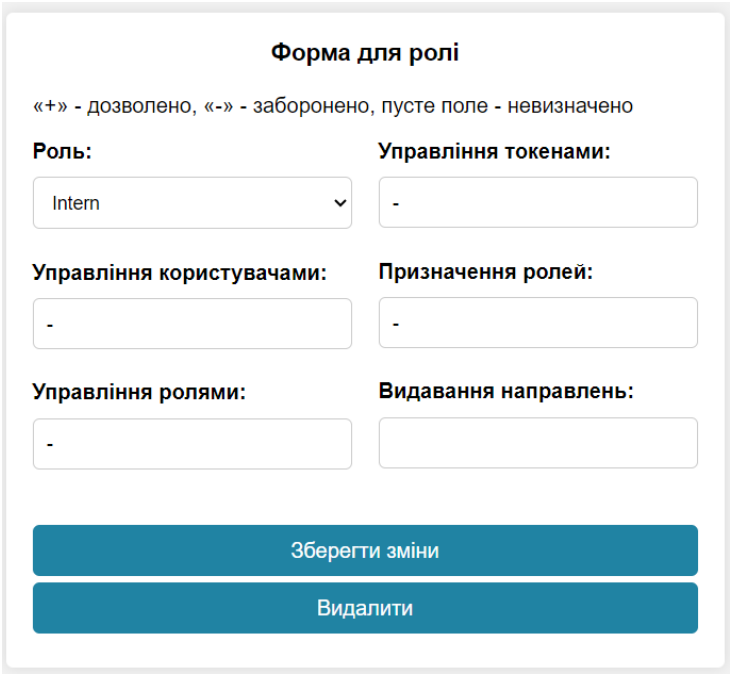


Рисунок В.1 – Заповнена форма для створення ролі

4	Intern	0	0	0	0	NULL

Рисунок В.2 – Вміст бази даних внаслідок виконання запиту

```
Received data for saving to the database: {
  action: 'submit',
  role: 'Intern',
  permissions: {
    UserManagementActions: 0,
    TokenManagementActions: 0,
    RoleManagementActions: 0,
    RoleAssignment: 0,
    IssuingReferrals: null
  }
}
```

Рисунок В.3 – Результат виконання запиту на сервері

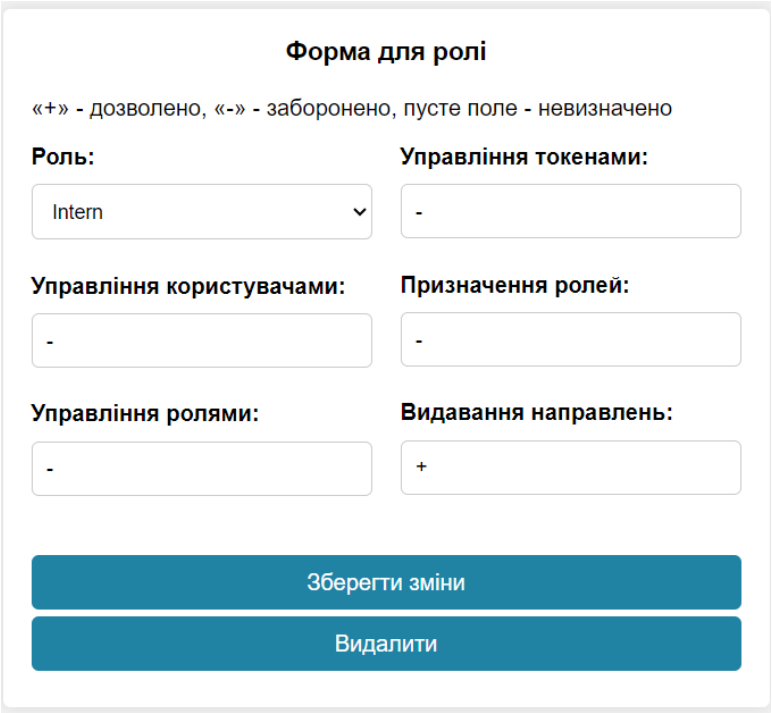


Рисунок В.4 – Заповнена форма для оновлення ролі

4	Intern	0	0	0	0	1

Рисунок В.5 – Результат виконання запиту в БД

	Role	UserManagementActions	TokenManagementActions	RoleManagementActions	RoleAssignment	IssuingReferrals
1	Administrator	1	1	1	1	NULL
2	Family doctor	NULL	NULL	NULL	NULL	1
3	Head of Department	1	1	0	1	NULL
4	Nurse	0	0	0	0	1

Рисунок В.6 – Результат виконання запиту в БД при видаленні ролі

```
Received data to be deleted in the database: {
  action: 'delete',
  role: 'Intern',
  permissions: {
    UserManagementActions: 0,
    TokenManagementActions: 0,
    RoleManagementActions: 0,
    RoleAssignment: 0,
    IssuingReferrals: 1
  }
}
```

Рисунок В.7 – Результат виконання запиту на сервері при видаленні ролі

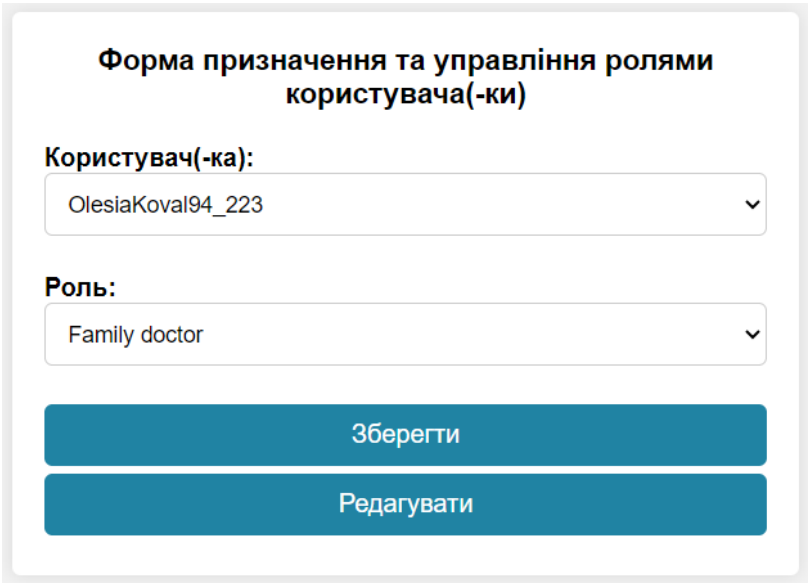


Рисунок В.8 – Заповнена форма для призначення ролі користувачу

7	1017	OlesiaKoval94_223	Family doctor

Рисунок В.9 – Вміст таблиці «UsersAndRoles» внаслідок виконання запиту

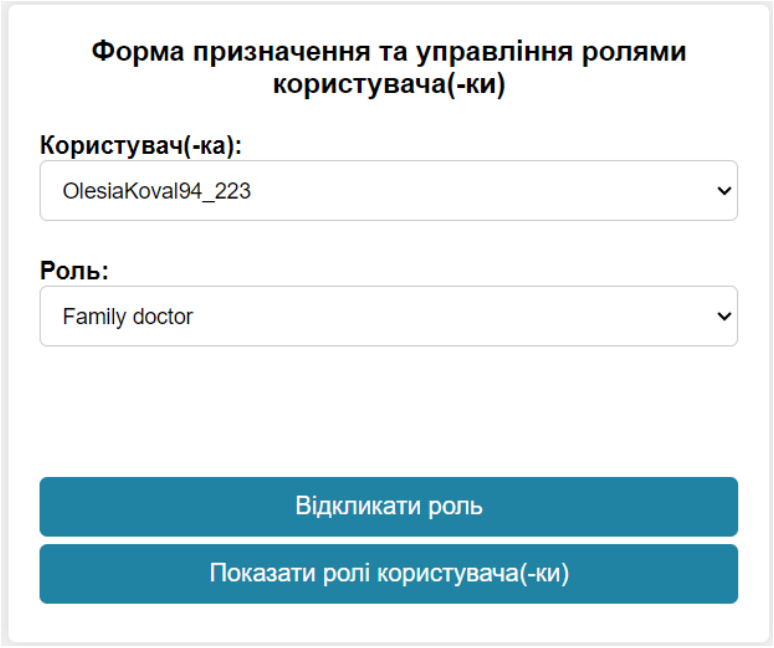


Рисунок В.10 – Заповнена форма з обраними користувачем та роллю, яку потрібно відкликати

Results Messages

	ID	Login	Role
1	1013	admin	Administrator
2	1014	AnnaKoval99_0	Family doctor
3	1015	AnnaKoval99_0	Head of Department

Рисунок В.11 – Вміст бази даних внаслідок виконання запиту

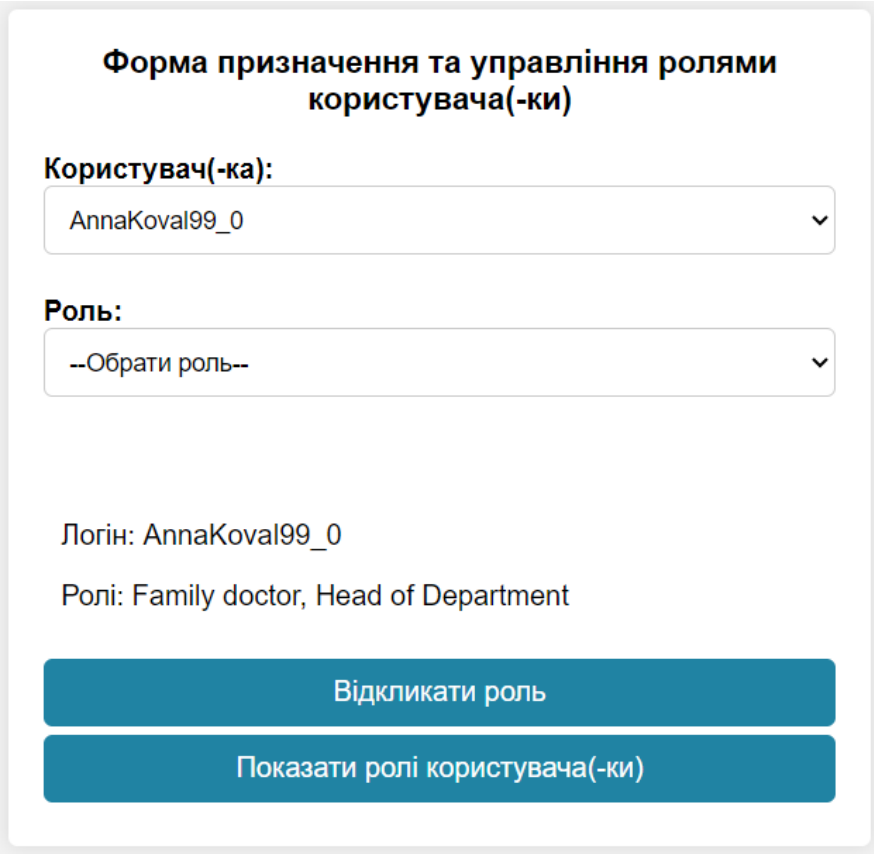


Рисунок В.12 – Результат внаслідок натискання на кнопку «Показати ролі користувача(-ки)»

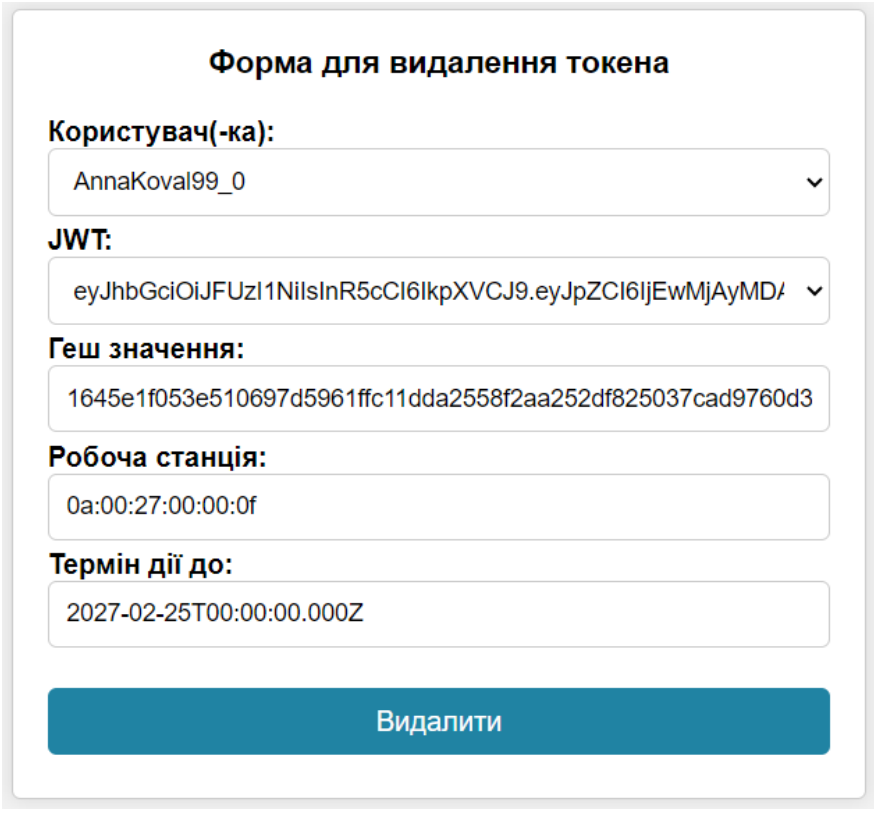


Рисунок В.13 – Заповнена форма для видалення токена користувача з БД

	ID	Hash	JWT	WorkStation	ExpirationTime	Login	WorkHours
1	5242028	9b22bcba064dda77e2...	eyJhbGciOiJFUzI1NiIsInR5...	0a:00:27:00:00:0f	2029-05-25	admin	00:00-23:59

Рисунок В.14 – Вміст бази даних внаслідок виконання запиту

```
Received data to be delete in the database: {
  hash: '1645e1f053e510697d5961ffc11dda2558f2aa252df825037cad9760d3c0a68f',
  token: 'eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXVCJ9.
  eyJpZCI6IjEwMjAyMDAiLCJsb2dpbiI6IkFubmFLb3ZhbDk5XzAiLCJleHAiOjE4MDM1MTM2MDAsInN0YXJ0d29yayI6IjA
  5INC70ZbQutCw0YAsINCX0LDQstGW0LTRg9Cy0LDRhyDQsNC80LHRg9C70LDRgtC-0YDRltGXIiwid29ya3N0YXRpb24iOi
  3AY9GLnMqcH_nzJPO-IEZJG_QwZqaDh61MkTJdV_mT2YHp6r5zT3vdNy3sn_cnO8ltvR1qDMHtoUClmdfPb2yg',
  workstation: '0a:00:27:00:00:0f',
  exptime: '2027-02-25T00:00:00.000Z',
  login: 'AnnaKoval99_0'
}
```

Рисунок В.15 – Результат видалення токена на сервері

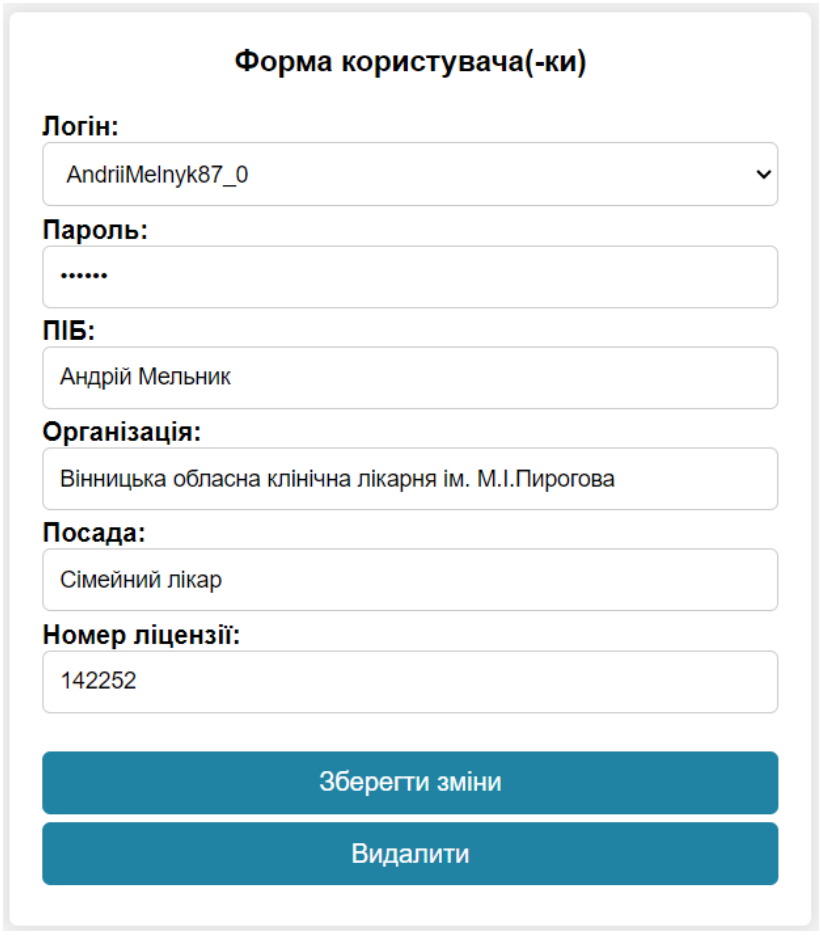


Рисунок В.16 – Результат автоматично заповненої форми при обранні користувача

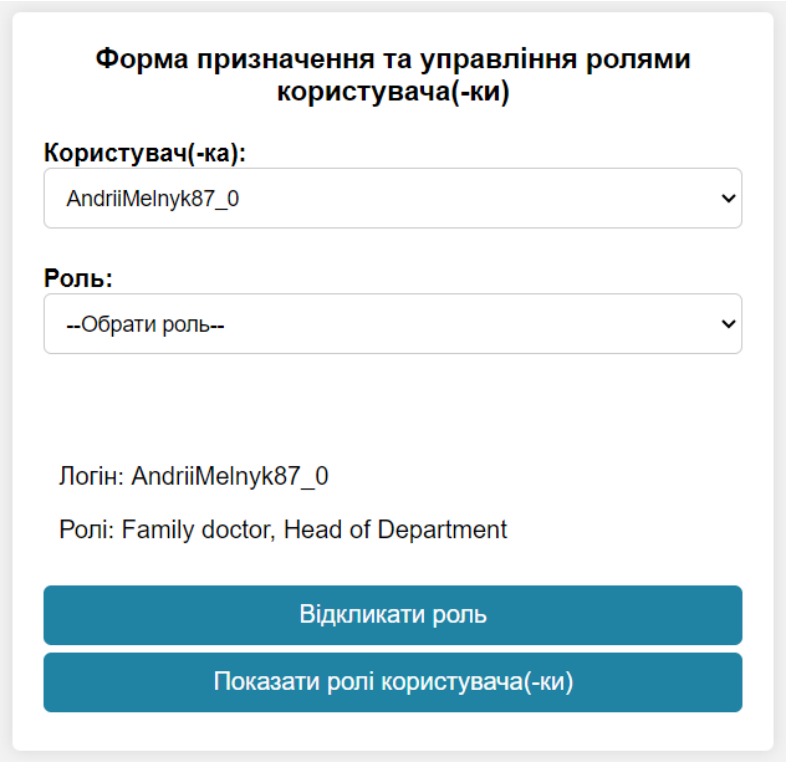


Рисунок В.17 –Заповнена форма для призначення ролі користувачу

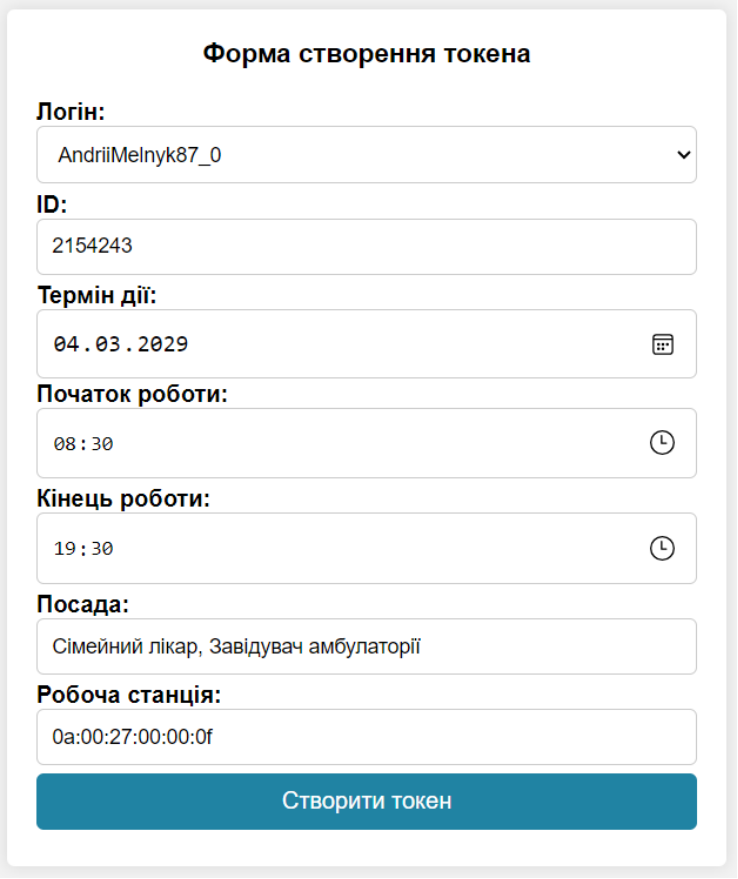


Рисунок В.18 – Заповнена форма для створення токена користувачу

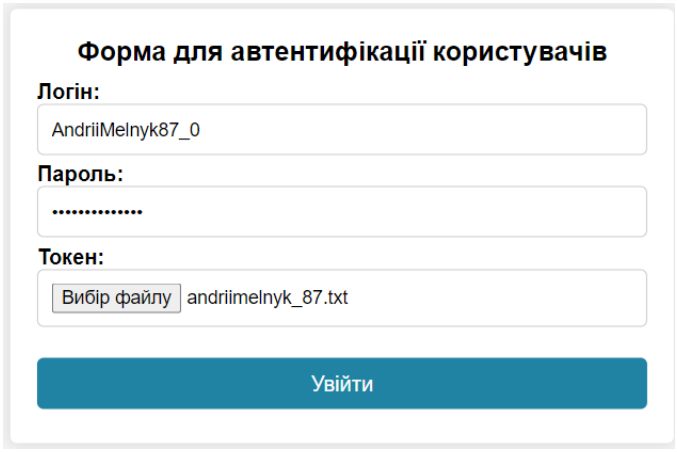


Рисунок В.19 – Заповнена форма для автентифікації користувача

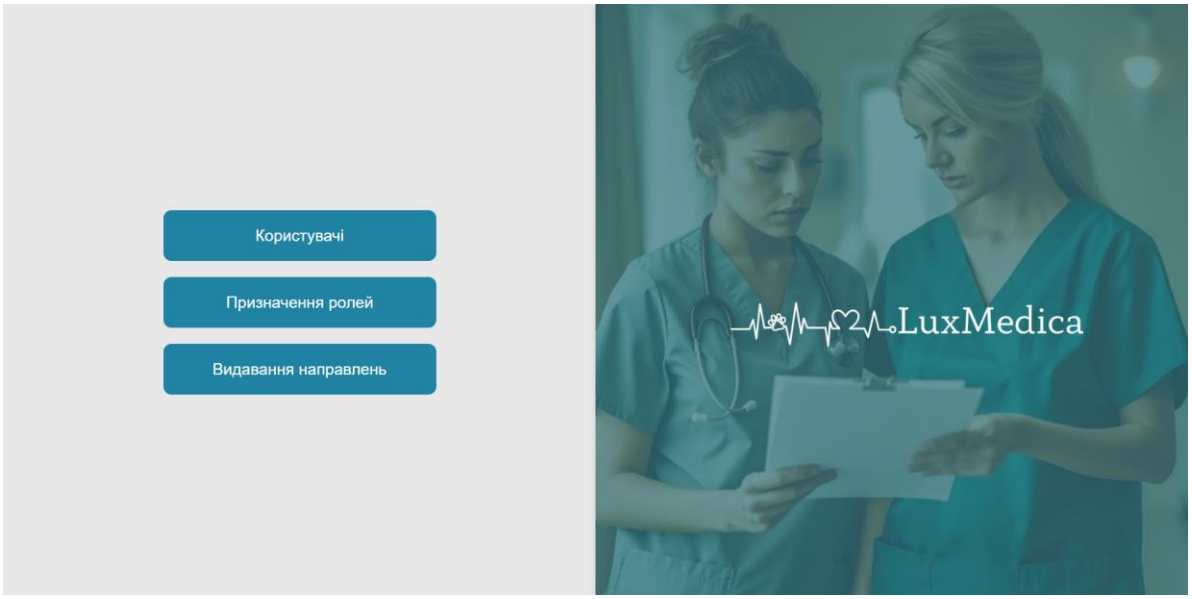


Рисунок В.20 – Головна сторінка з переліком доступних форм для користувача

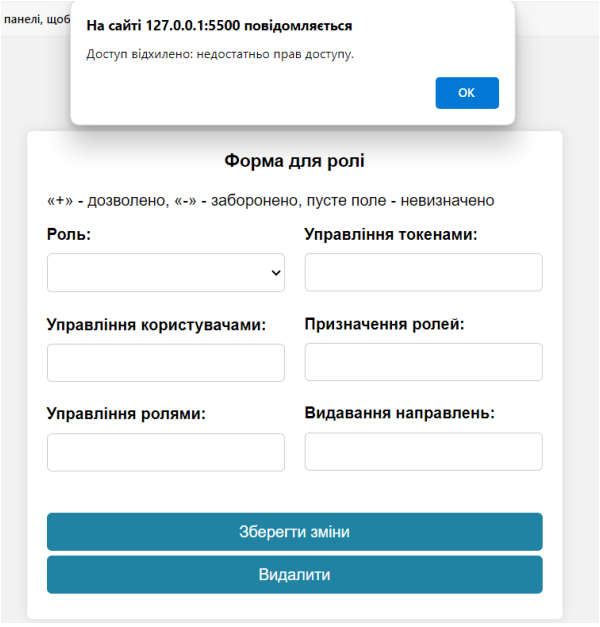


Рисунок В.21 – Діалогове вікно з повідомленням про відхилений доступ

Results | Messages

	Login	FullName	Organization	Position	NumberLicense	Password
1	admin	admin	Вінницька обласна клінічна лікарня ім. М.І.Пирог...	Адміністратор	NULL	8d969eef6ecad3c29a3a62928
2	AndriiMelnyk87_0	Андрій Мельник	Вінницька обласна клінічна лікарня ім. М.І.Пирог...	Сімейний лікар	142252	8d969eef6ecad3c29a3a62928
3	AnnaKoval99_0	Коваль Анна Сергіївна	Вінницька обласна клінічна лікарня ім. М.І.Пирог...	Сімейний лікар, Завідувач амбулаторії	6920212	8d969eef6ecad3c29a3a62928
4	OlesiaKoval94_223	Олеся Коваль	Вінницька обласна клінічна лікарня ім. М.І.Пирог...	Адміністраторка	NULL	8d969eef6ecad3c29a3a62928

Рисунок В.22 – Вміст таблиці «Users» в БД до видалення користувачки

Results | Messages

	ID	Login	Role
1	1013	admin	Administrator
2	1014	AnnaKoval99_0	Family doctor
3	1015	AnnaKoval99_0	Head of Department
4	1018	AndriiMelnyk87_0	Family doctor
5	1019	AndriiMelnyk87_0	Head of Department

Рисунок В.23 – Вміст таблиці «UsersAndRoles» в БД до видалення користувачки

Results | Messages

	ID	Hash	JWT	WorkStation	ExpirationTime	Login	WorkHours
1	2154243	41aa8e3101957a95df67f6154c...	eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXV...	0a:00:27:00:00:0f	2029-03-04	AndriiMelnyk87_0	08:30-19:30
2	5242028	9b22bcba064dda77e2e9f8ef4f...	eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXV...	0a:00:27:00:00:0f	2029-05-25	admin	00:00-23:59
3	5255125	a7e7b64be6a3d82008d09c05...	eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXV...	0b:00:30:12:00:0f	2030-06-29	AnnaKoval99_0	08:30-19:30
4	51451251	ac283bfda7415718af79c5e31c...	eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXV...	0a:00:27:00:00:0f	2029-12-05	AnnaKoval99_0	08:30-19:30

Рисунок В.24 – Вміст таблиці «Tokens» в БД до видалення користувачки

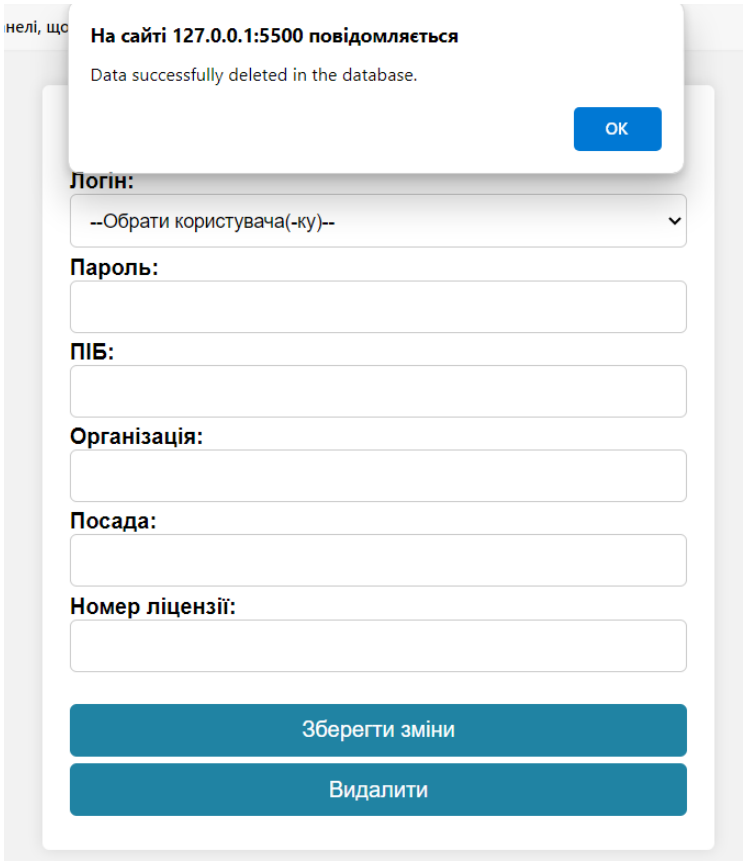


Рисунок В.25 – Діалогове вікно з повідомленням про успішне видалення користувачки

Results Messages

	Login	FullName	Organization	Position	NumberLicense	Password
1	admin	admin	Вінницька обласна клінічна лікарня ім. М.І.Пирог...	Адміністратор	NULL	8d969eef6ecad3c2
2	AndriiMelnyk87_0	Андрій Мельник	Вінницька обласна клінічна лікарня ім. М.І.Пирог...	Сімейний лікар	142252	8d969eef6ecad3c2
3	OlesiaKoval94_223	Олеся Коваль	Вінницька обласна клінічна лікарня ім. М.І.Пирог...	Адміністраторка	NULL	8d969eef6ecad3c2

Рисунок В.26 – Вміст таблиці «Users» в БД після видалення користувачки

Results Messages

	ID	Login	Role
1	1013	admin	Administrator
2	1018	AndriiMelnyk87_0	Family doctor
3	1019	AndriiMelnyk87_0	Head of Department

Рисунок В.27 – Вміст таблиці «UsersAndRoles» в БД після видалення користувачки

Results Messages

	ID	Hash	JWT	WorkStation	ExpirationTime	Login	WorkHours
1	2154243	41aa8e3101957a95df67f6154c...	eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXV...	0a:00:27:00:00:0f	2029-03-04	AndriiMelnyk87_0	08:30-19:30
2	5242028	9b22bcba064dda77e2e9f8ef4f0...	eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXV...	0a:00:27:00:00:0f	2029-05-25	admin	00:00-23:59

Рисунок В.28 – Вміст таблиці «Tokens» в БД після видалення користувачки

Додаток Г

Протокол перевірки бакалаврської дипломної роботи на наявність текстових запозичень

Назва роботи: Автоматизована система видавання направлень на обстеження в медичних закладах. Частина 2. Модуль розмежування прав доступу користувачів

Автор роботи: Клиш Вікторія Миколаївна

Тип роботи: бакалаврська дипломна робота (БДР, МКР).

Підрозділ кафедра захисту інформації ФІТКІ (кафедра, факультет).

Показники звіту подібності Unicheck

Оригінальність – 98,09%. Схожість – 1,91%.

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

☐ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

☐ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку (підпис) Валентина КАПЛУН (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи (підпис) Вікторія КЛИШ (прізвище, ініціали)

Керівник роботи (підпис) Юрій БАРИШЕВ (прізвище, ініціали)

Додаток Д

ІЛЮСТРАТИВНА ЧАСТИНА

АВТОМАТИЗОВАНА СИСТЕМА ВИДАВАННЯ НАПРАВЛЕНЬ НА ОБСТЕЖЕННЯ В МЕДИЧНИХ ЗАКЛАДАХ. ЧАСТИНА 2. МОДУЛЬ РОЗМЕЖУВАННЯ ПРАВ ДОСТУПУ КОРИСТУВАЧІВ

(Назва бакалаврської дипломної роботи)

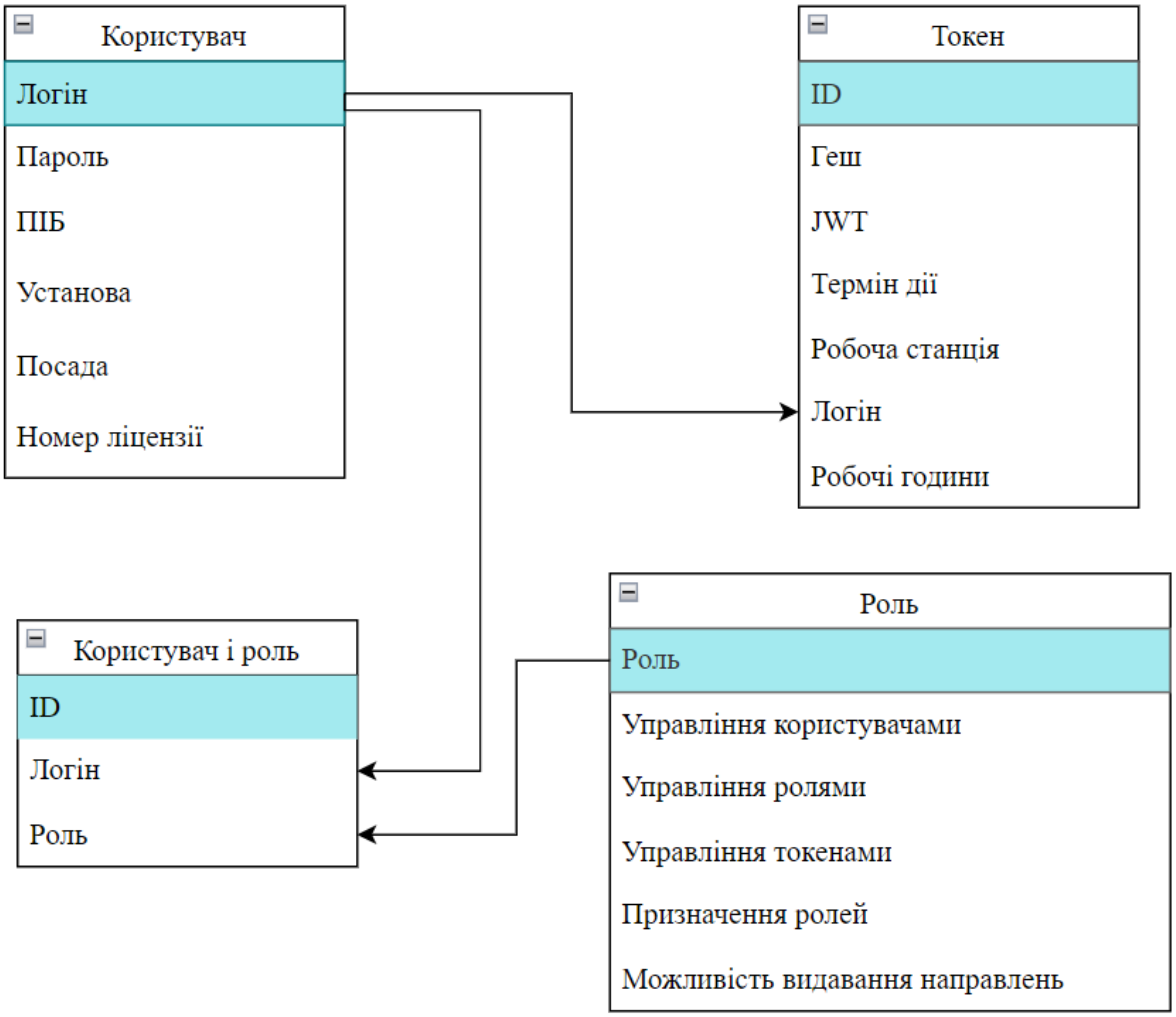
Виконала: студентка 4 курсу групи 1БС-20б
спеціальності 125 Кібербезпека
_____ Вікторія КЛИШ
14 червня 2024 р.

Керівник: к. т. н., доцент каф. ЗІ
_____ Юрій БАРИШЕВ
14 червня 2024 р.

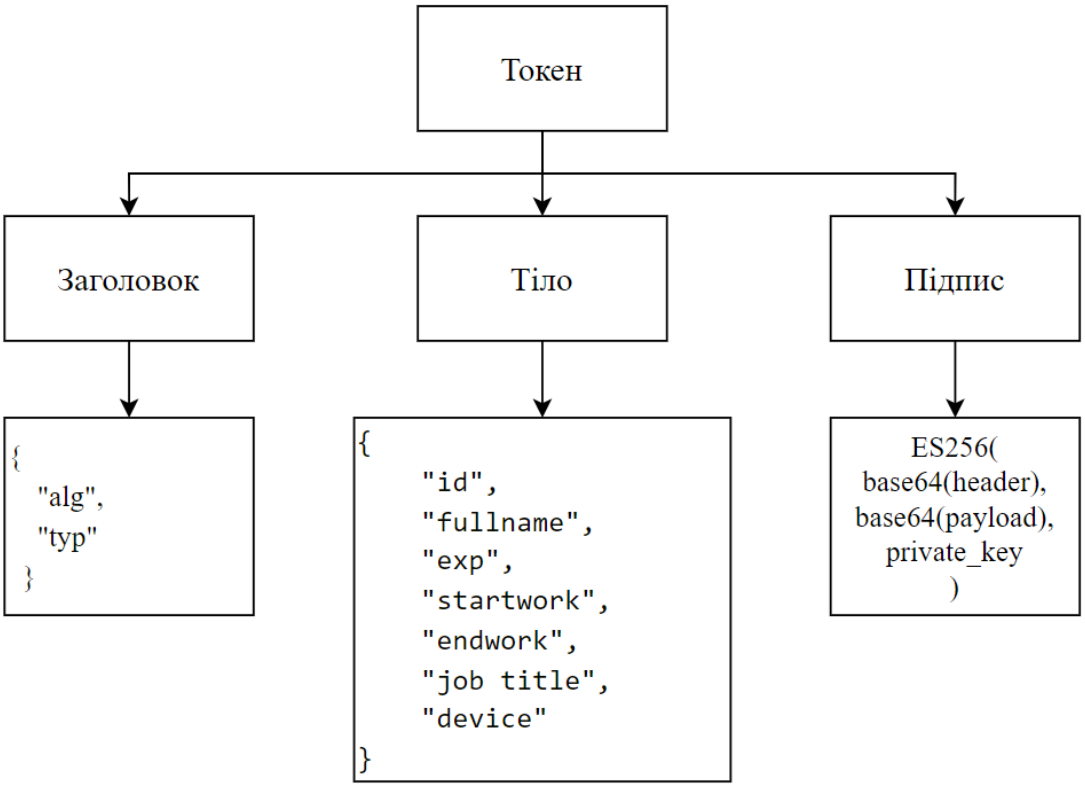
ПОРІВНЯЛЬНИЙ АНАЛІЗ МОДЕЛЕЙ РОЗМЕЖУВАННЯ ПРАВ ДОСТУПУ

Вимоги	HRU	Bell-LaPadula	Biba	RBAC	ABAC	OrBAC
Конфіденційність	Висока	Висока	Низька	Середня	Висока	Середня
Цілісність	Середня	Середня	Висока	Середня	Середня	Середня
Гнучкість	Середня	Низька	Низька	Висока	Висока	Висока
Протидія внесенню дезінформації	Середня	Висока	Середня	Висока	Висока	Середня

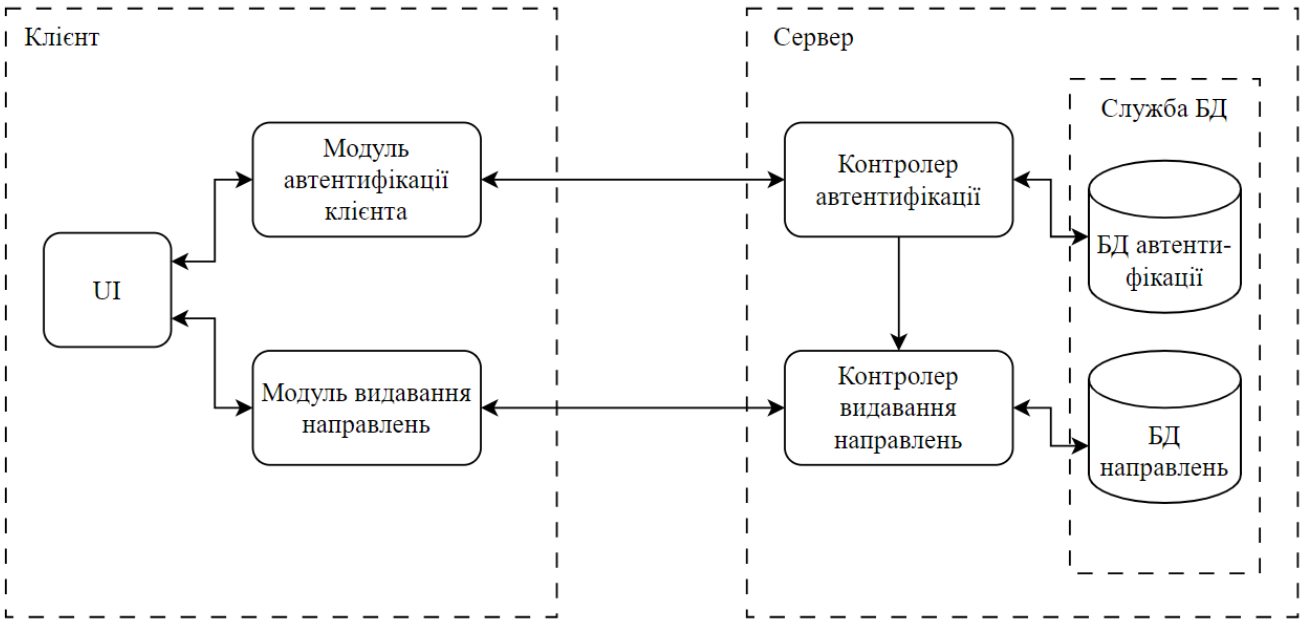
МОДЕЛЬ БАЗИ ДАНИХ



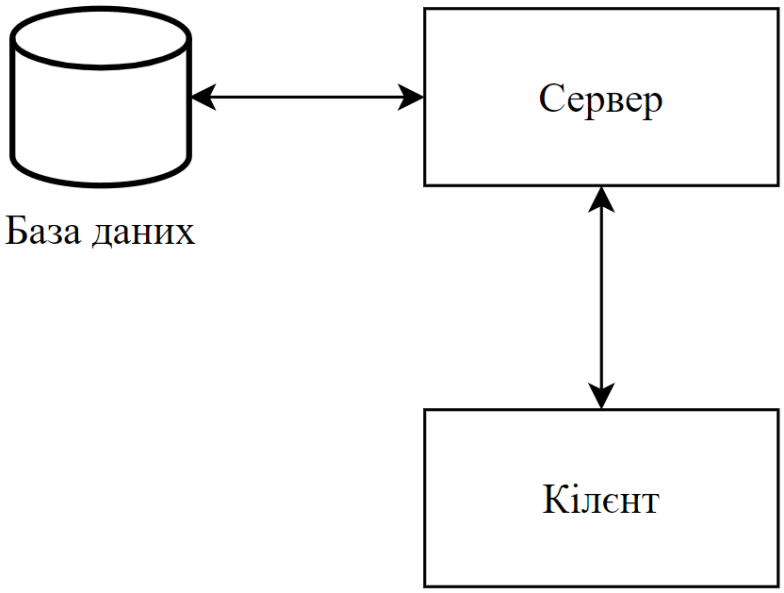
СТРУКТУРА ТОКЕНА



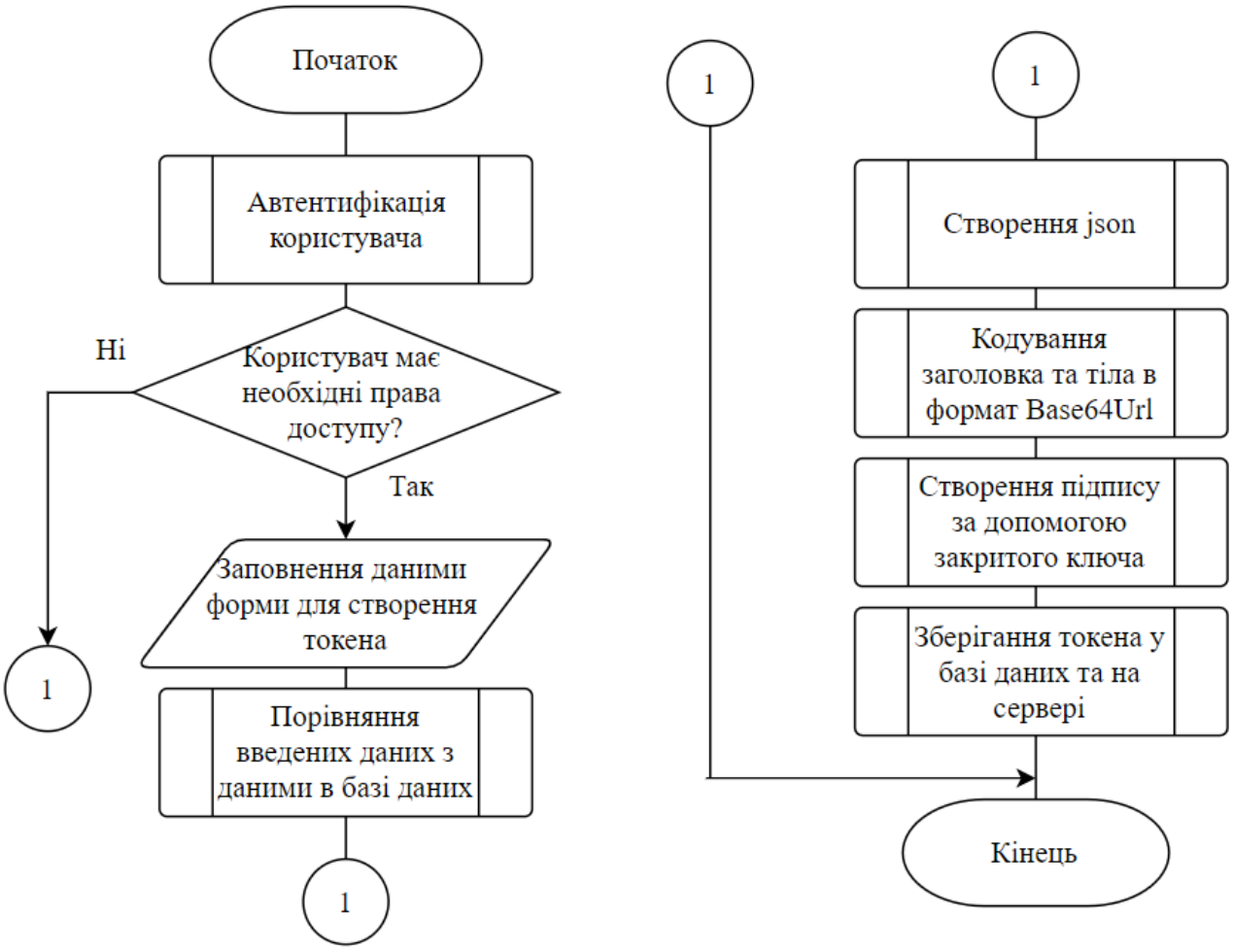
УЗАГАЛЬНЕНА АРХІТЕКТУРА ПРОГРАМИ



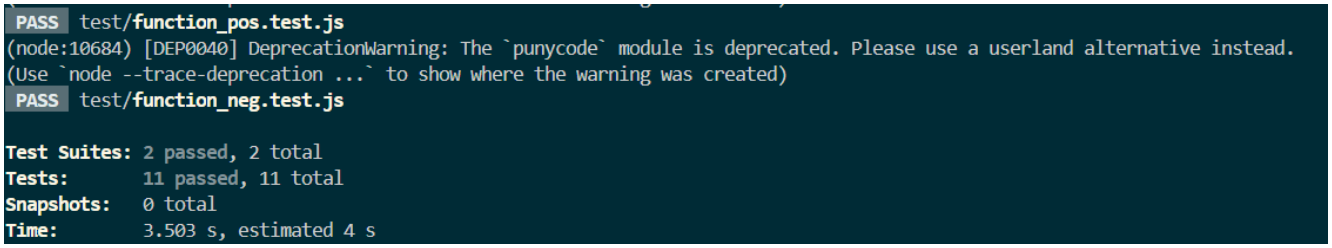
АЛГОРИТМ РОБОТИ МОДУЛЯ



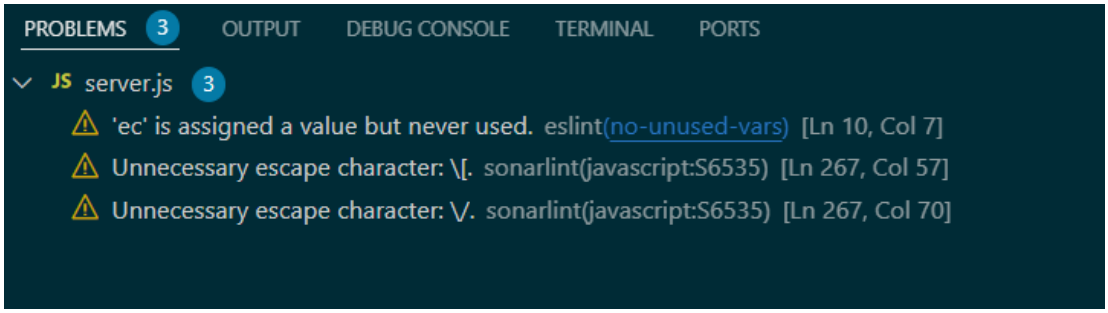
АЛГОРИТМ СТВОРЕННЯ ТОКЕНА



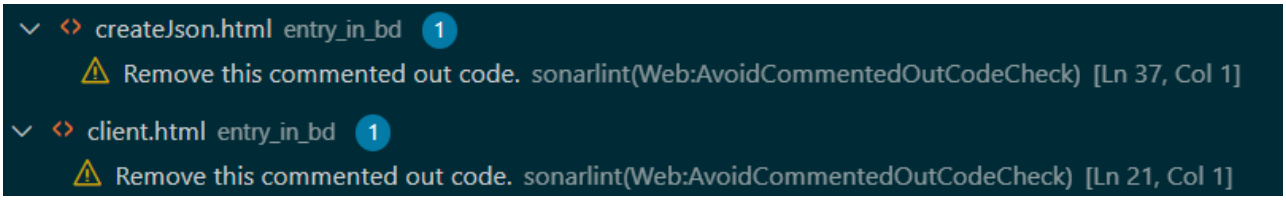
РЕЗУЛЬТАТ БЛОКОВОГО ТЕСТУВАННЯ



РЕЗУЛЬТАТИ СТАТИЧНОГО ТЕСТУВАННЯ БЕЗПЕКИ



Результати тестування за допомогою SonarLint



Результати тестування за допомогою HTMLHint

```
PS E:\Projects\Node_Js\AccessControl_diploma> npm run lint

> lint
> eslint . --config eslint.config.mjs --format ./eslint-reporter.mjs

Checked: E:\Projects\Node_Js\AccessControl_diploma\FormAuthentication.js - No issues found
Checked: E:\Projects\Node_Js\AccessControl_diploma\entry_in_bd\script\RolesForm.js - No issues found
Checked: E:\Projects\Node_Js\AccessControl_diploma\entry_in_bd\script\TokenForm.js - No issues found
Checked: E:\Projects\Node_Js\AccessControl_diploma\entry_in_bd\script\UsersAndRoles.js - No issues found
Checked: E:\Projects\Node_Js\AccessControl_diploma\entry_in_bd\script\client.js - No issues found
Checked: E:\Projects\Node_Js\AccessControl_diploma\server.js - No issues found

All files passed linting successfully!
```

Результати тестування за допомогою ESLint