МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»

Освітньо-професійна програма: «Безпека комп'ютерних систем і мереж»

Група: 4КБ-02

Дипломний проект

здобувача освіти денної форми навчання
КБ.02.25.000.ДП

ЯРОШЕНКО
АЛІНИ АНАТОЛІЇВНИ

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітньо-професійна програма: «**Безпека комп'ютерних систем і мереж»**

Група: **4КБ-02**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту на тему:

Розробка web-застосунку для аутентифікації користувачів з використанням методів криптографії

Проектний матеріал складається з пояснювальної записки на 90 сторінках та графічного (презентаційного) матеріалу на 12 аркушах (слайдах)

Дипломник _____ (Ярошенко А.А.)

Керівник _____ (Гаджиєв М.М.)

Консультанти:

з економічного розділу _____ (Канський М.Ю.)

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Краснокутська К.Г.)

Захист «26» червня 2025 р. Протокол ЕК № 5

Оцінка ЕК 5 (відмінно) / 90 б.

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Безпека комп'ютерних систем і мереж»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"19" 05 2025 р.

ЗАВДАННЯ

на дипломний проект

Здобувачеві освіти *Ярошенко Аліні Анатоліївні*

(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка web-застосунку для аутентифікації користувачів з використанням методів криптографії*

затверджена наказом по коледжу від "14" листопада 2024р. № 246

2. Термін здачі закінченого проекту 16 червня, 2025 р.

3. Вихідні данні до проекту *1. Реалізувати web-застосунок для безпечної реєстрації та авторизації користувачів із перевіркою надійності пароля; 2. Використати криптографічні алгоритми для шифрування даних; 3. Забезпечити токен-базовану аутентифікацію із застосуванням JWT; 4. Реалізувати хешування паролів для подальшого зберігання; 5. Реалізувати заходи захисту від поширених атак на систему аутентифікації, таких як XSS, CSRF та MITM; 6. Забезпечити механізм перевірки профілю користувача після здійснення входу;*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)
Аналіз наявних методів аутентифікації та криптографічного захисту даних; Проєктування архітектури системи; Розробка серверної та клієнтської частини web-застосунку для аутентифікації користувачів із використанням сучасних методів криптографії; Проведення тестування захищеності системи; Аналіз отриманих результатів; Економічний розділ; Розділ охорони праці і техніки безпеки

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)
Блок-схема алгоритму реєстрації; Блок-схема алгоритму входу; Блок-схема алгоритму видалення користувача; Блок-схема алгоритму реєстрації на серверній частині; Блок-схема алгоритму входу на серверній частині; Блок-схема алгоритму видалення користувача на серверній частині; Схема атаки розпиленням паролів; Схема атаки підробкою міжсайтових запитів; Схема атаки людини посередині; Схема атаки XSS; Розробка інтерфейсу застосунка; Схема клієнт-серверної архітектури

6. Консультанти по проекту, із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Гаджиєв М.М.*		
Економічний розділ	Канський М.Ю.		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання 12.05.25

Керівник *Гаджиєв М.М.* (підпис)

Завдання прийняв до виконання *Ярошенко А.А.* (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	*Вступ. Постановка задачі проєктування*	20.10.2024	Виконано
2	*Аналіз завдання та опис вимог до системи*	12.01.2025	Виконано
3	*Аналіз існуючих рішень для аутентифікації*	10.03.2025	Виконано
4	*Розробка архітектури застосунку*	19.03.2025	Виконано
5	*Розробка алгоритму роботи клієнтської частини застосунку*	23.03.2025	Виконано
6	*Розробка алгоритму роботи логіки серверної частини застосунку*	30.03.2025	Виконано
7	*Розробка web-інтерфейсу застосунку*	09.04.2025	Виконано
8	*Визначення програмних засобів розробки*	14.04.2025	Виконано
9	*Написання коду web-застосунку*	15.04.2025	Виконано
10	*Випробування застосунку та аналіз результатів*	20.04.2025	Виконано
11	*Оформлення пояснювальної записки*	12.05.2025	Виконано
12	*Виконання економічних розрахунків*	22.05.2025	Виконано
13	*Розробка питань з охорони праці та техніки безпеки*	25.05.2025	Виконано
14	*Підготовка мультимедійної презентації проекту*	02.06.2025	Виконано

Дипломник (підпис)

Керівник (підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			КБ 02. 25 000. 00 ДП	Дипломний проект		
А4			КБ 02. 25 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	КБ 02. 25 000. 00 ДП		
Розробив		Ярошенко А.А.		18.06	Розробка web-застосунку для аутентифікації користувачів з використанням методів криптографії	Літ.	Аркуш 4 / Аркушів 90
Перевірив		Гаджиєв М.М.		18.06		НДП	
Н. контр.		Петрашова В.І.		18.06		ВСП "ОТФК ОНТУ" гр.4КБ-02	
Затверд.		Кривченко Ю.В.		18.06			

ЗМІСТ

ВСТУП

У сучасному світі цифрових технологій, де більшість послуг перейшли в онлайн-середовище, питання захисту персональних даних користувачів стає надзвичайно важливим. Одним із ключових механізмів управління доступом до конфіденційної інформації є аутентифікація, і саме її надійність визначає рівень безпеки як для індивідуальних користувачів, так і для цілих інформаційних систем. Актуальність цієї теми обумовлена постійним зростанням кількості кібератак, спрямованих на отримання облікових даних. Уразливості в цьому контексті часто пояснюються недостатнім або застарілим захистом процесів аутентифікації. З огляду на це, розробка web-застосунку із застосуванням сучасних криптографічних методів є важливим кроком до підвищення рівня безпеки інформаційних систем.

Значущість цієї роботи полягає у практичному впровадженні криптографічних алгоритмів, таких як хешування паролів, а також в аналізі поширених атак на механізми аутентифікації і методів їхнього запобігання. Прикладна цінність проєкту полягає у створенні веб-застосунку, який дозволяє безпечно реєструвати та аутентифікувати користувачів та зберігати токени доступу.

Метою даної дипломної роботи є розробка безпечного веб-застосунку для аутентифікації користувачів із використанням сучасних методів криптографії. Для досягнення поставленої мети було виконано такі завдання: детальний аналіз наявних методів аутентифікації та криптографічного захисту даних; проєктування архітектури системи, враховуючи актуальні вимоги до безпеки; реалізація безпечної реєстрації та авторизації користувачів із перевіркою введених паролів; використання криптографічного хешування паролів на серверній стороні; забезпечення токен-базовану аутентифікацію із застосуванням JWT; реалізація заходів захисту від поширених атак на систему аутентифікації, таких як XSS, CSRF та MITM; проведення тестування веб-застосунку.

У першому розділі розглянуто теоретичні засади аутентифікації користувачів у web-середовищі, здійснено порівняльний аналіз методів аутентифікації, зокрема паролів, токенів доступу, а також обґрунтовано вибір криптографічного алгоритму RSA для захисту даних. Також у цьому розділі представлено архітектуру застосунку, описано основні компоненти серверної та клієнтської частин, механізми передачі та збереження даних, а також реалізацію заходів безпеки.

Другий розділ присвячений економічному обґрунтуванню доцільності впровадження розробленого веб-застосунку. У ньому наведено розрахунки витрат на розробку, впровадження та експлуатацію системи, а також визначено економічну ефективність розробки шляхом оцінки потенційної економії ресурсів при використанні web-застосунку у порівнянні з традиційними засобами ведення діяльності.

У третьому розділі розглянуто питання охорони праці, зокрема умови безпечної роботи користувача за комп’ютером. Проведено аналіз шкідливих та небезпечних чинників, що можуть впливати на працівника під час розробки або експлуатації програмного забезпечення, подано гігієнічні вимоги до приміщення, освітлення, рівня шуму та мікроклімату. Розкрито вимоги до електробезпеки, організації робочого місця, а також заходи протипожежного захисту в умовах використання комп’ютерної техніки.

Робота демонструє технічну реалізацію актуального та безпечного рішення в сфері web-розробки, обґрунтовує його економічну доцільність і забезпечення належного рівня безпеки та охорони праці при розробці й використанні подібних інформаційних систем.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналіз сучасного ринку у сфері web-застосунків

Сучасний ринок веб-застосунків є одним із найдинамічніших напрямів у сфері інформаційних технологій. Із зростанням чисельності онлайн-сервісів, електронної комерції, хмарних обчислень і мобільних платформ, важливість безпечної аутентифікації користувачів стала надзвичайно великою. Саме тому питання криптографічного захисту, зокрема розробка надійних систем авторизації та аутентифікації, вийшли на перший план у створенні веб-застосунків. Відзначається стабільне зростання попиту на безпечні рішення, що зумовлено збільшенням кількості кіберзагроз. У зв'язку з цим розробники все частіше впроваджують криптографічні методи як у клієнтську, так і у серверну логіку. Серед них – симетричне та асиметричне шифрування, цифрові підписи, токени доступу, а також механізми двофакторної та багатофакторної аутентифікації. Одним із ключових напрямів еволюції web-застосунків стало використання JSON Web Token (JWT) у поєднанні з асиметричним шифруванням (наприклад, RSA) або HMAC. Такий підхід забезпечує децентралізовану перевірку сесій, знижуючи навантаження на базу даних і покращуючи масштабованість систем. За останні роки JWT здобув позицію стандарту де-факто для авторизації в мобільних застосунках, SPA (односторінкових застосунках) і системах, побудованих за мікросервісною архітектурою. Зростає й зацікавленість у використанні асиметричної криптографії, зокрема алгоритмів RSA та ECDSA. Ці методи забезпечують високий рівень безпеки при підписанні токенів, передачі ключів або встановленні безпечних з'єднань. Асиметричні ключі дозволяють надійно передавати дані навіть через незахищені мережі, що робить їх надзвичайно важливими для сучасних вебсистем із відкритим API та користувачами зі значною географічною розпорошеністю. Ще однією помітною тенденцією є запровадження безпарольна аутентифікації. У цьому випадку використовуються апаратні ключі, біометричні дані або тимчасові токени. Хоча цей підхід ще не повністю витіснив класичну модель логін-пароль, він стрімко набирає популярності – особливо у

банківській сфері, корпоративних платформах та екосистемах Google, Apple і Microsoft. З технічного боку найбільш затребуваними інструментами для реалізації аутентифікаційних систем залишаються платформені рішення на Node.js з використанням фреймворку Express. До них додаються бібліотеки bcrypt для хешування паролів, jsonwebtoken для генерації токенів і MongoDB – документо-орієнтована база даних із можливістю ефективного зберігання облікових записів, сесій і ключів. Також варто враховувати важливість відповідності міжнародним стандартам безпеки. Зокрема, OWASP Top 10, GDPR та ISO/IEC 27001 акцентують увагу на дотриманні політик безпечної аутентифікації: від хешування паролів із сольовими значеннями до захисту від автоматизованих атак. Ринок web-застосунків виразно демонструє зміну пріоритетів від базових форм автентифікації до впровадження складних криптографічно захищених механізмів. Сучасні підходи спрямовані на забезпечення гнучкості, високого рівня безпеки, інтеграції з хмарними сервісами та простоти масштабування. У цьому контексті створення web-застосунку для безпечної аутентифікації користувачів із залученням криптографічних технологій, таких як RSA, JWT, bcrypt, управління ролями користувачів і захист від брутфорс-атак, є надзвичайно актуальним.

1.2 Поняття аутентифікації в інформаційних системах

Аутентифікація – це процес перевірки, чи є особа або організація, яка отримує доступ до обчислювальної системи, дійсно тим, за кого себе видає. Системи автентифікації приймають рішення за двома можливими варіантами: дозволити або відмовити в доступі, базуючись на наданих облікових даних чи інших доказах доступу. Часто автентифікація функціонує разом із системами авторизації, які визначають, який саме рівень або тип доступу має бути наданий користувачеві. Практично будь-яка обчислювальна система потребує автентифікації: від апаратного забезпечення, мереж і серверів до окремих робочих станцій, мобільних пристроїв та гаджетів Інтернету речей. Однак багатьом пристроям та системам властива слабка або неефективна автентифікація. У деяких випадках вона неправильно налаштована адміністраторами, що створює значні

загрози безпеці. У традиційному процесі аутентифікації користувач надає свої облікові дані, такі як логін та пароль. Система перевіряє ці дані через користувацький каталог, який може зберігатися локально в операційній системі або на сервері аутентифікації. У разі збігу даних доступ до системи надається. Надалі встановлюються права доступу, які визначають, які конкретно дії чи ресурси доступні користувачу, а також включають такі деталі, як часові обмеження або кількість можливих запитів.

З розвитком інтернету більшість додатків почали працювати через HTTP або HTTPS – протоколи без стану. У рамках традиційного підходу це змушувало користувачів повторно вводити облікові дані для кожного нового сеансу. Парольна аутентифікація, крім того, що може бути незручною для користувачів, часто є вразливою до атак. Сучасні підходи до аутентифікації Новітні методи аутентифікації пропонують ефективні рішення до зазначених проблем. Стандартизовані протоколи аутентифікації дозволяють розробникам інтегрувати перевірені та безпечні механізми аутентифікації замість розробки власних рішень. Аутентифікація за допомогою токенів працює таким чином, що користувач ідентифікує себе один раз через сервіс аутентифікації. Потім цей сервіс генерує підписаний токен, який використовується додатком для входу без повторного введення даних до завершення строку дії токена. Багатофакторна аутентифікація додає додаткові рівні безпеки до парольної системи, застосовуючи такі методи, як біометричні дані, фізичні токени або одноразові паролі, що надсилаються на мобільні пристрої.

1.3 Методи аутентифікації користувачів

Однофакторна аутентифікація – однофакторна аутентифікація є найбільш базовим видом і вимагає лише одного фактора. Це може бути зв’язка з імені користувача та пароля або будь-який інший окремий фактор із раніше згаданих. Проте цей тип аутентифікації вважається небезпечним і особливо вразливим до кібератак і порушень безпеки. Паролі, як основа однофакторної аутентифікації, часто використовуються повторно та можуть легко потрапити до рук

зловмисників. Тому компаніям доцільно впроваджувати двофакторну або багатофакторну аутентифікацію, що значно покращує рівень захисту. Такі рішення передбачають використання більш ніж одного фактора, суттєво ускладнюючи несанкціонований доступ.

Двофакторна аутентифікація (2FA) – двофакторна аутентифікація потребує використання двох різних факторів для підтвердження особи. Наприклад, це може бути генератор кодів або біометрія. Завдяки додатковому рівню безпеки, впровадження 2FA значно знижує ризик атак. Навіть якщо пароль користувача опиниться в руках зловмисників, без другого фактора доступ до облікового запису буде неможливим.

Багатофакторна аутентифікація – багатофакторна аутентифікація виходить за межі стандартної 2FA і передбачає використання трьох або більше факторів для підтвердження особи. Кожен додатковий фактор зміцнює рівень захисту. Цей підхід зазвичай застосовується у системах із високими вимогами до безпеки.

Біометрична аутентифікація – біометрія забезпечує підтвердження особи через унікальні фізіологічні характеристики, такі як відбитки пальців, розпізнавання обличчя або голосу. Цей підхід вважається одним із найнадійніших, адже біометричні дані важко відтворити чи викрасти. Окрім високого рівня захисту, біометрична аутентифікація є зручною для користувачів, адже не вимагає запам’ятовування паролів. Завдяки своїм перевагам вона швидко набирає популярності серед бізнесу, сприяючи кращій взаємодії із клієнтами та спрощуючи процес реєстрації нових користувачів.

Аутентифікація без пароля – безпарольна аутентифікація дозволяє безпечно проходити авторизацію без необхідності вводити паролі. До таких методів належать наприклад біометричні дані та одноразові коди з генераторів. Їхня популярність стрімко зростає через зручність використання, що покращує взаємодію з користувачами, пришвидшує доступ до системи та усуває необхідність пам’ятати складні паролі. Крім того, відсутність паролів мінімізує ризик компрометації облікового запису. Надійна аутентифікація клієнта (Strong Customer Authentication, SCA) – вид багатофакторної аутентифікації створений відповідно

до Директиви ЄС про платіжні послуги, що регулює захист онлайн-платежів і фінансових транзакцій. Цей вид аутентифікації передбачає використання як мінімум двох різних факторів для підтвердження особи користувача.

Аутентифікація тісно пов’язана з авторизацією, ці два поняття виконують різні завдання в процесі надання доступу користувачеві до системи. Коректне розуміння і впровадження цих механізмів є надзвичайно важливим для забезпечення надійного захисту програмного забезпечення. Авторизація визначає права доступу користувача після того, як його особу було аутентифіковано. Цей процес встановлює, до яких даних, функцій або ресурсів користувач має доступ на основі його ролі, статусу чи застосованих політик доступу. Наприклад, звичайний користувач може редагувати лише власний профіль, тоді як адміністратор має розширені повноваження – наприклад, управління усіма обліковими записами. Хоча аутентифікація та авторизація тісно пов’язані, це різні поняття і вони мають різні механізми дії. Різницю між ними наведено у табл. 1.1.

Таблиця 1.1 Порівняння аутентифікації та авторизації

№	*Аспект*	*Аутентифікація*	*Авторизація*
1	Визначення	Автентифікація перевіряє особу користувача, зазвичай за допомогою облікових даних, таких як паролі.	Авторизація визначає дозволи та рівні доступу користувача після підтвердження його особи.
2	Механізм	Використовує облікові дані для входу, біометричні дані та інші персональні ідентифікатори, надані користувачем.	Працює за допомогою конфігурацій, встановлених та керованих адміністраторами системи або організації.
3	Передача даних	Дані автентифікації в основному передаються через ідентифікаційні токени, такі як JWT (JSON Web Tokens).	Дані авторизації передаються через токени доступу, що вказують права доступу користувача.

1.4 Основи криптографії

Криптографія – це процес приховування або кодування інформації, що дозволяє лише одержувачу повідомлення отримати доступ до його змісту. Це мистецтво шифрування існує вже тисячоліттями і сьогодні широко застосовується в банківських картах, комп'ютерних паролях та сфері електронної комерції.

Сучасні методи криптографії базуються на використанні складних алгоритмів та шифрів, які забезпечують ефективне шифрування й розшифрування даних. Наприклад, шифрувальні ключі довжиною 128 чи 256 біт гарантують високу ступінь захищеності. Такі сучасні шифри, як Стандарт передового шифрування (Advanced Encryption Standard, AES), вважаються практично неспроможними до злому.

Криптографію часто визначають як практику кодування даних з метою гарантувати, що тільки визначений адресат зможе їх прочитати й опрацювати. Ця галузь кібербезпеки, яка також має назву криптологія, об'єднує знання з інформатики, інженерії та математики для створення складних кодів, що надійно приховують справжній зміст інформації.

1.4.1 Симетричне шифрування

Симетричне шифрування – це метод шифрування, який використовує один ключ для шифрування та розшифрування даних. Хоча воно менш безпечне, ніж асиметричне шифрування, часто вважається ефективнішим, оскільки вимагає меншої обчислювальної потужності.

Шифрування – це процес перетворення читабельного звичайного тексту на нечитабельний зашифрований текст для маскування конфіденційних даних від неавторизованих користувачів.

Майже все, що люди роблять на своїх комп’ютерах, телефонах та пристроях Інтернету речей, залежить від шифрування для захисту даних та безпечного зв’язку. Воно може захистити дані в стані спокою, під час передачі та під час обробки, що робить його критично важливим для кібербезпеки майже кожної організації.

Симетричне шифрування, також відоме як криптографія із симетричним ключем або шифрування із секретним ключем, є одним із двох основних методів шифрування поряд з асиметричним шифруванням. Симетричне шифрування працює шляхом створення єдиного спільного ключа для шифрування та розшифрування конфіденційних даних. Головною перевагою симетричного шифрування є те, що воно просте та ефективне в захисті даних.

Однак симетричне шифрування часто вважається менш безпечним, ніж асиметричне, значною мірою тому, що воно залежить від безпечного обміну ключами та ретельного їх управління. Будь-хто, хто перехоплює або отримує симетричний ключ, може отримати доступ до даних.

Також, оскільки досягнення в галузі штучного інтелекту та квантових обчислень загрожують зруйнувати традиційні методи шифрування, багато організацій покладаються на інтегровані рішення для шифрування для захисту конфіденційних даних.

Симетричне шифрування починається з генерації ключа, який створює єдиний секретний ключ, який усі залучені сторони повинні зберігати в таємниці. Під час процесу шифрування система передає звичайний текст та секретний ключ до алгоритму шифрування даних. Цей процес використовує математичні операції для перетворення звичайного тексту на шифрований текст. Без ключа дешифрування розшифрування зашифрованих повідомлень стає неможливим. Схема роботи симетричного шифрування наведено у рис. 1.1.

Рисунок 1.1 Схема роботи симетричного шифрування

Потім система передає шифрований текст одержувачу, який використовує той самий секретний ключ для розшифрування шифрованого тексту назад у звичайний текст, здійснюючи процес шифрування у зворотному напрямку.

Симетричне шифрування включає два основні типи симетричних шифрів: блокові шифри та потокові шифри.

Блокові шифри, такі як Advanced Encryption Standard (AES), шифрують дані блоками фіксованого розміру. Потокові шифри, такі як RC4, шифрують дані по одному біту або байту за раз, що робить їх придатними для обробки даних у режимі

реального часу. Користувачі часто обирають блокові шифри, щоб забезпечити цілісність даних та безпеку великих обсягів даних, а потокові шифри обираються для ефективного шифрування менших, безперервних потоків даних, таких як зв'язок у режимі реального часу.

Симетричне шифрування є критично важливим для сучасних практик безпеки даних. Це один з найважливіших та найпоширеніших інструментів безпеки даних.

Кодуючи простий текст як зашифрований текст, шифрування може допомогти організаціям захистити дані від низки кібератак, включаючи програми-вимагачі та інші шкідливі програми.

Примітно, що використання шкідливого програмного забезпечення для крадіжки інформації, яке викрадає конфіденційні дані, зросло на 266% порівняно з 2022 роком. Шифрування допомагає боротися з цією загрозою, роблячи дані непридатними для використання хакерами, перешкоджаючи меті їх крадіжки.

Симетричне шифрування ефективне для шифрування великих обсягів даних, оскільки воно обчислювально ефективне та може швидко обробляти великі обсяги даних. Організації широко використовують симетричне шифрування для захисту каналів зв'язку. Протоколи, такі як Transport Layer Security (TLS), використовують симетричне шифрування для ефективного захисту цілісності та конфіденційності даних, що передаються через Інтернет, включаючи електронні листи, миттєві повідомлення та перегляд веб-сторінок.

Під час підключення SSL/TLS клієнт отримує відкритий ключ веб-сайту зі свого сертифіката SSL/TSL для встановлення безпечного ключа сеансу, тоді як веб-сайт зберігає свій закритий ключ у таємниці. Початкове підключення використовує асиметричне шифрування для обміну інформацією та встановлення безпечного ключа сеансу перед переходом до симетричного шифрування для більш ефективної передачі даних. Ця комбінація гарантує, що конфіденційні дані залишаються конфіденційними та захищеними від несанкціонованого доступу під час передачі. Бази даних часто зберігають величезні обсяги конфіденційної інформації, від особистих даних до фінансових записів. Симетричне шифрування

може допомогти зашифрувати ці бази даних або певні поля в них, такі як номери кредитних карток та соціального страхування. Хоча постачальники хмарних послуг відповідають за безпеку сховища, клієнти несуть відповідальність за безпеку в ньому, включаючи безпеку будь-яких даних.

Недавні дослідження показують, що сьогодні більшість організацій використовують гібридну криптографічну інфраструктуру через хмарні та локальні криптографічні рішення. Шифруючи дані в стані спокою, організації можуть гарантувати захищеність даних, навіть якщо база даних скомпрометована. Організації також часто використовують симетричне шифрування для захисту файлів, що зберігаються на локальних системах, спільних дисках та знімних носіях. Шифрування файлів гарантує, що конфіденційні дані залишаються конфіденційними, навіть якщо носій інформації втрачено або викрадено. Шифрування всього диска розширює цей захист, шифруючи цілі пристрої зберігання даних, захищаючи конфіденційні дані на кінцевих точках, таких як ноутбуки та мобільні пристрої.

Симетричні алгоритми шифрування не тільки забезпечують конфіденційність, але й цілісність даних, що є критичним фактором у фінансових транзакціях. Генеруючи коди аутентифікації повідомлень, симетричні ключі можуть допомогти підтвердити, що ніхто не змінював дані під час передачі.

Для кращого захисту конфіденційних даних, особливо коли програмного шифрування може бути недостатньо, організації часто використовують спеціалізовані апаратні компоненти, такі як чіпи або модулі шифрування. Ці апаратні рішення для шифрування зазвичай зустрічаються в смартфонах, ноутбуках та пристроях зберігання даних.

Багато галузей та юрисдикцій мають нормативні вимоги, які вимагають від організацій використовувати певні види шифрування для захисту конфіденційних даних. Дотримання цих правил допомагає організаціям уникнути юридичних санкцій та зберегти довіру клієнтів.

Розглянемо декілька прикладів алгоритмів симетричного шифрування:

1. RC4 – це потоковий шифр із алгоритмом ключа змінної довжини. Цей алгоритм шифрує один байт за раз. Ключовий вхід – це генератор псевдовипадкових бітів, який генерує потокове 8-бітове число, яке непередбачуване без знання вхідного ключа. Вихід генератора називається потоком ключів і поєднується один байт за раз із потоковим шифром відкритого тексту за допомогою операції X-OR;

2. AES – алгоритм, відомий як симетричний блоковий шифр, для шифрування відкритого тексту в нечитабельний код у блоках фіксованого розміру, які можна розшифрувати лише за допомогою оригінального ключа шифрування. Один і той самий ключ використовується як для шифрування, так і для розшифрування. Найновіші стандарти AES безпечні завдяки великій кількості можливих комбінацій ключів.

1.4.2 Асиметричне шифрування

Асиметричне шифрування – це метод шифрування, який використовує два різні ключі: відкритий ключ і закритий ключ – для шифрування та розшифрування даних. Загалом воно вважається безпечнішим, хоча й менш ефективним, ніж симетричне шифрування.

Асиметричне шифрування, також відоме як криптографія з відкритим ключем або асиметрична криптографія, є одним із двох основних методів шифрування поряд із симетричним шифруванням.

Асиметричне шифрування працює шляхом створення пари ключів, одного відкритого та одного закритого. Будь-хто може використовувати відкритий ключ для шифрування даних. Однак лише власники відповідного закритого ключа можуть розшифрувати ці дані. Схему роботи асиметричного шифрування наведено у рис. 1.2.

Головною перевагою асиметричного шифрування є те, що воно усуває необхідність безпечного обміну ключами, що більшість експертів вважають основною проблемою ненадійності симетричного шифрування.

Однак асиметричне шифрування помітно повільніше та вимагає більше ресурсів, ніж симетричне. З цієї причини організації та програми обміну

повідомленнями все частіше покладаються на гібридний метод шифрування, який використовує асиметричне шифрування для безпечного розподілу ключів та симетричне шифрування для подальшого обміну даними.

Рисунок 1.2 Схема роботи асиметричного шифрування

Ключі функціонують як складні коди, необхідні для відмикання сейфа. Без правильного криптографічного ключа користувачі не можуть розшифрувати зашифровані дані. Як правило, чим довший розмір ключа, тим вища безпека. Асиметричне шифрування відоме тим, що має набагато довші довжини ключів, ніж симетричне шифрування, що сприяє його вищій безпеці.

Відкритий ключ шифрує дані або перевіряє цифрові підписи і може вільно поширюватися та ділитися.

Закритий ключ розшифровує дані та створює цифрові підписи, але повинен залишатися в таємниці для забезпечення безпеки.

Безпека криптографії з відкритим ключем залежить від збереження конфіденційності закритого ключа та вільного обміну відкритим ключем. Відкритий ключ може лише шифрувати дані, тому він не має великої цінності для зловмисників. А оскільки користувачам ніколи не потрібно ділитися своїми закритими ключами, це значно знижує ризик перехоплення хакерами цих набагато цінніших ключів.

Після того, як закритий та відкритий ключі створені, люди можуть обмінюватися конфіденційною інформацією. Відправник шифрує повідомлення за допомогою відкритого ключа одержувача, а одержувач використовує свій закритий ключ для розшифрування інформації.

Асиметричне шифрування також може допомогти в процесі аутентифікації.

Наприклад, відправник може зашифрувати повідомлення за допомогою свого закритого ключа та надіслати його одержувачу. Потім одержувач може використати відкритий ключ відправника для розшифрування повідомлення, тим самим підтверджуючи, що його надіслав оригінальний відправник.

Схеми асиметричного шифрування зазвичай реалізуються через інфраструктуру відкритих ключів PKI (Public Key Infrastructure). PKI – це платформа для створення, розповсюдження та перевірки пар відкритих та закритих ключів.

Асиметричні алгоритми шифрування є основою сучасних криптосистем, забезпечуючи основу для безпечного зв'язку та захищаючи конфіденційні дані від несанкціонованого доступу.

Розглянемо декілька прикладів алгоритмів асиметричного шифрування:

1. RSA – це асиметричний алгоритм шифрування, названий на честь його винахідників. Він спирається на математичну складність простих чисел для генерації пар ключів. Він використовує пару публічно-приватних ключів для шифрування та дешифрування, що робить його придатним для безпечної передачі даних та цифрових підписів;

2. ECC – це асиметричний метод шифрування, заснований на математичних властивостях еліптичних кривих над скінченними полями. Він пропонує надійну безпеку з коротшими довжинами ключів, ніж інші алгоритми, що призводить до швидших обчислень та меншого енергоспоживання. Ефективність ECC робить його ідеальним для програм з обмеженою обчислювальною потужністю та часом автономної роботи, таких як мобільні програми, програми безпечного обміну повідомленнями та пристрої Інтернету речей.

3. Протокол Діффі-Геллмана (рис. 1.3) – це метод обміну криптографічними ключами. Один з перших практичних прикладів узгодження ключа, що дозволяє двом учасникам, що не мають жодних попередніх даних один про одного, отримати спільний секретний ключ із використанням незахищеного каналу зв'язку. В основі протоколу лежить концепція часткового обміну ключами у вигляді криптографічного алгоритму з відкритим ключем. За цією схемою кожна сторона

обміну: генерує випадкове натуральне число – свій закритий ключ; узгоджує з віддаленою стороною відкриті параметри, які будуть використані для створення секретного ключа; обчислює відкритий ключ на основі закритого ключа та відкритих параметрів; обмінюється відкритими ключами з віддаленою стороною; обчислює спільний секретний ключ, використовуючи відкритий ключ віддаленої сторони, свій власний закритий ключ та відкриті параметри. Протокол Діффі-Геллмана можна застосовувати до будь-якої кількості сторін обміну, а не лише до двох. Щоб отримати спільний секретний ключ, усі сторони обміну по черзі обчислюють ключі, використовуючи спільні відкриті параметри, відкриті ключі, отримані від інших сторін, та свої відповідні закриті ключі. Але протокол сам по собі не забезпечує перевірку особи сторін. Це означає, що зловмисник може перехопити зв'язок і провести атаку "людина посередині" (MITM), підмінивши ключі. Також протокол використовує великі числа і дорогі обчислення;

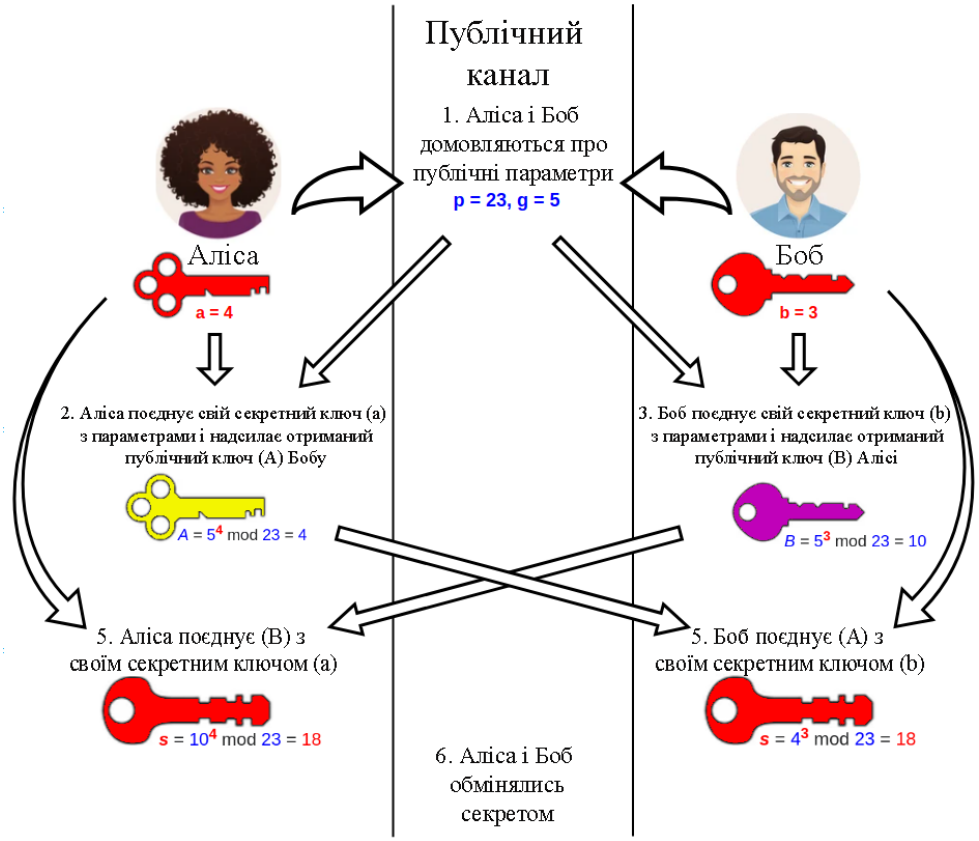


Рисунок 1.3 Схема роботи протоколу Діффі-Геллмана

4. DSA – це асиметрична схема шифрування, яка генерує пару ключів, є дуже схожим на алгоритм RSA. Підпис створюється конфіденційно, хоча його

можна ідентифікувати публічно, перевага цього полягає в тому, що лише один авторитет може створити підпис, але будь-яка інша сторона може перевірити підпис за допомогою відкритого ключа. Як наслідок, DSA швидше підписує, але повільніше перевіряє.

1.4.3 Огляд різниці між симетричним та асиметрични шифруванням

У кібербезпеці шифрування є вирішальним для захисту конфіденційної інформації або даних від несанкціонованого доступу. Це базова технологія, яка зазвичай використовує два різні типи: симетричне та асиметричне шифрування. Ці методи працюють за різними принципами та пропонують різні рівні безпеки.

Основна відмінність між симетричним та асиметричним шифруванням полягає у використанні ключа. Симетричне шифрування використовує один ключ для шифрування та дешифрування, що робить його швидшим, але вимагає безпечного розподілу ключів. З іншого боку, асиметричне шифрування використовує пару ключів: відкритий та приватний – що забезпечує покращену безпеку для обміну ключами та цифрових підписів. Порівняння симетричного і асиметричного шифрування наведено у табл.1.2.

Розглянемо переваги та недоліки симетричного шифрування.

Переваги симетричного шифрування:

1. Швидше, оскільки використовується один ключ для шифрування та дешифрування, воно виконується швидше;

2. Використання аутентифікація за паролем як засіб безпеки для підтвердження особи одержувача;

3. Легкість виконання та керування, користувачі мають лише один ключ для шифрування та дешифрування, тому його легко виконувати та керувати;

4. Ефективне для масового шифрування, його швидкість робить його добре пристосованим для шифрування великих обсягів даних, таких як файли, бази даних та резервні копії.

Недоліки симетричного шифрування:

1. Складність з безпечним обмін ключами шифрування. Це створює проблеми з ризиком компрометації ключів;

2. Симетричне шифрування не є таким масштабованим, оскільки обмін ключами між багатьма людьми може не підходити для різних користувачів.

Таблиця 1.2 Різниця між симетричним та асиметричним шифрування

№	*Характеристика*	*Симетричне*	*Асиметричне*
1	Розмір зашифрованого тексту	Менший зашифрований текст порівняно з оригінальним файлом відкритого тексту.	Більший зашифрований текст порівняно з оригінальним файлом відкритого тексту.
2	Розмір даних	Використовується для передачі великих даних.	Використовується для передачі малих даних.
3	Використання ресурсів	Працює з низьким споживанням ресурсів.	Вимагає високого споживання ресурсів.
4	Довжини ключів	Розмір ключа 128 або 256 біт.	Розмір ключа RSA 2048 біт або більше.
5	Безпечність	Менш безпечно через використання одного ключа для шифрування.	Набагато безпечніше, оскільки для шифрування та дешифрування використовуються два ключі.
6	Кількість ключів	Використовує один ключ.	Використовує два ключі.
7	Конфіденційність	Один ключ для шифрування та дешифрування має ймовірність його компрометації.	Два окремі ключі для шифрування та дешифрування усувають необхідність спільного використання ключа.
8	Швидкість	Є швидким методом.	Працює повільніше з точки зору швидкості.
9	Алгоритми	RC4, AES, DES, 3DES, QUAD.	RSA, протокол Діффі-Геллмана, ECC.

Тепер розглянемо переваги та недоліки асиметричного шифрування.

Переваги асиметричного шифрування:

1. Асиметричне шифрування має два ключі: один відкритий, а інший приватний, тому немає проблем із розподілом ключів;

2. Більш масштабоване у великих мережах.

Недоліки асиметричного шифрування:

1. Має повільнішу продуктивність порівняно із симетричним шифруванням;

2. Складність впровадження та керування ним через великий розмір ключів.

Асиметричне шифрування було введено для вирішення проблеми необхідності спільного використання ключа в симетричній моделі шифрування, усуваючи необхідність спільного використання ключа за допомогою пари публічно-приватних ключів. У таблиці наведено ключові відмінності між симетричним та асиметричним шифруванням.

При виборі алгоритму шифрування варто враховувати такі фактори, як вимоги безпеки, розмір даних та обчислювальна потужність. Як симетричне, так і асиметричне шифрування мають свої унікальні сильні та слабкі сторони. Симетричне шифрування пропонує швидкість та ефективність для передачі великих даних, тоді як асиметричне шифрування забезпечує підвищену безпеку для менших даних або обміну ключами. Завжди потрібно використовувати алгоритм шифрування, який підходить для поточного завдання.

1.5 Вибір алгоритму RSA як основного алгоритму шифрування

Для реалізації web-застосунку було обрано алгоритм RSA для забезпечення безпечної аутентифікації користувачів як головний метод шифрування. Це рішення ґрунтується на кількох ключових перевагах, які роблять RSA надійним і практичним варіантом. Перш за все, RSA вважається одним із добре перевірених алгоритмів асиметричної криптографії, що забезпечує високий рівень захисту даних завдяки використанню пари ключів – публічного та приватного. У цій моделі публічний ключ зберігається на сервері, тоді як приватний ключ залишається захищеним на стороні клієнта, що усуває ризики, пов’язані з передачею секретних ключів, які притаманні симетричним алгоритмам. Другою важливою перевагою є висока сумісність RSA із сучасними стандартами безпеки, зокрема з форматом JSON Web Token (JWT), який застосовується в системі аутентифікації. Підпис токена за допомогою приватного ключа RSA дає змогу клієнту перевірити справжність токена, використовуючи лише публічний ключ, без необхідності доступу до серверних секретів. Таке рішення не лише підвищує безпеку, але й полегшує масштабування системи та інтеграцію з іншими сервісами. Крім цього, RSA є загальноприйнятим стандартом у сфері інформаційної безпеки через його

доведену ефективність та надійність. Його широко підтримує більшість сучасних криптографічних бібліотек, що спрощує впровадження алгоритму та гарантує стабільність роботи програмних рішень. Попри існування новітніх алгоритмів, таких як криптографія на еліптичних кривих (ECC), RSA зберігає свою популярність завдяки зрозумілості, простоті реалізації та достатньому рівню безпеки для більшості веб-сценаріїв. Зважаючи на всі перераховані причини, алгоритм RSA було обрано як найбільш оптимальний варіант для досягнення поставлених завдань.

1.6 Огляд сучасних систем і протоколів аутентифікації

Сучасна аутентифікація має велику кількість механізмів безпеки, розроблених для безпечної та зручної аутентифікації користувачів. Використовуються різноманітні протоколи та технології, щоб гарантувати, що лише авторизовані особи отримують доступ до систем і даних. Розглянемо сучасні системи та протоколи аутентифікації.

Аутентифікація на основі токенів – це протокол, який дозволяє користувачам підтверджувати свою особу та отримувати унікальний токен доступу.

Протягом терміну дії токена користувачі отримують доступ до веб-сайту або програми, для якої його було видано, замість того, щоб повторно вводити облікові дані щоразу, коли вони повертаються на ту саму веб-сторінку, програму чи будь-який ресурс, захищений тим самим токеном.

Токени аутентифікації працюють як квиток. Користувач зберігає доступ, поки токен залишається дійсним. Після того, як користувач виходить із системи або закриває програму, токен стає недійсним.

Аутентифікація на основі токенів відрізняється від традиційних методів аутентифікації на основі пароля або сервера. Токени пропонують другий рівень безпеки, а адміністратори мають детальний контроль над кожною дією та транзакцією. Але використання токенів вимагає певних знань кодування.

SSO (Single Sign-On з англ. Переклад Єдиний вхід) – це схема автентифікації, яка дозволяє користувачам входити в систему один раз за допомогою одного

набору облікових даних та отримувати доступ до кількох програм протягом одного сеансу.

SSO спрощує автентифікацію користувачів, покращує взаємодію з користувачем і, за умови правильного впровадження, підвищує безпеку. Він часто використовується для керування аутентифікацією та безпечного доступу до, наприклад корпоративних мереж, студентських порталів, публічних хмарних сервісів та інших середовищ, де користувачам потрібно перемикатися між різними програмами для виконання своєї роботи.

SSO працює наступним чином:

1. Користувач входить до одного з постачальників послуг або до центрального порталу, використовуючи облікові дані для входу SSO.

2. Коли користувача успішно аутентифіковано, рішення SSO генерує токен аутентифікації сеансу, що містить певну інформацію про особу користувача – ім'я користувача, адресу електронної пошти тощо. Цей токен зберігається у веб-браузері користувача або в системі SSO.

3. Коли користувач намагається отримати доступ до іншого довіреного постачальника послуг, програма перевіряє в системі SSO, чи користувач вже аутентифікований для сеансу. Якщо так, SSO перевіряє користувача, підписуючи токен автентифікації цифровим сертифікатом, і користувач отримує доступ до програми. Якщо ні, користувачеві пропонується повторно ввести облікові дані для входу.

OAuth 2.0 – протокол, який дозволяє додаткам отримувати обмежений доступ до облікових записів користувачів за допомогою HTTP-сервісів. Це протокол відкритого стандарту, який дозволяє користувачам ділитися своїми приватними ресурсами, що зберігаються на одному сайті, з іншим сайтом, без необхідності надавати свої облікові дані.

OpenID Connect – протокол, який надає можливість клієнтам підтверджувати особу користувача на основі аутентифікації, виконаної сервером авторизації. Це сумісний протокол автентифікації, що базується на специфікаціях OAuth 2.0. Він спрощує спосіб перевірки особи користувачів на основі аутентифікації, виконаної

сервером авторизації, та отримання інформації про профіль користувача. OpenID Connect дозволяє розробникам додатків та веб-сайтів запускати процеси входу та отримувати перевірені твердження про користувачів через веб-клієнти, мобільні пристрої та JavaScript. Набір специфікацій можна розширювати для підтримки низки додаткових функцій, таких як шифрування даних ідентифікації, виявлення постачальників OpenID та вихід із сеансу.

SAML 2.0 (Security Assertion Markup Language) – протокол, що забезпечує полегшення обміну ідентифікаторами аутентифікації та авторизації між доменами безпеки. Вона дозволяє використовувати веб-орієнтований міждоменний єдиний вхід (SSO), щоб мінімізувати кількість токенів автентифікації, що використовуються для кожного користувача. Цей протокол на основі XML використовує токени безпеки з твердженнями для передачі інформації про певну особу.

SCIM (System for Cross-domain Identity Management) – протокол, створений для автоматизації обміну даними про користувачів між різними системами ідентифікації. Специфікація Системи керування міждоменною ідентифікацією (SCIM) розроблена для спрощення керування ідентифікаторами користувачів у хмарних додатках та сервісах. Набір специфікацій прагне спиратися на досвід роботи з існуючими схемами та розгортаннями, приділяючи особливу увагу простоті розробки та інтеграції, застосовуючи при цьому існуючі моделі автентифікації, авторизації та конфіденційності. Його метою є зменшення вартості та складності операцій управління користувачами шляхом надання загальної схеми користувача та моделі розширення, а також зв'язуючих документів для забезпечення шаблонів обміну цією схемою за допомогою стандартних протоколів.

1.7 Вимоги до безпеки web-застосунку

Вимоги до безпеки веб-застосунків включають низку практик та заходів для захисту застосунків від різних загроз, забезпечення конфіденційності, цілісності та доступності даних. Ключові аспекти включають надійну автентифікацію,

перевірку вхідних даних, шифрування, регулярне тестування безпеки та впровадження надійних механізмів контролю доступу.

Розглянемо кожну вимогу окремо.

1. Аутентифікація та авторизація. Системи аутентифікації та авторизації призначені для підтвердження особи користувачів та управління їхніми правами доступу. Використання ефективних методів контролю, як-от багатофакторна аутентифікація або доступ за допомогою ключів, допомагає захистити систему від проникнення зловмисників.

2. Захист даних. Особисті чи платіжні дані користувачів повинні бути надійно захищені. Необхідно забезпечити їхнє зберігання з дотриманням суворих правил доступу та організовувати регулярне резервне копіювання. Шифрування підвищує рівень захищеності, оскільки у разі перехоплення дані залишаються нерозбірливими.

3. Охорона сесій. Авторизовані сесії користувачів потребують надійного управління. Це запобігає спробам сторонніх осіб доступитися до облікових записів або отримати конфіденційні дані.

4. Валідація введених даних. Для уникнення атак на кшталт SQL-ін’єкцій або XSS усі дані, введені користувачами, повинні піддаватися перевірці. Застосунки мають приймати лише інформацію, що відповідає очікуваному формату: наприклад, у числових полях – лише цифри, а текстові – фільтрувати потенційно шкідливий код. Валідуючи дані на кожному етапі, можна значно знизити ризик впровадження шкідливих скриптів, здатних порушити функціонування системи чи отримати доступ до даних.

5. Захист від автоматизованих атак. Для запобігання брутфорс-атакам, скануванню та іншим автоматизованим загрозам слід використовувати обмеження частоти запитів (rate limiting), паузу між спробами входу та механізми блокування при підозрілій активності.

6. Безпека API. У разі використання веб-API потрібно забезпечити їхню автентифікацію, авторизацію та захист від типових атак наприклад, API rate

limiting, CORS-контроль, перевірка вмісту запитів. Додатково варто використовувати API-ключі або токени доступу, що мають обмежений термін дії.

7. Безпека конфігурації. Конфігураційні файли не повинні містити чутливої інформації у відкритому вигляді наприклад, пароль до БД. Важливо вимикати непотрібні функції, уникати демонстраційних або тестових модулів у версіях застосунку, які будуть опубліковані, а також приховувати інформацію про версії серверного ПЗ.

8. Регулярне тестування безпеки. Веб-застосунок має проходити періодичні перевірки безпеки, включно з ручним і автоматизованим тестуванням. Це дозволяє виявити та усунути вразливості до того, як ними скористаються зловмисники.

1.8 Огляд систем-аналогів

Створимо порівняння систем аналогів у табл. 1.3 і розглянемо кожну окремо.

Для створення безпечного веб-застосунку з аутентифікацією користувачів та використанням криптографічних методів було проаналізовано кілька поширених рішень, що застосовуються у сучасних вебсервісах та фреймворках. Порівняння систем-аналогів дозволяє виявити сильні та слабкі сторони існуючих рішень і врахувати їх при розробці власної системи.

Одним із найбільш популярних рішень є Firebase Authentication – хмарна платформа від Google, яка підтримує автентифікацію за допомогою електронної пошти та пароля, соціальних мереж, а також одноразових кодів. Вона використовує надійні механізми захисту, зокрема SSL/TLS для передавання даних та bcrypt для зберігання хешів паролів.

Ще одним широко використовуваним сервісом є Auth0 – SaaS-рішення для централізованої автентифікації та авторизації. Auth0 підтримує протоколи OAuth 2.0, OpenID Connect, JSON Web Token і багатофакторну автентифікацію. Серед переваг: масштабованість, гнучкість налаштувань і підтримка різних типів автентифікації. Недоліком може бути висока вартість при збільшенні кількості користувачів або функціоналу, а також певні обмеження безкоштовної версії.

Таблиця 1.3 Порівняння систем-аналогів

№	*Система*	*Опис*	*Методи захисту*	*Переваги*	*Недоліки*
1	Firebase Auth	Хмарна автентифікація з підтримкою паролів, соцмереж, OTP	SSL/TLS, bcrypt, зберігання токенів	Легка інтеграція, MFA, автоматичне управління сесіями	Залежність від Google, обмежена кастомізація
2	Auth0	SaaS-сервіс для автентифікації та авторизації користувачів	JWT, OAuth2, OpenID, MFA	Підтримка багатьох протоколів, керування політиками доступу	Висока вартість при масштабуванні, обмеження безкоштовної версії
3	Keycloak	Open-source IAM-система з підтримкою SSO, OAuth2, LDAP	SSO, шифрування, MFA	Безкоштовність, кастомізація, панель керування	Складна інтеграція, високі вимоги до ресурсів
4	Passport.js	Бібліотека для Node.js з багатьма стратегіями аутентифікації	JWT, локальні стратегії, інтеграція з bcrypt	Гнучкість, велика спільнота, добра інтеграція з Express.js	Потребує ручної реалізації перевірок, складне масштабування
5	NextAuth.js	Аутентифікація для Next.js з підтримкою Email, OAuth	JWT, TLS, шифрування сеансів	Швидка інтеграція, кастомізація, безпечне зберігання сесій	Обмежена підтримка за межами Next.js, кастомізація ускладнена

Keycloak – це open-source система управління ідентичністю та доступом, яка підтримує SSO, протоколи OAuth 2.0, OpenID і інтеграцію з LDAP. Вона надає розширені можливості керування користувачами та правами доступу, підтримує шифрування, багатофакторну автентифікацію та створення власних політик доступу.

Для розробників Node.js-платформи зручною є бібліотека Passport.js, яка забезпечує підтримку численних стратегій аутентифікації – локальної , OAuth, Google, Facebook тощо. Бібліотека добре інтегрується з Express.js і дозволяє реалізувати кастомні механізми захисту. Проте вона не надає готового інтерфейсу для користувача чи централізованого керування, тож потребує ручного налаштування захисту сесій, перевірки прав доступу та іншого.

1.9 Огляд засобів розробки

Наразі найпопулярнішою системою керування базами даних є MongoDB. Вона використовує формат JSON для зберігання даних і є нереляційною системою. Якщо порівнювати MySQL і MongoDB, то MySQL – це RDBMS(Relational Database Management System), а MongoDB – NoSQL(Not Only SQL). Це означає що MySQL є системою керування реляційними базами даних, тобто дані в ній зберігаються у вигляді таблиць із чітко визначеною структурою (рис. 1.4). У таких таблицях дані організовані в рядки й стовпці, а зв’язки між таблицями реалізуються за допомогою первинних та зовнішніх ключів. Робота з даними у MySQL здійснюється за допомогою мови SQL (Structured Query Language), яка забезпечує інструменти для створення, читання, оновлення та видалення даних. MySQL вимагає попереднього визначення схеми таблиці, тобто структура бази даних є строгою, і будь-які зміни у структурі потребують модифікації самої схеми. Такий підхід ідеально підходить для систем, де важлива структурованість, узгодженість і складні зв’язки між сутностями, як-от фінансові системи чи бухгалтерські обліки.

Натомість MongoDB належить до категорії NoSQL-баз даних і є документно-орієнтованою системою зберігання даних. У MongoDB дані зберігаються у вигляді документів формату JSON, які можуть мати довільну структуру. Це означає, що

різні документи в одній колекції можуть мати різні поля, що надає гнучкість у зберіганні інформації. MongoDB не вимагає заздалегідь визначеної схеми, що дозволяє легко змінювати структуру даних у процесі розробки. Такий підхід особливо ефективний у випадках, коли структура даних часто змінюється або коли необхідна висока швидкість масштабування, як-от у веб-застосунках, логах подій чи системах з великою кількістю неуніфікованих даних. MongoDB швидший, бо працює асинхронно, а MySQL – синхронно. Щоб пришвидшити синхронну систему, потрібно додавати потоки, що вимагає більше апаратного забезпечення – це економічно неефективно. Саме тут MongoDB має перевагу.

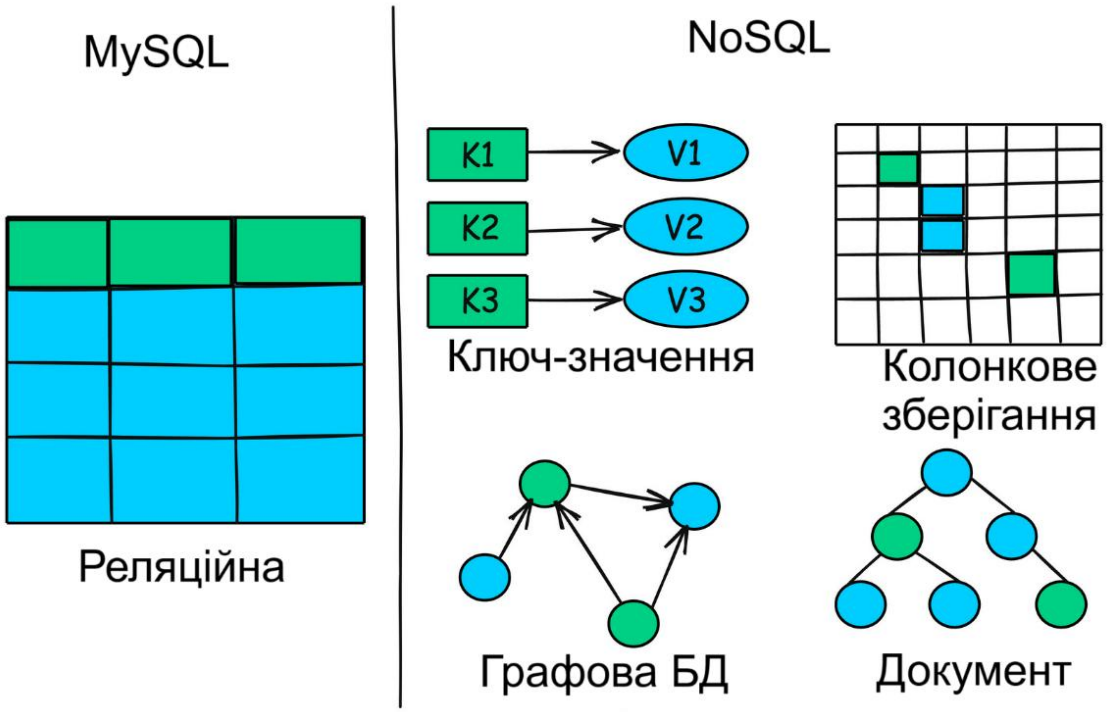


Рисунок 1.4 Дата бази типів MySQL та NoSQL

JavaScript було обрано як основну мову програмування для реалізації логіки web-застосунку з кількох вагомих причин. JavaScript є стандартом для розробки динамічних вебінтерфейсів. Усі сучасні веббраузери підтримують цю мову без необхідності встановлення додаткового програмного забезпечення. Це дозволяє запускати код напряму у браузері користувача, забезпечуючи миттєву реакцію на дії без потреби звернення до сервера. JavaScript дозволяє реалізовувати логіку взаємодії клієнта з сервером через HTTP-запити , зокрема надсилання облікових даних для авторизації, отримання токена доступу, збереження його у браузері та

автоматичне перенаправлення користувача після успішного входу. Окрім цього, JavaScript допомагає у перевірці наявності токена при завантаженні сторінки профілю.

Також JavaScript забезпечує широкі можливості для інтерактивності. З його допомогою можна реалізувати перевірку введених даних на стороні клієнта, обробку подій, динамічне оновлення вмісту сторінки без перезавантаження та інтеграцію з API.

JavaScript є невід'ємною частиною повного стеку веброзробки, зокрема завдяки бібліотекам, що дозволяє використовувати одну й ту ж мову як на клієнтській, так і на серверній частині. Це значно спрощує розробку, полегшує обмін даними між клієнтом і сервером. Крім того, мова має велику спільноту, розвинену документацію та безліч готових рішень і бібліотек, що дозволяє пришвидшити розробку, а також швидко знаходити відповіді на типові питання.

Для реалізації клієнтської частини web-застосунку були використані HTML та CSS, які є основою будь-якого інтерфейсу користувача. HTML (HyperText Markup Language) застосовується для створення структури вебсторінок. З його допомогою було реалізовано розмітку всіх елементів інтерфейсу, таких як форми для введення логіна та пароля, кнопки, повідомлення про помилки, профіль користувача тощо. Завдяки HTML вдалося чітко організувати логіку та ієрархію компонентів сторінки. HTML дозволяє пов’язати клієнтську частину з JavaScript-логікою, зокрема для обробки подій та надсилання запитів на сервер.

Для стилізації інтерфейсу використовувалася мова CSS. CSS (Cascading Style Sheets) – це мова стилів, яка використовується для опису зовнішнього вигляду та форматування HTML-документів. Вона визначає, як саме елементи веб-сторінки мають відображатися в браузері: кольори, шрифти, відступи, розташування блоків, ефекти при наведенні, а також адаптивну поведінку на різних екранах. Основна мета CSS – відокремити структуру від презентації, що дозволяє зручніше підтримувати, змінювати та масштабувати вебінтерфейси. CSS дозволяє керувати стилями для окремих елементів або для цілих класів і блоків, що робить його ефективним інструментом для повторного використання стилів та забезпечення

єдиного вигляду сайту. Сучасні можливості CSS включають підтримку анімацій, переходів, гнучких макетів, темного режиму, а також адаптивного дизайну через медіазапити, що забезпечує зручність перегляду сайту на різних пристроях. Інтерфейс вебзастосунку відіграє важливу роль, адже напряму впливає на користувацький досвід.

1.10 Архітектура застосунку

Архітектура web-застосунку – це модель, яка визначає, як компоненти взаємодіють один з одним. Наприклад, клієнт-серверна архітектура, де клієнт є інтерфейсом користувача, а сервер – сервером, є поширеною моделлю. Вибір архітектури тісно пов'язаний з тим, як логіка додатку розподілена між клієнтською та серверною сторонами. Технічно, це складний скелет веб-додатку, що охоплює його елементи, бази даних, системи, сервери, інтерфейси та всю комунікацію між ними. Більш це вказує на складну логіку відповідей на запити клієнта та сервера. Загальна архітектура web-застосунків наведена на рис. 1.5.

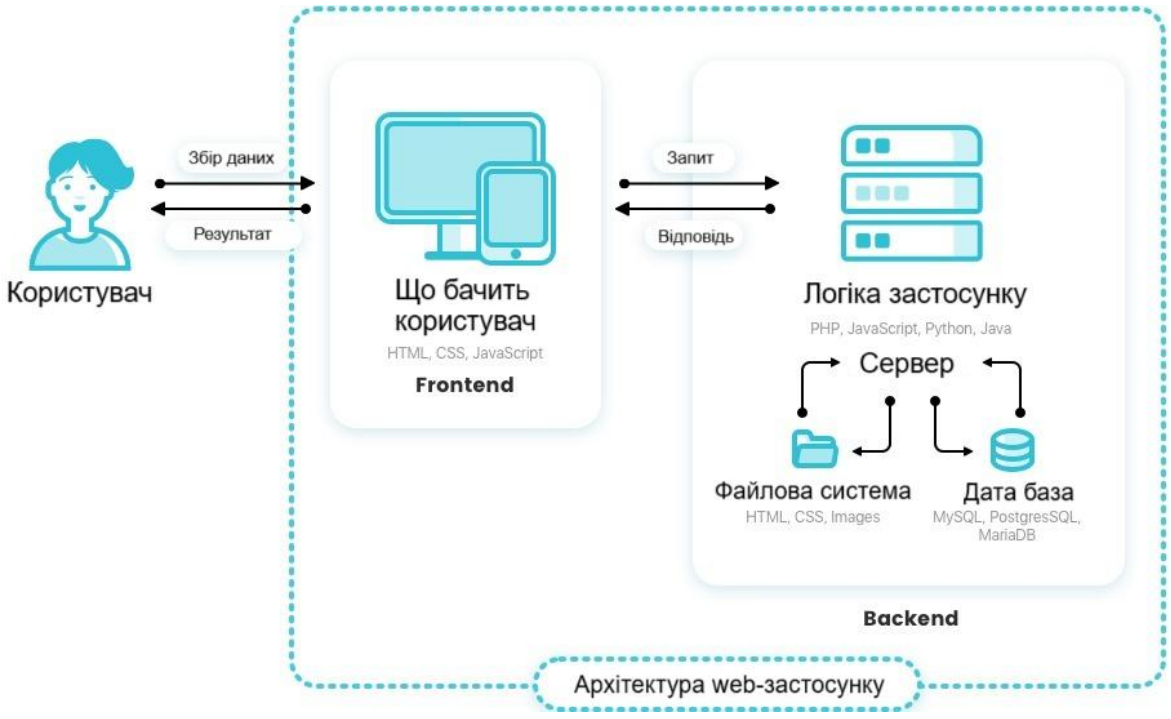


Рисунок 1.5 Загальна архітектура web-застосунків

Оскільки додатки відрізняються складністю та функціональністю, кількість шарів та компонентів відповідно змінюється. У деяких випадках застосунок достатньо простий, щоб працювати як моноліт, зберігаючи всю архітектуру

дизайну веб-додатку в одному місці. Однак більшість веб-додатків складаються з кількох компонентів або рівнів, які взаємодіють один з одним. Як правило, архітектура поділяється на дві основні групи: інтерфейс користувача та структурні веб-компоненти. Структурні веб-компоненти, у свою чергу, включають як клієнтські, так і серверні компоненти.

Коли задіяно багато компонентів, одних лише описів може бути недостатньо, щоб чітко зрозуміти всю систему. Розглянемо кожен компонент архітектури окремо.

1. Сервери web-застосунків. Цей компонент є дескриптором розгортання застосунків. Він обробляє запити користувача та надсилає відповіді назад до початкового браузера. Для цього він звертається до серверної інфраструктури, включаючи базу даних, чергу завдань, сервер кешу тощо.

2. База даних. Цей компонент пропонує різні інструменти для виконання, видалення, упорядкування та оновлення записів даних. Сервери web-застосунків взаємодіють із серверами завдань без будь-якого посередника.

3. Сервіс кешування забезпечує просте та швидке зберігання та пошук даних. Коли користувач отримує інформацію від сервера, результати пошуку можна кешувати. В результаті майбутні запити будуть повертатися набагато швидше.

4. Черга завдань. Ця черга має два компоненти: чергу завдань та сервери, які обробляють ці завдання. Багато веб-серверів виконують велику кількість завдань другорядної важливості. Завдання, яке має бути виконане, потрапляє в чергу та буде виконуватися відповідно до розкладу.

1.10.1 Клієнт-серверна модель

Модель клієнт-сервер – це концепція організації обчислювальної мережі, яка визначає взаємодію між двома або більше комп’ютерами. У цій схемі клієнтські пристрої формують запити й отримують доступ до послуг або ресурсів, які надаються централізованим сервером. Вона встановлює логіку обміну даними, дозволяючи численним клієнтам запускати застосунки або завантажувати файли з єдиного сервера. Завдяки цьому модель забезпечує узгодженість даних на всіх підключених пристроях.

Основними можливостями мережі, побудованої за моделлю клієнт-сервер, є: робота з тимчасовим і локальним сховищем; формування запитів до сервера; виконання функції посередника між користувачами і серверами; проведення операцій із базами даних; організація зв'язку між серверами; обробка користувацьких запитів; збереження файлів на серверах; доступ до серверної інформації; виконання запитів у базах даних; розробка інтерактивних застосунків.

Мережі з клієнт-серверною моделлю складаються з клієнтів та серверів. Клієнти – пристрої, які звертаються до серверів. Сервери – це апаратне чи програмне забезпечення, що виконує певні функції для своїх клієнтів. На рис.1.6 показано схему взаємодії між клієнтом, сервером та базою даних.

Рисунок 1.6 Взаємодія компонентів клієнт-серверної моделі

Клієнт-серверна модель має низку переваг, що роблять її ефективною для побудови сучасних веб-застосунків. Вона забезпечує чіткий розподіл обов'язків: клієнт відповідає за відображення інтерфейсу та взаємодію з користувачем, а сервер – за зберігання, обробку даних і логіку. Також централізоване зберігання даних на сервері спрощує керування інформацією, її оновлення та резервне копіювання.

Крім того, клієнт-серверна модель підвищує безпеку та забезпечує гнучкість у розробці, дозволяючи змінювати або оновлювати клієнт і сервер незалежно один від одного.

1.10.2 Безпечне зберігання даних

Для безпечного веб-сервера всі компоненти також повинні бути захищені, щоб забезпечити належний захист конфіденційних даних. Якщо безпека в будь-

який момент порушиться, зловмисники можуть отримати доступ до веб-програми та отримати дані з бази даних або підробити їх.

Конфіденційні дані можуть бути будь-якою інформацією, яку потрібно захистити від несанкціонованого доступу для захисту конфіденційності або безпеки особи чи організації. Вона може включати будь-яку інформацію, що стосується: паролі, парольні фрази, ключі шифрування, токени, номери кредитних карток, особиста контактна інформація, така як імена, номери телефонів, адреси електронної пошти, облікові записи користувачів, фізичні адреси тощо. Також це може бути особиста ідентифікаційна інформація або дані з високим впливом на бізнес. Для зберігання цих даних слід вибирати правильні механізми зберігання. Вони повинні найнадійніше зберігати інформацію, зменшувати пропускну здатність та покращувати швидкість реагування.

Безпека зберігання даних у веб-застосунку є одним із ключових елементів загальної безпеки системи, оскільки робота з персональною чи платіжною інформацією потребує суворого дотримання принципів конфіденційності, цілісності та доступності. У таких застосунках дані зазвичай зберігаються на сервері, у базах даних, а також можуть тимчасово оброблятися чи зберігатися на стороні клієнта – наприклад. Тому особливо важливо забезпечити комплексний підхід до безпеки на всіх етапах роботи з даними: у зберіганні, передачі та обробці. Серверна частина застосунку має використовувати сучасні методи шифрування для захисту чутливих даних. Паролі не повинні зберігатися у відкритому вигляді – замість цього застосовуються криптографічні хеш-функції, такі як bcrypt чи Argon2, які значно ускладнюють відновлення оригінального пароля навіть у разі витоку інформації. Для більш критичних даних, як-от токени, ключі доступу чи платіжні реквізити, варто використовувати методи симетричного чи асиметричного шифрування. Окрім цього, на рівні бази даних варто впроваджувати принцип найменших привілеїв, тобто забезпечувати доступ лише до тих даних, які необхідні конкретному компоненту для виконання його завдань.

У середовищі веб-застосунків надзвичайно важливо забезпечити безпечну передачу даних між клієнтом і сервером. Для цього необхідно обов'язково

використовувати протокол HTTPS, який надає можливість шифрування трафіку і захисту від перехоплення інформації під час передачі. Проте навіть із використанням HTTPS варто з обережністю зберігати чутливу інформацію на стороні клієнта, адже вони можуть бути вразливими до атак. Якщо все ж є необхідність зберігати дані тимчасово, потрібно ретельно обирати відповідний механізм. Сучасні браузери підтримують декілька способів локального збереження даних: sessionStorage, localStorage та IndexedDB. Розглянемо кожен із них:

1. SessionStorage дозволяє зберігати інформацію лише до закриття вкладки браузера і може бути корисним для зберігання невеликої кількості інформації, специфічної для сеансу. Однак дані у sessionStorage доступні через JavaScript, тому зберігати там конфіденційну інформацію не слід. SessionStorage недоступний для веб-працівників або сервісних працівників. Він обмежений приблизно 5 МБ і може містити лише рядки.

2. LocalStorage зберігає дані безстроково до їх видалення користувачем або самим застосунком і використовується для постійного зберігання даних для всього веб-сайту. Незважаючи на простоту використання, цей механізм також є небезпечними для зберігання чутливих відомостей, оскільки він не шифрується та залишається доступними через будь-які скрипти на сторінці, що робить їх потенційними цілями для XSS-атак. LocalStorage також недоступний для веб-працівників або сервісних працівників. Він обмежений приблизно 5 МБ і може містити лише рядки.

3. IndexedDB є більш потужним інструментом для роботи зі складно структурованими даними, має переваги у гнучкості організації доступу й підходить для зберігання великих обсягів локальної інформації. Працює як об'єктна база даних і дозволяє зберігати JavaScript-об'єкти у вигляді пар «ключ–значення». IndexedDB підтримує створення індексів, що дає змогу швидко шукати й фільтрувати дані за певними полями, подібно до класичних баз даних. На відміну від простіших рішень, таких як localStorage або sessionStorage, які працюють лише з текстовими рядками, IndexedDB дозволяє зберігати складні типи даних: об'єкти,

масиви, дати, файли тощо. База даних в IndexedDB організована у вигляді сховищ, які можна порівняти з таблицями в реляційних БД. Доступ до IndexedDB здійснюється асинхронно через API, що не блокує основний потік виконання сторінки.

У табл.1.4 наведено основну різницю між sessionStorage, localStorage та IndexedDB.

Таблиця 1.4 Порівняння сховищ на стороні клієнта

№	*Характеристика*	*LocalStorage*	*SessionStorage*	*IndexedDB*
1	Обсяг збереження	5-10 мегабайт	5-10 мегабайт	До сотень мегабайт і більше
2	Термін збереження	Постійне поки не видалено вручку	До закриття вкладки	Постійне
3	Доступність	Сторінки того ж походження	Лише в межах однієї вкладки	Сторінки того ж походження
4	Тип даних	Тільки рядки	Тільки рядки	Об’єкти, файли, складні структури
5	Асинхронність	Ні	Ні	Так
6	Чи підтримує офлайн-режим	Так	Обмежена	Так
7	Зручність використання	Просте	Просте	Складніше

IndexedDB має низку переваг, які роблять її хорошим інструментом для зберігання даних у web-застосунках. По-перше, вона дозволяє зберігати великі обсяги структурованих даних без потреби у сервері, що особливо корисно для офлайн-доступу до інформації. По-друге, IndexedDB працює асинхронно, тому не блокує основний потік виконання JavaScript, що позитивно впливає на продуктивність застосунку. Також вона підтримує складні типи даних, зокрема об’єкти, масиви та вкладені структури, на відміну від localStorage і sessionStorage, які працюють лише зі строковими значеннями. Крім того, IndexedDB підтримує механізми індексування та запитів, подібні до реляційних баз даних, що дає змогу ефективно шукати та фільтрувати дані. Завдяки цьому вона ідеально підходить для створення складних веб-застосунків, які потребують швидкого доступу до локально збережених даних.

1.11 Вибір технологій та інструментів реалізації

1.11.1 Node.js

Node.js – це багатоплатформне середовище для JavaScript із відкритим кодом, яке широко застосовується завдяки своїй універсальності і потужності для різних видів проєктів. Основою Node.js є рушій JavaScript V8, що використовується у браузері Google Chrome, але тут він функціонує поза межами браузера, забезпечуючи високу продуктивність.

Застосунки на Node.js виконуються в одному процесі, без створення окремих потоків для кожного вхідного запиту. У його стандартній бібліотеці реалізовані асинхронні інтерфейси введення/виведення, які дозволяють уникнути блокування під час виконання. Загалом, бібліотеки Node.js побудовані за принципом неблокуючої роботи, тому випадків блокування мінімум.

Під час операцій введення/виведення, наприклад, при роботі з мережею, базами даних або файловими системами, Node.js не блокує потік. Замість цього виконання продовжується одразу після отримання відповіді, уникаючи простоїв і перепон у роботі процесора.

Завдяки цьому Node.js здатний одночасно обробляти тисячі з’єднань на одному сервері без необхідності в складному управлінні потоками, яке зазвичай може стати джерелом помилок.

1.11.2 MongoDB

MongoDB є потужною і всебічною документно-орієнтованою базою даних, призначеною для підтримки широкого спектра застосунків у різних галузях промисловості. Модель документованих даних, яка використовується у MongoDB, пропонує гнучкість зберігання даних у формі документів, аналогічних до JSON. Це сприяє природній інтеграції з об’єктами в більшості мов програмування, що полегшує взаємодію розробників з даними. Така гнучка схема дозволяє легко адаптувати структуру даних у процесі зміни вимог до застосунку.

MongoDB оснащено виразним API для запитів, що підтримує різні структури даних. Висока продуктивність і масштабованість є одними з основних переваг

MongoDB. Його архітектура передбачає горизонтальне масштабування, що забезпечує обробку великих обсягів даних та високе навантаження, роблячи його ідеальним для застосунків з високою пропускною здатністю і низькою затримкою.

MongoDB забезпечує продуктивний досвід за допомогою драйверів для всіх основних мов програмування, графічного інтерфейсу MongoDB Compass та розширень для Visual Studio Code, дозволяючи працювати з базою безпосередньо з IDE. MongoDB Shell на базі JavaScript і Node.js діє як середовище для інтерактивної роботи з базою.

1.11.3 JSW

JSON Web Token (JWT) – це відкритий стандарт, який пропонує компактний і самодостатній спосіб безпечного обміну інформацією між сторонами у вигляді JSON-об’єкта. Ця інформація надійна та піддається перевірці завдяки цифровому підпису. JWT можуть підписуватися секретним ключем або парою відкритого і приватного ключів, наприклад алгоритми RSA або ECDSA.

Підписані токени дозволяють перевірити цілісність даних, що зберігаються в них, тоді як зашифровані токени забезпечують приховування цих даних від сторонніх. У тих випадках, коли токени підписуються парою відкритого/приватного ключів, підпис стає доказом того, що токен створений саме стороною, яка володіє приватним ключем. JSON Web Tokens має кілька поширених сценаріїв застосування: авторизація є одним із найрозповсюдженіших способів використання JWT, після успішної аутентифікації користувача кожен наступний запит включає JWT, це дозволяє отримувати доступ до маршрутів, ресурсів чи сервісів, які дозволені для даного токена. Завдяки компактності й універсальності токенів вони широко використовуються для реалізації схем єдиного входу, особливо в системах, що діють між різними доменами. JWT також забезпечують зручний спосіб безпечного передавання інформації між сторонами, оскільки токени можуть бути підписані, це гарантує, що відправник є справжнім і що інформація не була змінена, підпис формується на основі заголовка й корисного навантаження, що дозволяє легко перевірити автентичність і цілісність змісту токена. Принцип роботи JWT токенів зображено на рис. 1.7

Рисунок 1.7 Принцип роботи JWT

JWT токен складається з кількох частин (рис. 1.8):

1. Header (заголовок). Він вказує, який алгоритм підпису використано, і який тип токена;

2. Payload (навантаження). Містить claims – тобто корисні дані (твердження) про користувача або інші сутності. Усі ці дані не шифруються, а лише підписуються;

3. Signature (підпис). Перевіряє цілісність токена – чи не був Payload або Header змінений.

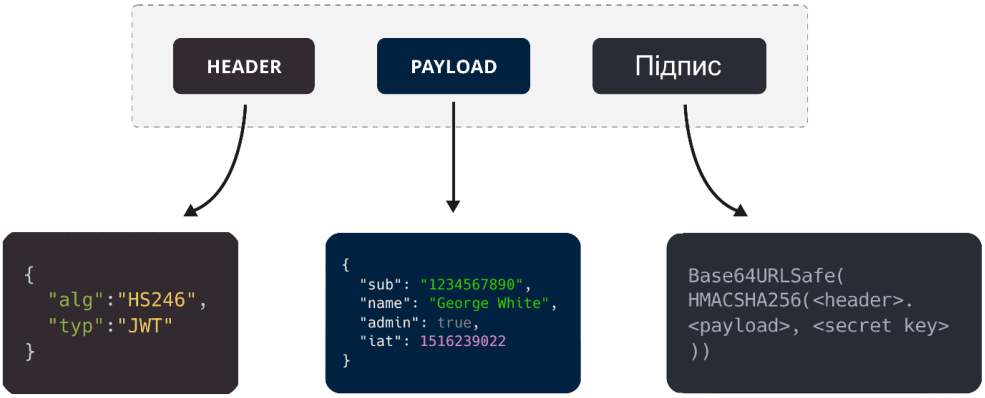


Рисунок 1.8 Структура JWT токена

1.11.4 Bcrypt

Bcrypt – це криптографічна функція хешування паролів, використовується для захисту паролів від атак методом перебору (brute force) або використання радужних таблиць (rainbow tables), які прискорюють злом паролів, порівнюючи їх хеші.

Bcrypt використовує адаптивний механізм – це означає, що час обчислення хешу можна налаштувати, щоб збільшувати складність обчислень у міру розвитку обчислювальних потужностей комп'ютерів. Таким чином, bcrypt залишається ефективним захистом навіть з часом, бо при зростанні потужності атакуючих збільшується і час обчислення, що ускладнює злому паролів. В основі bcrypt лежить алгоритм Blowfish, який застосовується для побудови хешу пароля. При цьому до пароля додається унікальна випадкова "сіль" (salt), що унеможливлює використання готових таблиць для відновлення паролів. Кожен раз, коли bcrypt хешує пароль, він генерує нову сіль і виконує багато раундів обчислень, що робить хеш унікальним навіть для однакових паролів. Формат збереження хешу bcrypt включає в себе інформацію про використаний алгоритм, параметр складності (cost factor), сіль і власне хеш (рис.1.9). При перевірці пароля система повторно хешує введений пароль з використанням тієї ж солі і параметра складності, і порівнює результат із збереженим хешем. bcrypt широко застосовується у веброзробці, системах аутентифікації, базах даних користувачів для забезпечення безпеки паролів.

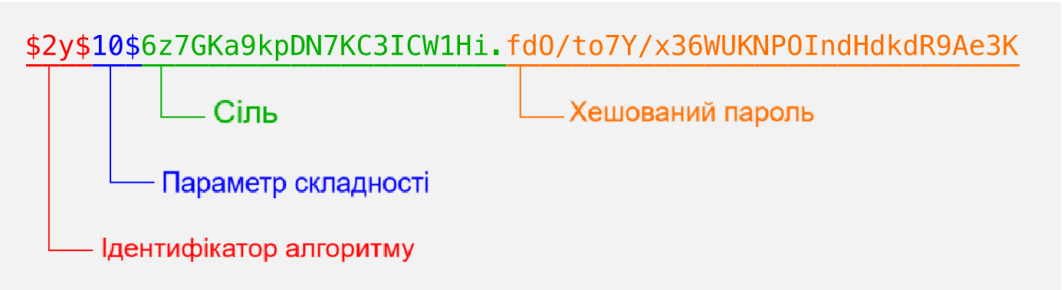


Рисунок 1.9 Структура захешованих даних за допомогою bcrypt

Переваги bcrypt полягають у високій стійкості до атак, адаптивності складності і простоті використання у різних мовах програмування через наявність численних бібліотек.

1.12 Розробка web-застосунку

Метою розробки є створення веб-застосунку, що забезпечує надійний механізм аутентифікації користувачів з використанням сучасних криптографічних методів та з дотриманням принципів безпечного веб-програмування.

У межах розробки передбачається вирішення таких основних проблем: забезпечення захищеної реєстрації та авторизації користувачів, захист персональних даних та облікової інформації від несанкціонованого доступу, запобігання спробам автоматизованого злому (brute-force, dictionary attack), гарантування стійкості паролів і безпечного зберігання їх у базі даних, надання доступу до певних ресурсів тільки для авторизованих користувачів або користувачів з відповідними ролями.

Для досягнення цієї мети застосовується декілька функціональних компонентів. Перша з яких це форма реєстрації користувача, де користувач створює обліковий запис, вводячи пароль та ім’я.

Наступний компонент це форма входу. Користувач вводить облікові дані для входу. Верифікація здійснюється шляхом порівняння введеного пароля з хешованим у базі даних. У випадку успішної перевірки генерується токен доступу на основі стандарту JWT, підписаний RSA-приватним ключем.

Реалізується система авторизації, яка визначає доступ користувача до різних частин застосунку відповідно до його ролі, “user” або “admin”. Всі запити до захищених маршрутів перевіряються на наявність та коректність JWT.

1.12.1 Розробка клієнстької частини

Клієнтська частина веб-застосунку виконує роль інтерфейсу взаємодії користувача з системою. Основна мета клієнтської частини – забезпечити зручний, інтуїтивно зрозумілий та функціональний графічний інтерфейс для реєстрації, входу в систему та взаємодії з обліковими даними.

Створимо дві блок-схеми алгоритму роботи клієнтської частини. Перша блок-схема відображає алгоритм реєстрації (рис. 1.10).

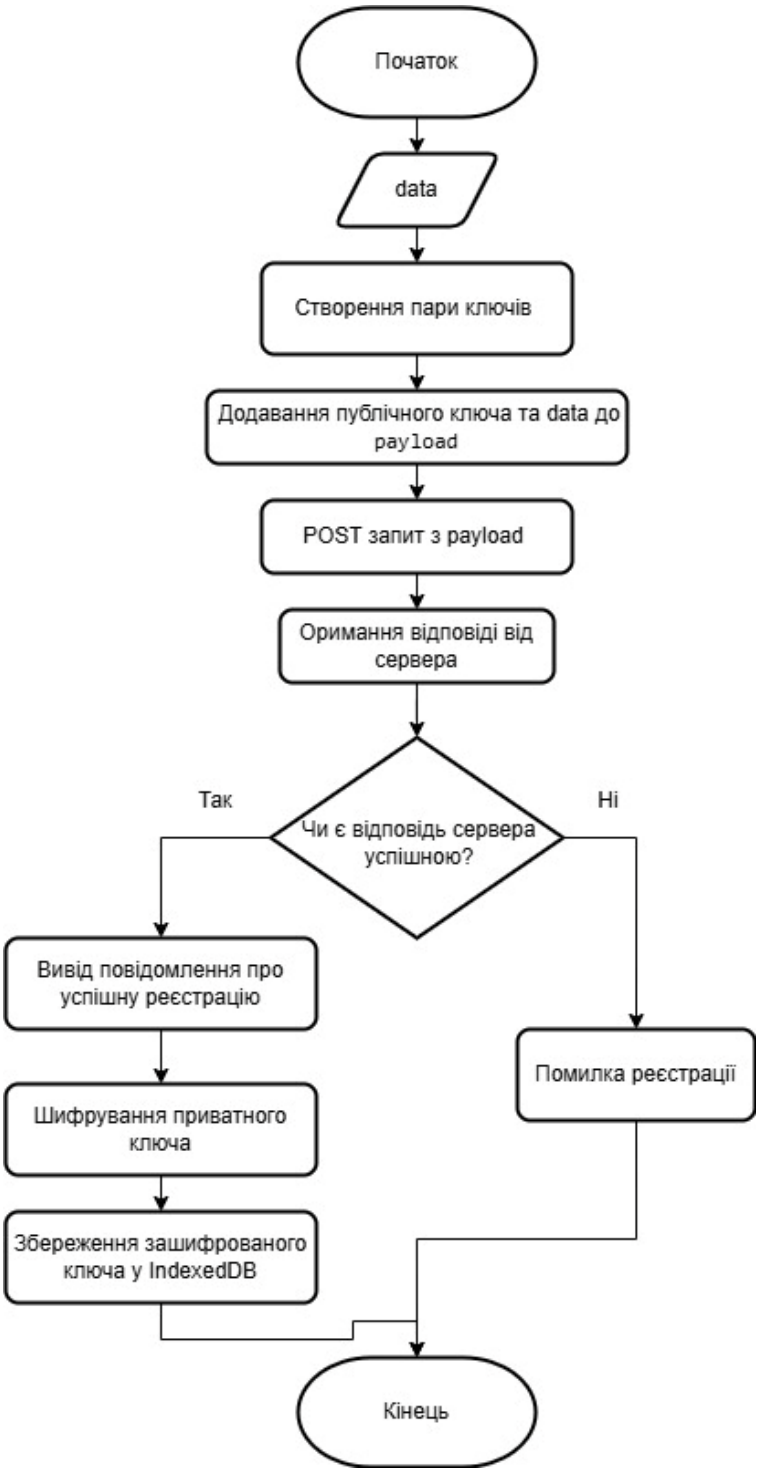


Рисунок 1.10 Блок-схема алгоритму реєстрації

Користувач вводить свої данні, створюється пара ключів RSA алгоритму, створений публічний ключ відправляється на сервер разом з введеними даними користувача. Якщо відповідь сервера є успішною, виводиться повідомлення про успішну реєстрацію та зберігається зашифрований приватний ключ. Якщо відповідь серверу є не успішною або не отриманою, це означає що сталась помилка реєстрації.

Створимо другу блок-схему, яка буде відображати алгоритм входу для клієнтської частини (рис. 1.11).

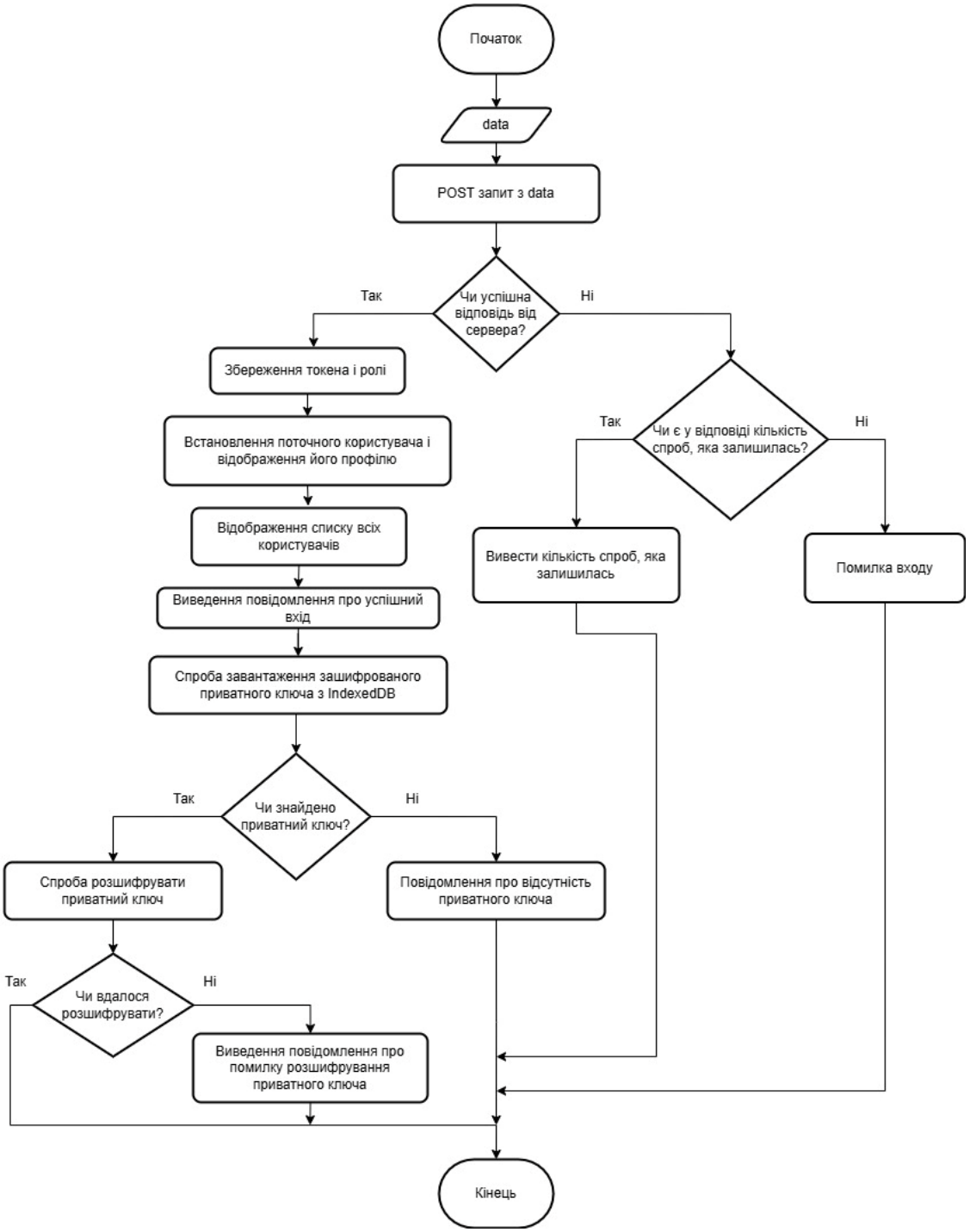


Рисунок 1.11 Блок-схема алгоритму входу

Користувач вводить свої данні, які відправляються на сервер. Якщо відповідь від серверу є успішною, застосунок зберігає токен і роль у sessionStorage, відображає профіль поточного користувача та список всіх користувачів. Також виводиться повідомлення про успішний вхід та перевіряється наявність

приватного ключа, якщо приватний ключ знайдено, відбувається спроба його розшифрування, якщо спроба не є успішною, відображається помилка розшифрування приватного ключа. Якщо відповідь від сервера є не успішною, перевіряється чи наявна у відповіді кількість спроб яка залишилась, якщо наявна, вона відображається користувачеві. Якщо не наявна, це означає що сталась помилка входу.

Створимо інтерфейс web-застосунку, який буде включати в себе компоненти вводу, кнопки і заголовок (рис. 1.12). Також зробимо мінімалістичні стилі для зрозумілості і зручності використання застосунку.

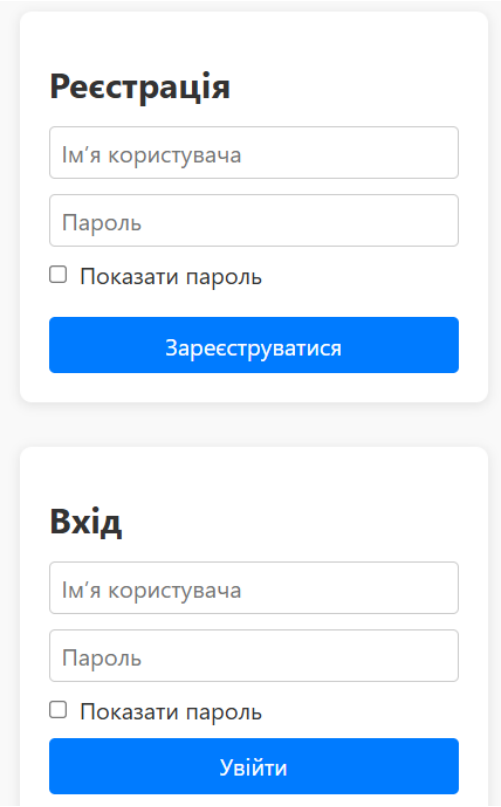


Рисунок 1.12 Інтерфейс форм реєстрації та входу

При введені паролю відображається інформація про його надійність (рис. 1.13), якщо пароль не відповідає вимогам надійності, реєстрація не відбувається.

За базою створених блок-схем, розробимо код клієнтської частини з відповідними потрібними функціями, які будуть виконувати шифрування, відправку даних на сервер, перевірку пароля на надійність, відображати потрібну інформацію користувачеві (наприклад рівень надійності пароля) та відображати потрібний інтерфейс (рис. 1.14 - 1.15).

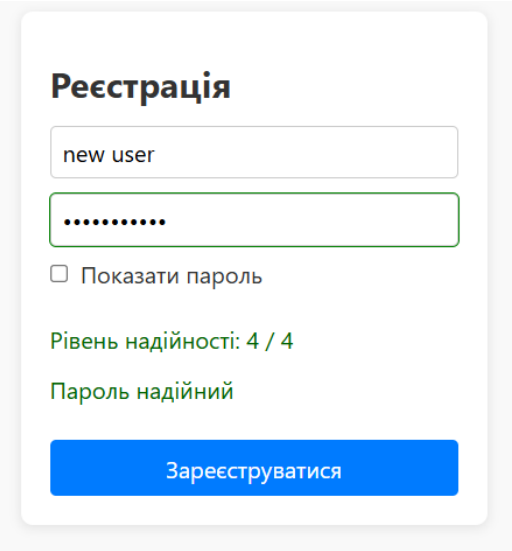


Рисунок 1.13 Демонстрація відображення надійності пароля

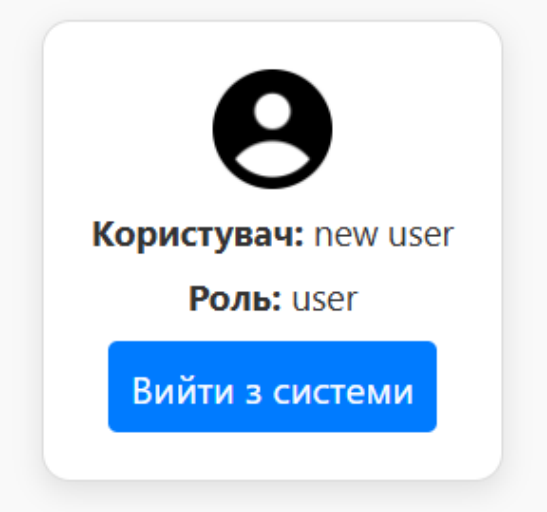


Рисунок 1.14 Профіль користувача

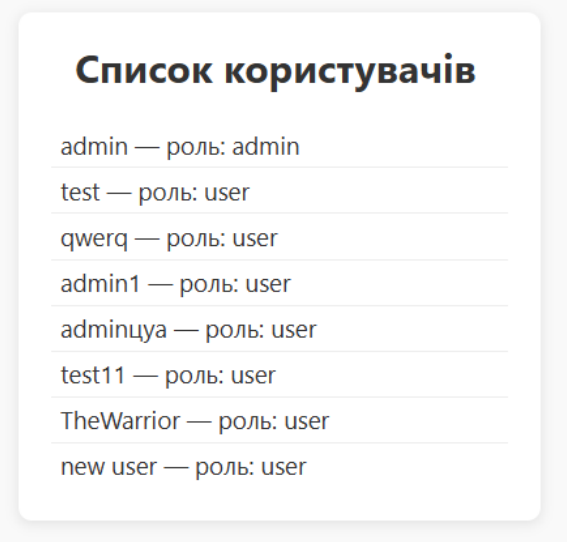


Рисунок 1.15 Вигляд списку користувачів для звичайного користувача

У створеному web-застосунку передбачені дві ролі: користувач та адміністратор. Якщо користувач який увійшов в аккаунт є адміністратором, то в такому випадку у загальному списку всіх користувачів також відображені кнопки видалення навпроти кожного користувача окрім адміністратора (рис. 1.16).

Було створено односторінковий застосунок з формою для реєстрації та входу. При успішному вході відображається профіль користувача з ім’ям та роллю, а також загальний список всіх існуючих користувачів.

Таким чином, клієнтська частина забезпечує необхідну функціональність для взаємодії користувача з системою, а також підвищує безпеку шляхом реалізації локальної валідації та дотримання принципів безпечної взаємодії з API.

Функцію видалення користувача було створено на основі блок-схеми на рис. 1.17. Виводиться повідомлення підтвердження, якщо видалення підтверджене, робить запит на сервер. Отримується відповідь про успішне чи не успішне видалення.

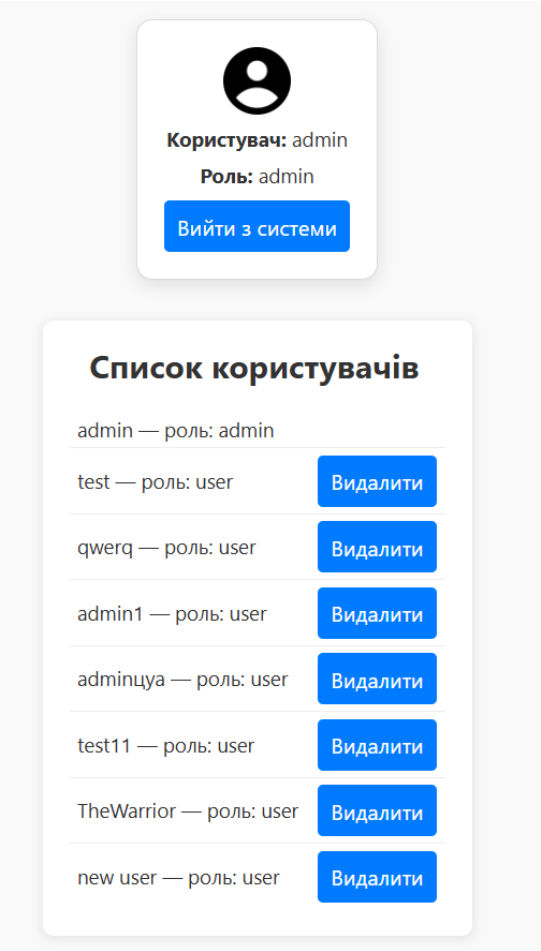


Рисунок 1.16 Відображення списку всіх користувачів для адміністратора

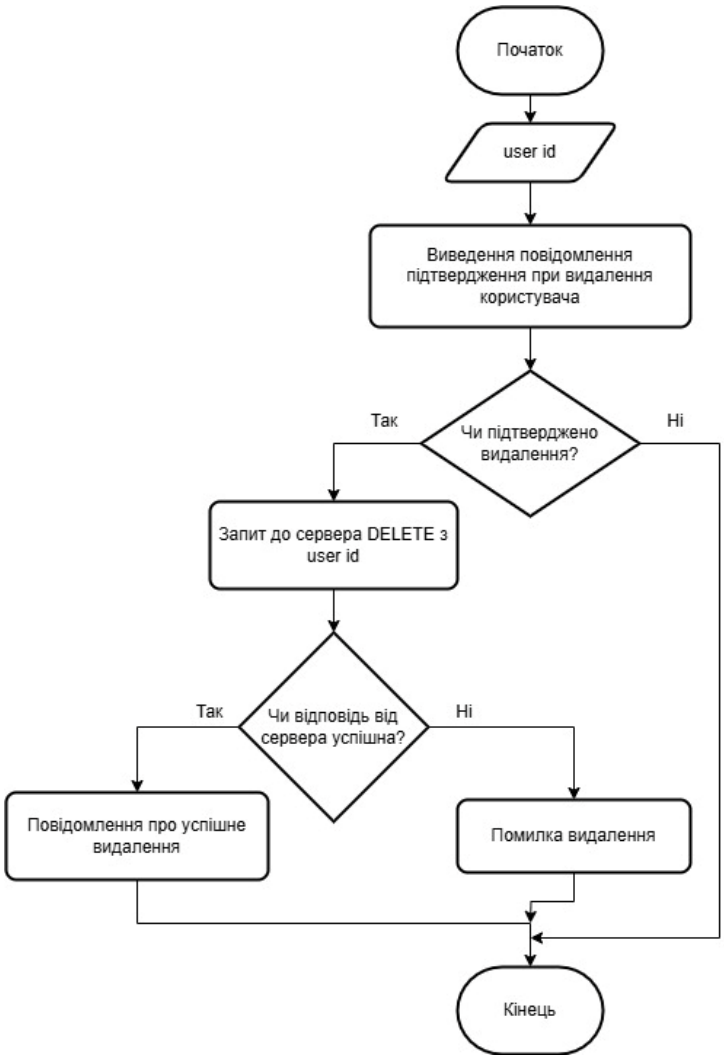


Рисунок 1.17 Блок-схема алгоритму видалення користувача

1.12.2 Розробка серверної частини

Серверна частина веб-застосунку відповідає за логіку, збереження та обробку даних, а також за забезпечення аутентифікації та авторизації користувачів. Вона приймає запити від клієнтської частини, взаємодіє з базою даних і повертає відповідні відповіді.

Створимо блок-схему алгоритму реєстрації користувачів (рис. 1.18). Клієнтська частина надсилає username, password, publicKey. Необхідно проводити декілька перевірок: чи username довше 3-х символів, чи отриманий publicKey, чи password більше 8-ми символів, чи достатньо надійний пароль та чи існує вже користувач з введеним ім’ям. Якщо всі перевірки пройдено, пароль хешується та створюється новий користувач і зберігається у базі даних.

Створимо блок-схему алгоритму входу користувачів у профіль (рис. 1.19).

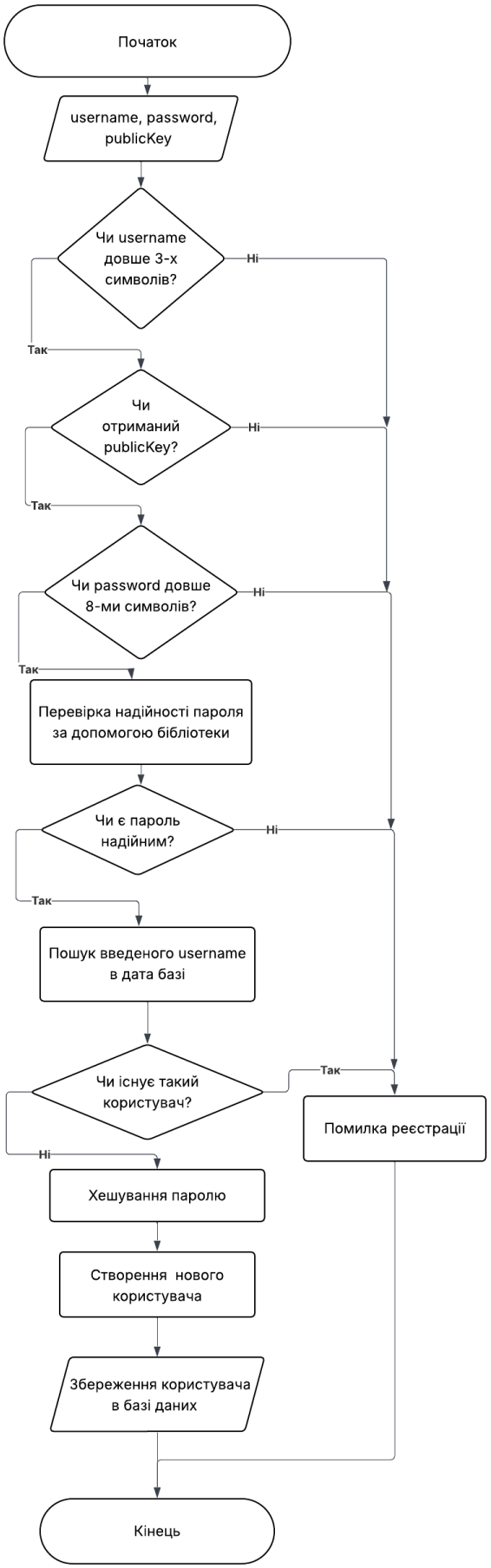


Рисунок 1.18 Блок-схема алгоритму реєстрації на серверній частині

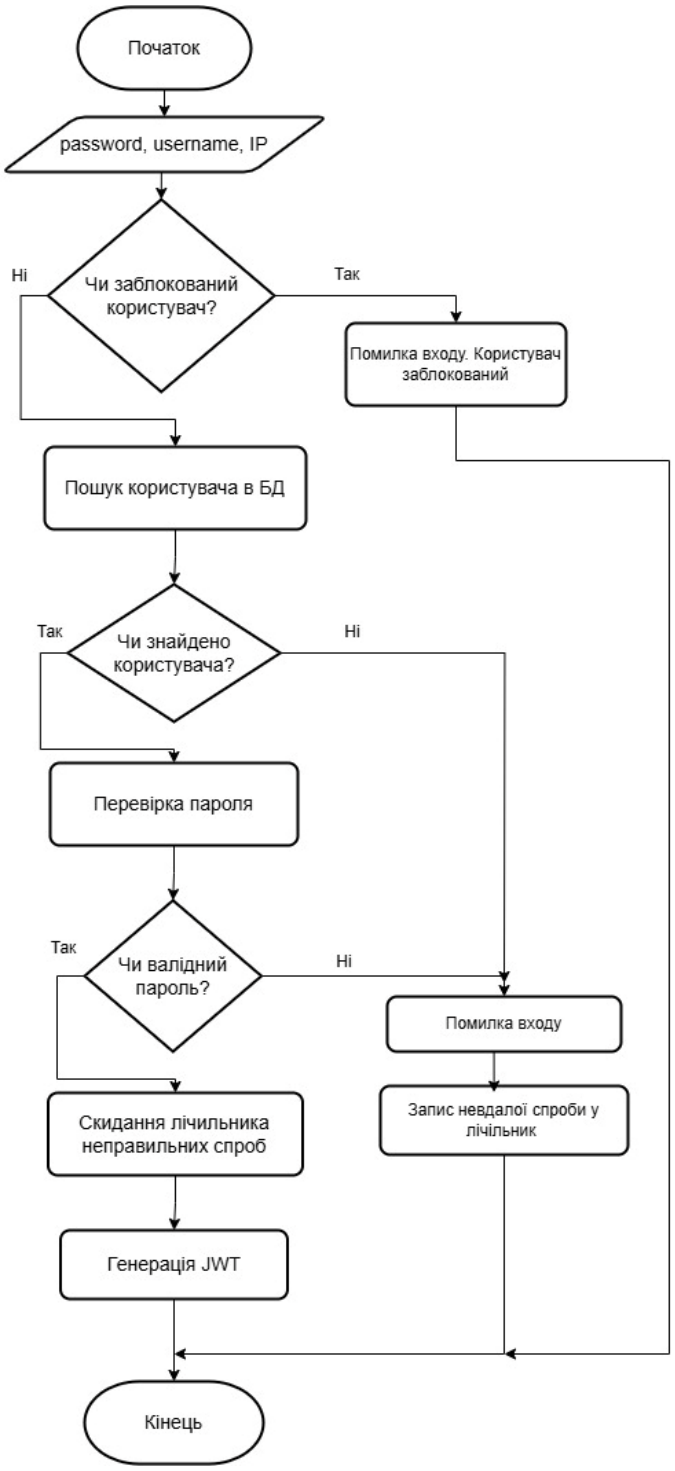


Рисунок 1.19 Блок-схема алгоритму входу на серверній частині

В першу чергу перевіряємо чи заблокований користувач, тобто чи вичерпані спроби входу. Якщо користувач не заблокований, далі йде пошук користувача у базі даних та перевірка паролю. Якщо пароль неправильний або такого користувача не знайдено, кількість доступних спроб зменшується на одну. Якщо всі спроби вичерпано, користувач не зможе повторно здійснити вхід до акаунта протягом певного часу.

Також реалізуємо видалення користувача адміністратором, на основі блок схеми на рис. 1.20 напишемо логіку яка буде валідувати токен користувача і перевіряти чи є користувач адміністратором. Реалізуємо перевірку чи намагається користувач видалити самого себе.

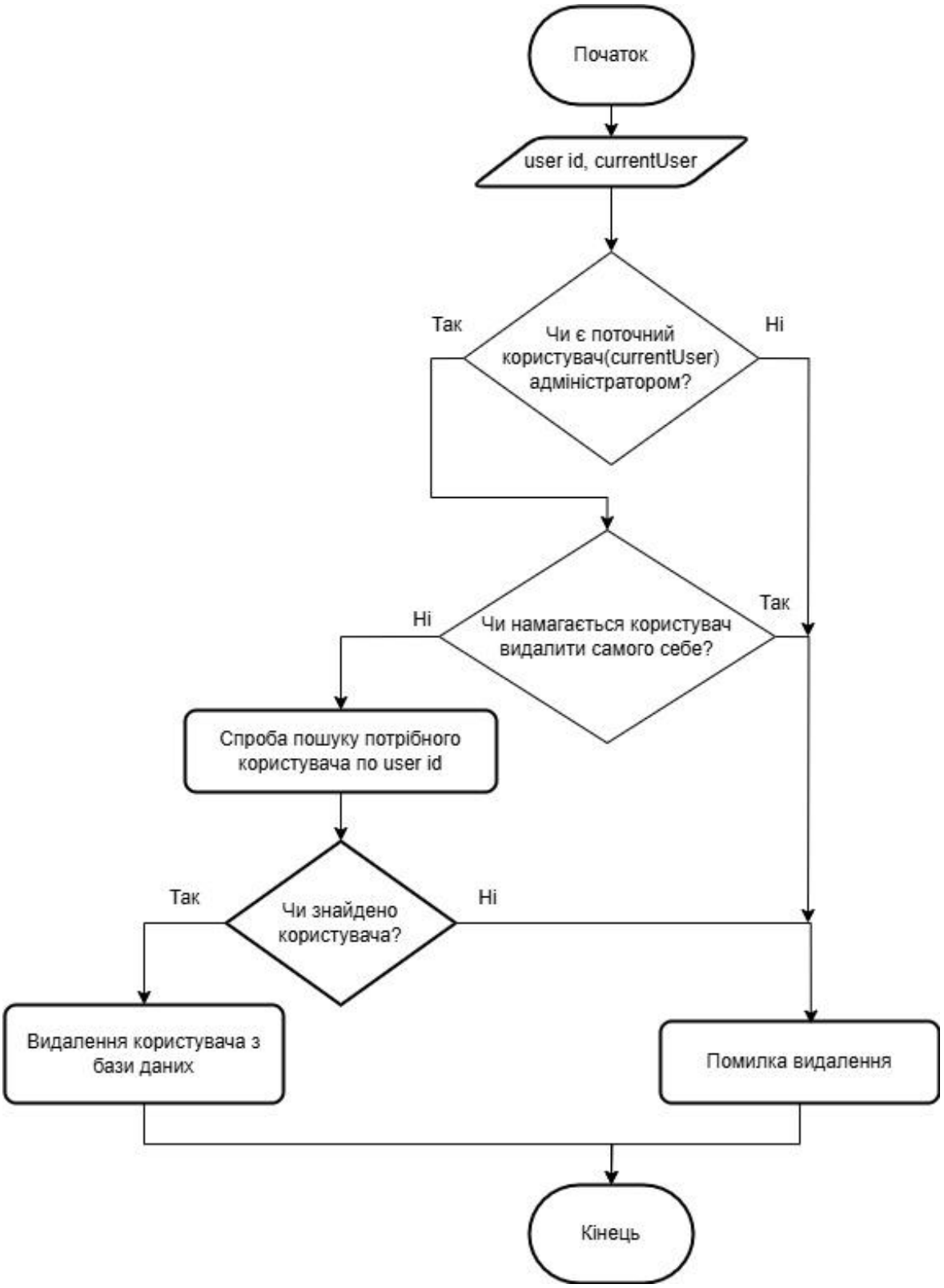


Рисунок 1.20 Блок-схема алгоритму видалення користувача на серверній частині

1.13 Перевірка захищеності системи

1.13.1 Захист від атак на паролі

Існує багато різних типів атак на паролі. Наприклад атаки за словником, вони використовують поширені слова або фрази для компрометації облікових даних користувача. Щоб запобігти злому пароля за допомогою атаки за словником, найкраще використовувати випадкові літери, цифри та символи, які не утворюють справжнього слова.

Розглянемо найбільш поширені методи атак на паролі.

1. Атаки brute force. Вони використовують методи спроб і помилок для вгадування облікових даних для входу, ключів безпеки або інших конфіденційних даних.

2. Кейлогери – це шкідливе програмне забезпечення, яке проникає в пристрій через точку доступу. Заражене програмне забезпечення, електронні листи, файли або хмарні програми можуть служити точками входу для кейлогера. Коли кейлогер проникає в пристрій користувача, він записує кожне натискання клавіші, щоб зібрати конфіденційну інформацію, таку як облікові дані для входу. Кейлогери надзвичайно небезпечні, оскільки їх важко виявити. Вони можуть легко виявити повторне використання паролів, розкриваючи облікові дані для входу численним обліковим записам.

3. Розпилення паролів (Password Spraying), відбувається, коли кіберзлочинець намагається отримати доступ до кількох облікових записів в одному домені, використовуючи поширені паролі (рис. 1.21). Зловмисник може отримати доступ до сотень облікових записів лише з однієї спроби, якщо він використовує список популярних слабких паролів, таких як 123456 або 111111.

4. Переписування облікових даних – це тип кібератаки, коли кіберзлочинці збирають та повторно використовують уже скомпрометовані дані облікових записів, щоб отримати доступ до нових облікових записів або онлайн-сервісів.

Розуміння поширених атак на паролі та загроз – це перший крок до їх запобігання.

Рисунок 1.21 Схема атаки розпиленням паролів

Застосуємо наступні методи для забезпечення безпеки паролів у web-застосунку:

1. Паролі зберігаються тільки у захешованому вигляді;
2. Впроваджена перевірки надійності пароля і неможливість реєстрації зі слабким паролем;
3. Обмежена кількість спроб входу для уникнення brute force атак;
4. Обмін даними тільки через протокол HTTPS.

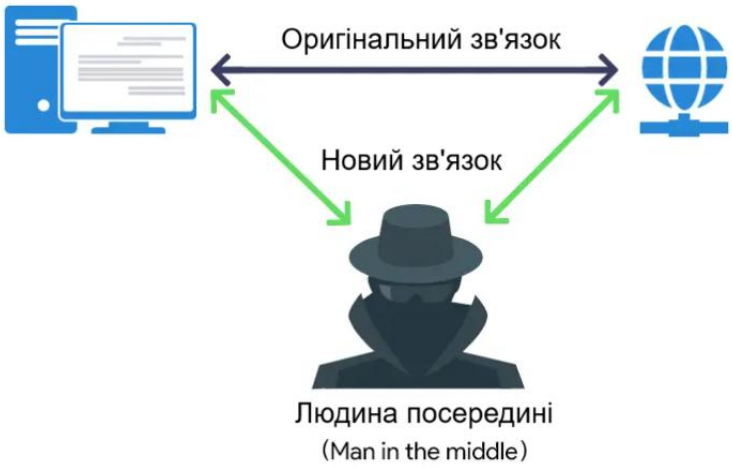
1.13.2 Захист від MITM (Man-In-The-Middle) атак

Атака типу «людина посередині» (MITM) – це загальний термін, який означає, що зловмисник позиціонує себе в розмові між користувачем і застосунком позиціонує себе як одну із сторін для підслуховування (рис. 1.22).

Інформація, отримана під час атаки, може бути використана для багатьох цілей, включаючи крадіжку особистих даних, несанкціоновані перекази коштів або незаконну зміну пароля.

Першим кроком зловмисник перехоплює трафік користувача через мережу, перш ніж він досягне цільового пункту призначення. Найпоширеніший спосіб зробити це – пасивна атака, під час якої зловмисник робить безкоштовні, шкідливі точки доступу Wi-Fi доступними для громадськості. Зазвичай вони називаються відповідно до їхнього розташування та не захищені паролем. Після того, як жертва підключається до такої точки доступу, зловмисник отримує повну видимість будь-

якого онлайн-обміну даними.



1.22 Схема атаки людини посередині

Методи MITM включають в себе:

1. Підміна IP-адреси передбачає маскування зловмисника під програму шляхом зміни заголовків пакетів в IP-адресі. В результаті користувачі, які намагаються отримати доступ до URL-адреси, підключеної до програми, перенаправляються на веб-сайт зловмисника;

2. ARP-спуфінг – це процес зв’язування MAC-адреси зловмисника з IP-адресою легітимного користувача в локальній мережі за допомогою підроблених ARP-повідомлень. В результаті дані, надіслані користувачем на IP-адресу хоста, передаються зловмиснику;

3. DNS-спуфінг, також відомий як отруєння кешу DNS, передбачає проникнення на DNS-сервер та зміну адресного запису веб-сайту. В результаті користувачі, які намагаються отримати доступ до сайту, перенаправляються зміненим DNS-записом на сайт зловмисника.

Запобіжними заходами проти MITM для підвищення безпеки web-застосунку є:

1. Шифрування даних за допомогою RSA та AES. При реєстрації генерується RSA-ключова пара: публічний ключ надсилається на сервер, а приватний – шифрується з використанням AES (Advanced Encryption Standard) на стороні клієнта. Це означає, що навіть якщо зловмисник перехопить передані дані, вони

будуть зашифровані та непридатні для використання без знання пароля та сольового значення (salt). Такий підхід унеможливлює доступ до приватних ключів без дешифрування, навіть якщо хтось втрутиться в передачу даних;

2. Передача даних лише через HTTPS, тобто всі дані між клієнтом і сервером зашифровані транспортним рівнем TLS/SSL. Всі дані, що передаються між клієнтом і сервером, шифруються. Це унеможливлює їх прочитання сторонніми особами. Також дані не можуть бути змінені або пошкоджені під час передачі без виявлення. Сертифікат HTTPS підтверджує справжність сайту, щоб користувач міг бути впевнений, що підключився до правильного сервера;

3. Використання JWT з обмеженим часом дії токена. Клієнт зберігає токен у sessionStorage і додає його до заголовку Authorization для всіх захищених запитів. Сервер перевіряє підпис токена при кожному запиті і, якщо все вірно, дозволяє доступ до ресурсів.

1.13.3 Запобігання CSRF (Cross-Site Request Forgery)

Підробка міжсайтових запитів (також відома як CSRF) – це вразливість веб-безпеки, яка дозволяє зловмиснику спонукати користувачів виконувати дії, які вони не мають наміру виконувати. Це дозволяє зловмиснику частково обійти ту саму політику походження, яка розроблена для запобігання взаємодії різних веб-сайтів.

Під час успішної атаки CSRF зловмисник змушує користувача-жертву ненавмисно виконати дію (рис. 1.23). Наприклад, це може бути зміна адреси електронної пошти в його обліковому записі, зміна пароля або здійснення переказу коштів. Залежно від характеру дії, зловмисник може отримати повний контроль над обліковим записом користувача. Якщо скомпрометований користувач має привілейовану роль у програмі, то зловмисник може отримати повний контроль над усіма даними та функціональністю програми.

У програмі існує дія, яку зловмисник має підстави ініціювати. Це може бути привілейована дія (наприклад, зміна дозволів для інших користувачів) або будь-яка дія з даними, що стосуються користувача (наприклад, зміна власного пароля користувача).

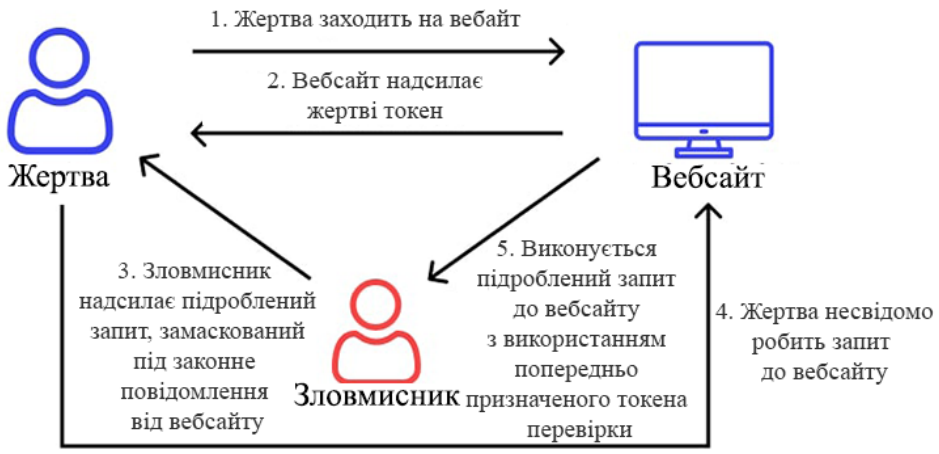


Рисунок 1.23 Схема атаки підробкою міжсайтових запитів

Для запобіганню атак CSRF використаємо наступні методи:

1. Відсутність автоматичного відправлення токенів браузером: на відміну від традиційних сесійних cookie, які браузер автоматично додає до кожного запиту на сервер, токени JWT у sessionStorage не додаються автоматично. Вони передаються лише тоді, коли клієнтський код явно додає їх у заголовок Authorization запиту. Це унеможливлює автоматичне використання токена сторонніми сайтами для виконання підроблених запитів.

2. Явна авторизація запитів: всі запити, які потребують автентифікації, містять у собі заголовок Authorization: Bearer з токеном JWT. Це означає, що атака CSRF, яка базується на примусовому виконанні запитів від імені користувача, не матиме доступу до цього токена і не зможе авторизувати підроблений запит.

3. Перевірка заголовка Origin на сервері: додатково на серверній стороні реалізована перевірка заголовка Origin у вхідних запитах. Ця перевірка дозволяє впевнитися, що запити надходять тільки з дозволених доменів, наприклад тільки з домена web-застосунку, і відкидає запити, які походять з інших сайтів. Це ефективно запобігає виконанню підроблених запитів із зовнішніх джерел, що значно підвищує захист від CSRF.

1.13.4 Запобігання XSS (Cross-Site Scripting)

Атаки міжсайтового скриптингу (XSS) – це тип ін'єкції, під час якого шкідливі скрипти впроваджуються на безпечні та надійні веб-сайти. XSS-атаки

трапляються, коли зловмисник використовує web-застосунок для надсилання шкідливого коду, зазвичай у формі скрипта на стороні браузера, іншому кінцевому користувачеві (рис. 1.24). Недоліки, які дозволяють цим атакам досягати успіху, досить поширені та трапляються будь-де, де веб-додаток використовує вхідні дані від користувача у виведених даних, які він генерує, не перевіряючи та не кодуючи їх.

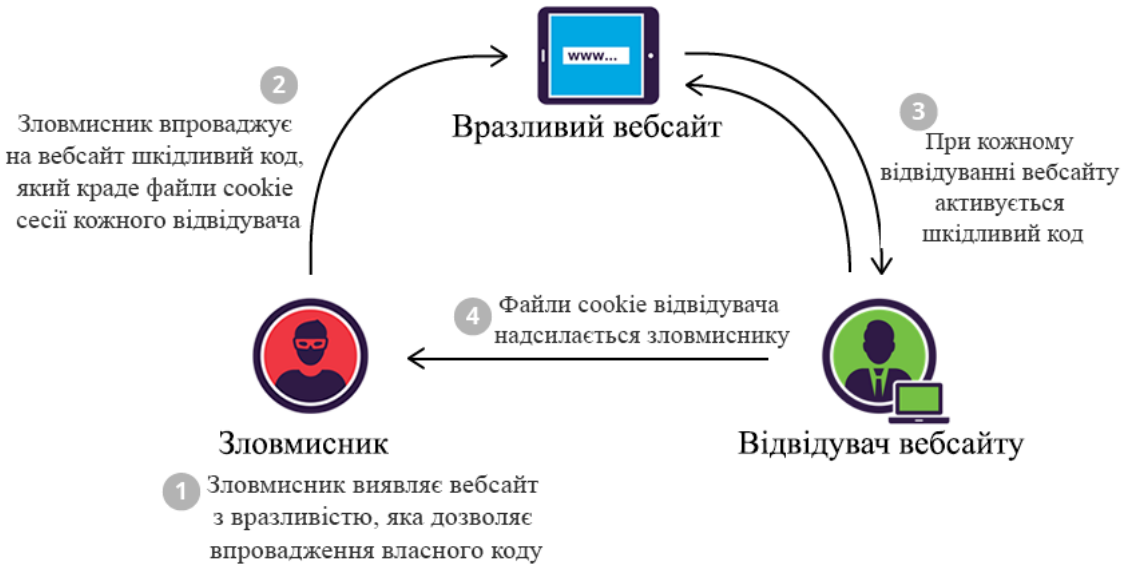


Рисунок 1.24 Схема атаки XSS

Браузер кінцевого користувача не має можливості дізнатися, що скрипту не слід довіряти, і виконає скрипт. Оскільки він вважає, що скрипт походить з надійного джерела, шкідливий скрипт може отримати доступ до будь-яких файлів cookie, токенів сеансу або іншої конфіденційної інформації, що зберігається браузером та використовується з цим сайтом. Ці скрипти можуть навіть перезаписувати вміст HTML-сторінки.

Відсутність необробленого відображення користувацьких даних: на клієнтській стороні застосовується безпечне вставлення контенту, зокрема за допомогою методів, які не інтерпретують введений текст як HTML або JavaScript, використовується textContent замість innerHTML. Це гарантує, що навіть якщо у введених даних залишилися скрипти, вони не будуть виконані у браузері.

Використання безпечних заголовків HTTP: сервер налаштований на додавання таких заголовків, як Content-Security-Policy (CSP), які обмежують

джерела виконуваного коду, дозволяючи запускати лише перевірені скрипти з надійних доменів. CSP ефективно знижує ризик виконання ін’єкційних скриптів і перешкоджає XSS-атакам.

Валідація та санітизація введених даних: всі дані, які користувачі вводять у форми або інші поля, проходять обов’язкову валідацію на відповідність очікуваному формату.

Використання сучасних бібліотек і фреймворків з вбудованим захистом: у проєкті застосовується JavaScript у поєднанні з безпечними методами обробки DOM, що зменшує ризик XSS. При роботі з динамічним контентом завжди використовується правильне екранування та шаблонізація.

Таким чином, захист від XSS досягається завдяки поєднанню правильного контролю над введеними даними, безпечним методам відображення контенту, використанню політик безпеки на рівні HTTP-заголовків і суворому контролю доступу. Це забезпечує надійний захист від впровадження та виконання шкідливих скриптів у браузері користувачів.

2 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

Роботу присвячено розробці web-застосунку для аутентифікації користувачів з використанням методів криптографічного захисту даних, зокрема алгоритмів асиметричного шифрування RSA та технології JSON Web Token (JWT) для забезпечення безпечної передачі та збереження даних під час взаємодії клієнта з сервером. Розроблений застосунок реалізує базові механізми реєстрації, входу та перевірки прав доступу користувачів, що є важливим етапом у створенні захищених інформаційних систем.

У межах економічного розділу було здійснено аналіз витрат, пов’язаних із створенням, впровадженням, а також подальшою експлуатацією розробленого web-застосунку. Особливу увагу приділено визначенню основних статей витрат на кожному етапі життєвого циклу системи, таких як: розробка дизайну, написання програмного коду, реєстрація доменного імені, розгортання сайту на хостингу та витрати на його підтримку в робочому стані.

2.2 Розрахунок економічної ефективності

2.2.1 Розрахунок витрат на розробку

Для визначення витрат на розробку ($В_{р)}$ сайту необхідно розрахувати оплату праці виконавців, безпосередньо притягнених до її виконання.

Витрати на розробку сайту ($В_р$) є одноразовими та складаються з вартості наступних видів робіт зі створення сайту : розробка макетів дизайну для сторінок сайту; розробка фірмового стилю; підготовка сторінок-шаблонів; наповнення та форматування web-сторінок; верстка (переклад в HTML-формат) web-сторінок; створення програмного коду сайту, програмування динамічних елементів (анімаційних елементів, флеш-заставок), серверної на клієнтської частини.

Розрахуємо трудомісткість розробки сайту за складеним план-графіком по розробці web-застосунку і тривалості виконання робіт. Розподіл робіт по етапах і видах виконавців наведено в таблиці 2.1.

Таблиця 2.1 – План-графік по розробці web-застосунку

№	Назва етапу	Час виконання (годин)	Посада виконавця
1	Дизайн макетів	5	Студент
2	Верстка шаблонів	10	Студент
3	Програмування логіки	50	Студент
	ВСЬОГО:	65	

Розмір заробітної плати розраховується виходячи з чисельності різних категорій виконавців, трудомісткості, що витрачається ними на виконання різних видів робіт, а також їх середньої заробітної плати за годину.

Витрати на заробітну плату приведені в таблиці 2.2.

Таблиця 2.2 – Витрати на заробітну плату

№	Персонал	Етапи розробки	Кількість робочих годин	Погодинна ставка, грн.	Заробітна плата, грн.
1	Студент	Дизайн макетів	2	60	120
2	Студент	Верстка шаблонів	10	60	600
3	Студент	Програмування логіки	50	60	3000
				ВСЬОГО:	$В_{зп}$ = 3720

До складу витрат на оплату праці також включаються податки, збори і інші обов'язкові платежі, встановлені системою оподаткування що діє. Розмір єдиного соціального внеску складає 22% від заробітної плати, розраховується за наступною формулою:

$$В_{єсв} = В_{зп} * 0,22 = 3720 * 0,22 = 818,4 \text{ грн}$$

Загальні витрати ($В_р$) на розробку веб-сайту розраховуються як сума витрат на заробітну плату праці персоналу ($В_{зп}$) та єдиного соціального внеску ($В_{єсв}$):

$$В_р = В_{зп} + В_{єсв} = 3720 + 818,4 = 4538,4 \text{ грн}$$

2.2.2 Розрахунок витрат на впровадження та експлуатацію

Витрати на впровадження web-застосунку ($В_в$) складаються витрат на реєстрацію доменного імені на 1 рік ($В_{в1}$) та витрат на реєстрацію в пошукових системах ($В_{в2}$). Вартість реєстрації доменного імені складає 350 грн. Витрати на реєстрацію в пошукових системах є безкоштовним, тож витрати на реєстрацію становлять 0 грн. Витрати на впровадження web-застосунку ($В_в$) розраховуються по формулі:

$$В_в = В_{в1} + В_{в2} = 350 + 0 = 350 \text{ грн}$$

Роботи по підтримці сайту в робочому стані включають в себе: оновлення контенту; виправлення помилок; резервне копіювання; моніторинг працездатності сайту. Орієнтовно співробітнику потрібно виділяти 1 годину в місяць на підтримку сайту. Розрахуємо витрати на підтримку сайту в робочому стані ($В_{підтр}$) на рік з урахуванням погодинної ставки фахівця у розмірі 60 грн/год:

$$В_{підтр} = 1 \text{ год/міс} * 12 \text{ міс} * 60 \text{ грн/год} = 720 \text{ грн}$$

Витрати на експлуатацію web-застосунку ($В_е$) включають вартість робіт з підтримки сайту в робочому стані і вартість послуг по продовженню доменного імені на 1 рік.

$$В_в = 350 + 720 = 1070 \text{ грн}$$

У таблиці 2.3 визначаються постійні витрати як сума витрат на впровадження та експлуатацію сайту протягом року.

Таблиця 2.3 – Постійні витрати

№	Стаття витрат	Вартість за рік, грн.
1	Реєстрація доменного імені на 1 рік ($В_{в1}$)	350
2	Реєстрація в пошукових системах ($В_{в2}$)	0
3	Підтримка сайту в робочому стані ($В_{підтр}$)	720
4	Продовження доменного імені на наступний рік	350
	Всього:	$В_{пост}$ = 1420 грн

Загальні витрати ($В_з$) на розробку, впровадження та експлуатацію веб-сайту розраховуються за наступною формулою:

$$В_з = В_р + (В_в + В_е) = 4538,4 + (350 + 1070) = 5958,4 \text{ грн}$$

2.2.3 Розрахунок коефіцієнту економічної ефективності

Економічна ефективність за рік ($Е_р$) визначається як сукупність коштів, вивільнених за рахунок впровадження сайту:

$$Е_р = Е_{р1} + Е_{р2}$$

Вивільнення коштів може бути досягнуто за рахунок:

$Е_{р1}$ - скорочення коштів, що витрачаються на розміщення реклами на телебаченні і в періодичній пресі, а також на виготовлення і поширення прайс-листів, візитних карток, буклетів тощо.

$Е_{р2}$ - скорочення засобів за рахунок рішення окремих бізнес-завдань за допомогою сайту.

Загальна економія за рік склала: Ер1 = 2500 грн, Ер2 = 8000 грн.

Загальна економічна ефективність:

$$Е_р = Е_{р1} + Е_{р2} = 10500 \text{ грн}$$

Коефіцієнт економічної ефективності за перший рік вираховується таким чином:

$$К_е = Е_р / В_з = 10500 / 5958,4 = 1,8$$

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

Охорона праці є важливою складовою будь-якої професійної діяльності, спрямованої на забезпечення безпеки, комфортних умов праці, збереження життя, здоров'я та працездатності працівників під час виконання їхніх обов'язків. В умовах швидкого розвитку інформаційних технологій, автоматизації виробничих процесів і зростання інтелектуального навантаження особливе значення набувають питання безпеки праці. Це стосується розробки програмного забезпечення, роботи з комп'ютерною технікою, мережами та іншими цифровими інструментами.

Метою заходів із охорони праці в галузі інформаційних технологій є створення умов, які мінімізують вплив шкідливих і небезпечних чинників виробництва, попереджають професійні захворювання та знижують ризики травматизму. Особливу увагу варто приділяти організації робочого місця програміста: забезпеченню ергономіки, правильному освітленню, дотриманню балансу між працею та відпочинком, урахуванню психофізіологічних аспектів, а також питанням електробезпеки, пожежної безпеки та безпечного використання офісного обладнання.

Цей розділ охоплює основні принципи охорони праці та вимоги до організації безпечного робочого місця розробника програмного забезпечення. Розглядаються потенційні загрози, шкідливі й небезпечні фактори, а також технічні та організаційні засоби для їхнього усунення або мінімізації. Також приділяється увага заходам із забезпечення пожежної та електробезпеки. Особливий акцент зроблено на нормативно-правовій базі з охорони праці, а також на впровадженні системи управління охороною праці на підприємствах.

3.1 Аналіз шкідливих та небезпечних факторів, що впливають на користувача ПК

Робота з ПК передбачає тривале перебування людини у статичній позі, зосереджену увагу, високий рівень зорового та розумового навантаження, що в сукупності створює потенційні ризики для здоров'я працівника. Насамперед

варто виділити фізіологічні шкідливі фактори. Тривале сидіння в одній позі без належної підтримки спини та зміни положення тіла призводить до порушень у роботі опорно-рухового апарату — зокрема викривлення хребта, болю в шиї, плечах і попереку. Відсутність регулярних перерв і фізичної активності також сприяє розвитку застійних явищ у судинах нижніх кінцівок та загальному фізичному виснаженню.

Важливим чинником є зорове навантаження. Тривала робота за монітором, особливо при недостатньому або надмірному освітленні, призводить до втоми очей, зниження гостроти зору, розвитку сухості очей та навіть головного болю. Надто яскраві екрани, відсутність фільтрації синього світла, неправильно налаштована яскравість монітора також негативно впливають на зорову систему користувача.

3.1 Гігієнічні вимоги до виробничого середовища

Гігієнічні вимоги до виробничого середовища спрямовані на забезпечення безпечних і комфортних умов праці, які мінімізують вплив шкідливих і небезпечних факторів на здоров’я працівників. Вони включають дотримання норм щодо параметрів мікроклімату, таких як температура, вологість і рух повітря, а також контроль за рівнем освітлення, шуму, вібрації, пилу та інших забруднювачів повітря. Виробниче середовище повинно бути організоване так, щоб забезпечувати достатню вентиляцію і очищення повітря, а також уникати підвищеної запиленість і токсичності. Важливе значення мають регулярне прибирання робочих приміщень і контроль за станом устаткування для запобігання аварій і небезпечних ситуацій. Дотримання цих вимог сприяє збереженню працездатності працівників, зниженню ризику професійних захворювань та підвищенню продуктивності праці.

3.1 Вимоги до приміщення

Для забезпечення безпечних і комфортних умов праці при роботі з персональним комп’ютером приміщення повинно відповідати ряду санітарно-гігієнічних вимог. Згідно з ДСанПіН 3.3.2-007-98, робоче приміщення повинно

мати достатній об’єм не менше 20 м³ на одного працівника та площу не менше 6 м², що забезпечує оптимальний повітрообмін і виключає скупчення шкідливих речовин. Важливим є наявність належної вентиляції для забезпечення свіжого повітря і підтримки допустимого рівня вологості (40-60%) і температури (18-22 °C). Приміщення має бути оснащене системою кондиціювання або опалення для підтримки комфортних кліматичних умов протягом року.

3.1 Вимоги до освітлення

Оптимальне освітлення на робочому місці користувача ПК відіграє важливу роль у збереженні зору і зниженні стомлюваності. Відповідно до ДСанПіН 3.3.2-007-98, робочі місця повинні бути забезпечені природним і штучним освітленням. Рівень освітленості має становити не менше 300-500 лк для офісної роботи. Рекомендується використовувати дифузне освітлення, щоб уникнути відблисків на екрані монітора. Джерела світла повинні бути розміщені таким чином, щоб світло не падало безпосередньо в очі працівника і не створювало тіней на робочій поверхні. Використання ламп з відповідною колірною температурою (4000-4500 К) забезпечує комфортне для очей світло.

3.1 Вимоги до шуму в приміщенні

Шум у робочому приміщенні не повинен перевищувати допустимі рівні, що встановлені санітарними нормами. Згідно з ДСанПіН 3.3.2-007-98, допустимий рівень шуму в офісних приміщеннях, де виконується робота, пов’язана з використанням ПК, не повинен перевищувати 50 дБ протягом робочого дня. Перевищення рівня шуму може призводити до підвищеної стомлюваності, зниження продуктивності та негативного впливу на нервову систему. Для забезпечення нормального рівня шуму рекомендується застосовувати звукопоглинальні матеріали у внутрішньому оздобленні приміщень та правильно розташовувати джерела шуму.

3.1 Вимоги до організації робочого місця працівника

Організація робочого місця працівника при роботі з персональним

комп’ютером має відповідати ергономічним, санітарно-гігієнічним і безпековим нормам. Робоче місце повинно бути достатньо просторим, щоб забезпечити вільний доступ до обладнання та комфортну позу працівника. Рекомендується використовувати регульовані за висотою стілець і стіл, що дозволяють адаптувати робоче місце під індивідуальні параметри користувача. Екран монітора має розміщуватися на відстані 50-70 см від очей на рівні очей або трохи нижче для зниження напруги на зір. Поверхня столу повинна бути матовою, щоб уникнути відблисків, а робочі аксесуари (клавіатура, миша) — розташовані так, щоб забезпечувати природне положення рук. Організація робочого місця повинна враховувати запобігання монотонності та можливість періодичних перерв для зменшення стомлюваності.

3.1 Вимоги до мікроклімату

Оптимальні параметри мікроклімату на робочому місці забезпечують комфортні умови праці і знижують ризики негативного впливу на здоров’я працівника. Згідно з гігієнічними нормами, температура повітря у приміщенні повинна підтримуватись у межах 18-22 °C, відносна вологість — 40-60%, а швидкість руху повітря — не більше 0,1 м/с. Належна вентиляція має забезпечувати постійний приплив свіжого повітря та видалення надлишкової вологи і шкідливих речовин. Відхилення від оптимальних параметрів мікроклімату можуть викликати дискомфорт, зниження працездатності та розвиток професійних захворювань.

3.1 Вимоги з електробезпеки

При роботі з персональним комп’ютером необхідно дотримуватись правил електробезпеки, щоб запобігти ураженню електричним струмом і пожежам. Всі електричні прилади повинні бути правильно заземлені і підключені до мережі відповідно до стандартів (ДСТУ, ПУЕ). Забороняється використовувати пошкоджені кабелі, розетки та подовжувачі. Робоче місце повинно бути обладнане пристроями захисту від перенапруг і коротких замикань. Працівник повинен мати базові знання з правил безпечного користування

електроприладами, уникати контакту з відкритими електричними частинами, а у випадку несправностей негайно повідомляти відповідальним особам. Важливим є дотримання правил експлуатації комп'ютерної техніки, регулярний технічний огляд і своєчасне обслуговування.

3.1 Пожежна безпека

Для забезпечення пожежної безпеки необхідно дотримуватися низки вимог, серед яких — облаштування приміщень засобами пожежогасіння (вогнегасники, пожежні крани, автоматичні системи сповіщення та гасіння пожеж), забезпечення легкого і безперешкодного доступу до евакуаційних виходів, а також підтримка належного стану цих виходів — вони мають бути чітко позначені та вільні від перешкод. Важливим елементом є правильна організація зберігання і використання легкозаймистих і горючих матеріалів, що знижує ризик виникнення пожежі. Для цього встановлюють спеціальні норми та правила, які регламентують кількість, розміщення і умови зберігання таких матеріалів.

Ще одним напрямком є регулярне проведення інструктажів, навчань та тренувань для персоналу з питань пожежної безпеки. Працівники повинні знати основні правила поведінки при пожежі, порядок користування первинними засобами пожежогасіння, маршрути евакуації та дії в надзвичайних ситуаціях.

Не менш важливою є організація систем контролю та профілактики — регулярне технічне обслуговування електрообладнання, вентиляційних систем, газового устаткування, що можуть бути джерелами займання. Проводиться моніторинг дотримання вимог пожежної безпеки, усунення виявлених порушень і недоліків. Пожежна безпека також передбачає розробку і впровадження планів евакуації та дій у разі пожежі, що дає змогу оперативно і організовано евакуювати персонал, мінімізувати втрати та збитки.

ВИСНОВКИ

У ході виконання дипломної роботи було успішно досягнуто поставленої мети — створено веб-застосунок, який забезпечує безпечну аутентифікацію користувачів із використанням сучасних криптографічних методів.

У процесі розробки проведено ґрунтовний аналіз існуючих способів захисту інформації, зокрема технологій хешування паролів та асиметричного шифрування. На основі отриманих даних обґрунтовано вибір алгоритму RSA як надійного способу шифрування, а також впровадження токенів формату JWT для реалізації механізму авторизації.

Створений веб-застосунок підтримує функції реєстрації, аутентифікації та авторизації користувачів, відповідаючи сучасним стандартам безпеки. Зокрема, впроваджено механізми перевірки складності пароля, захисту від автоматизованих атак, обмеження спроб входу, використання HTTPS, CORS, токен-базованої аутентифікації та безпечного зберігання токенів.

Окрім цього, реалізовано функціонал управління ролями користувачів і механізми протидії актуальним загрозам, зокрема XSS, CSRF, brute-force та MITM-атакам.

Отриманий результат відповідає сучасним стандартам безпеки та може послужити базовою платформою для побудови розширених систем керування доступом. Завдяки ефективному застосуванню криптографічних підходів і продуманій архітектурі роботи вдалося створити рішення, яке гарантує високий рівень захисту даних у веб-застосунках.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Бойко, А. І. Криптографічні методи захисту інформації: навч. посібник. – Львів: Львівська політехніка, 2019. – 214 с.

2. Дьяконов, В. Ю. JavaScript та AJAX: підручник. – Київ: Діалектика, 2020. – 312 с.

3. Петренко, В. І. Захист інформації в комп'ютерних системах. – Харків: ХНУРЕ, 2018. – 196 с.

4. Ferguson, N., Schneier, B., & Kohno, T. Cryptography Engineering: Design Principles and Practical Applications. – Wiley Publishing, 2010. – 384 p.

5. Stallings, W. Cryptography and Network Security: Principles and Practice. – 8th ed. – Pearson, 2023. – 752 p.

6. Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. Handbook of Applied Cryptography. – CRC Press, 1996. – 816 p.

7. RFC 8017: PKCS #1: RSA Cryptography Specifications Version 2.2 [Електронний ресурс]. – https://datatracker.ietf.org/doc/html/rfc8017 – Дата звернення: 17.03.2025.

8. Mozilla Developer Network (MDN). Web Cryptography API [Електронний ресурс]. – https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API – Дата звернення: 18.03.2025.

9. bcrypt documentation. bcrypt - password hashing for Node.js [Електронний ресурс]. – Режим доступу: https://www.npmjs.com/package/bcrypt – Дата звернення: 22.03.2025.

10. JSON Web Token. Introduction to JWT [Електронний ресурс]. – Режим доступу: https://jwt.io/introduction – Дата звернення: 22.03.2025.

11. Node.js official docs. Node.js v20.x Documentation [Електронний ресурс]. – https://nodejs.org/en/docs – Дата звернення: 22.03.2025.

12. MongoDB Inc. MongoDB Manual [Електронний ресурс]. – https://www.mongodb.com/docs/manual/ – Дата звернення: 22.03.2025.

13. Express.js. Express – Node.js web application framework [Електронний ресурс]. –https://expressjs.com/ – Дата звернення: 22.03.2025.

ДОДАТОК А. Код клієнтської частини застосунку

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link href="https://cdn.jsdelivr.net/npm/modern-normalize@3.0.1/modern-normalize.min.css"
rel="stylesheet" />
  <link rel="stylesheet" href="style.css" />
  <link rel="icon" href="/favicon.png" type="image/png" />
  <title>Create profile</title>
</head>
<body>
  <div class="wrapper">
  <div class="container">
    <h1>Реєстрація</h1>
    <form id="registerForm">
      <input type="text" name="username" placeholder="Ім’я користувача" required />
      <input id="passwInput" type="password" name="password" placeholder="Пароль" required />
      <label class="inputCheck">
      <input type="checkbox" id="passwordToggle" name="passwordToggle" />Показати пароль
      </label>
      <div id="passwStr"></div>
      <button type="submit">Зареєструватися</button>
    </form>
    <p id="sucReg" style="display: none">Реєстрація успішна!</p>
  </div>
  <div class="container">
    <h1>Вхід</h1>
    <form id="loginForm">
      <input type="text" name="username" placeholder="Ім’я користувача" required />
      <input type="password" id="loginInput" name="password" placeholder="Пароль" required />
      <label class="inputCheck">
      <input type="checkbox" id="passwordToggle2" name="passwordToggle" />Показати пароль
      </label>
      <button type="submit">Увійти</button>
    </form>
    <p id="sucLog" style="display: none">Успішний вхід!</p>
    <p id="output"></p>
  </div>
  <div id="profileCard" style="
        display: none;">
```

```
    <img src="profile.png" alt="Avatar" style="display: block; margin-bottom: 10px; width: 50px" />
    <div><strong>Користувач:</strong> <span id="profileUsername"></span></div>
    <div><strong>Роль:</strong> <span id="profileRole"></span></div>
    <button id="logoutBtn">Вийти з системи</button>
  </div>
  <div class="container">
<div class="usersList-container">
  <h1 style=" display: none" id="listHeader">
    Список користувачів
  </h1>
  <ul id="userList" class="userList"></ul>
   </div>
   </div>
  </div>
  <script src="https://cdn.jsdelivr.net/npm/zxcvbn@4.4.2/dist/zxcvbn.js"></script>
  <script src="https://unpkg.com/node-forge@1.0.0/dist/forge.min.js"></script>
  <script>
    const isLocalhost =
      location.hostname === "localhost" || location.hostname === "127.0.0.1";
    const API_BASE = isLocalhost
      ? "http://localhost:5000"
      : "https://dp-jha0.onrender.com";
    const sucReg = document.getElementById("sucReg");
    const sucLog = document.getElementById("sucLog");
    const output = document.getElementById("output");
    const passwInput = document.getElementById("passwInput");
    const passwordToggle = document.getElementById("passwordToggle");
    const passwordToggle2 = document.getElementById("passwordToggle2");
    const passwStr = document.getElementById("passwStr");
    const loginInput = document.getElementById("loginInput");
    const profileRole = document.getElementById("profileRole");
    const profileCard = document.getElementById("profileCard");
    const profileUsername = document.getElementById("profileUsername");
    const logoutBtn = document.getElementById("logoutBtn");
    const registerForm = document.getElementById("registerForm");
    const loginForm = document.getElementById("loginForm");
    const listHeader = document.getElementById("listHeader");
    let token = sessionStorage.getItem("token") || "";
    function showProfile(username, role) {
      profileUsername.textContent = username;
      profileRole.textContent = role;
      profileCard.style.display = "block";  }
```

```
    let currentUser = {};
    if (sessionStorage.getItem("token")) {
      const payload = JSON.parse(atob(token.split(".")[1]));
      currentUser = {
        id: payload.id,
        username: payload.username,
        role: payload.role,  };  }
    const DB_NAME = "CryptoKeysDB";
const DB_VERSION = 2;
const STORE_NAME = "keys";
function openDB() {
  return new Promise((resolve, reject) => {
    const request = indexedDB.open(DB_NAME, DB_VERSION);
    request.onupgradeneeded = function (event) {
      const db = event.target.result;
      if (!db.objectStoreNames.contains(STORE_NAME)) {
        db.createObjectStore(STORE_NAME);}};
    request.onsuccess = function (event) {
      resolve(event.target.result);};
    request.onerror = function (event) {
      reject(event.target.error);};});}
    function generateRSAKeyPair() {
      const keypair = forge.pki.rsa.generateKeyPair(2048);
      return {
        publicKey: forge.pki.publicKeyToPem(keypair.publicKey),
        privateKey: forge.pki.privateKeyToPem(keypair.privateKey),
      };}
    function hideProfile() {
      profileCard.style.display = "none";
      profileUsername.textContent = "";
      profileRole.textContent = "";
      token = "";
      sessionStorage.removeItem("token");
      sessionStorage.removeItem("role");
      alert("Ви вийшли з системи"); }
    passwInput.addEventListener("input", () => {
      const password = passwInput.value;
      const result = zxcvbn(password);
      const score = result.score;
      let feedback = "";
      if (score < 4) {
        feedback = "Занадто слабкий пароль";
```

```
      passwInput.classList.remove("strong");
      passwInput.classList.add("weak");
      passwStr.classList.remove("strongFeed");
      passwStr.classList.add("weakFeed");
    } else {
      feedback = "Пароль надійний";
      passwInput.classList.remove("weak");
      passwInput.classList.add("strong");
      passwStr.classList.remove("weakFeed");
      passwStr.classList.add("strongFeed"); }
    passwStr.innerHTML = `
  <p>Рівень надійності: ${score} / 4</p>
  <p>${feedback}</p>`;});
  async function encryptPrivateKey(privateKeyString, password) {
    const encoder = new TextEncoder();
    const salt = crypto.getRandomValues(new Uint8Array(16)); // сіль
    const iv = crypto.getRandomValues(new Uint8Array(12)); // ініціалізаційний вектор
    const key = await deriveKeyFromPassword(password, salt);
    const encrypted = await crypto.subtle.encrypt(
      { name: "AES-GCM", iv: iv },
      key,
      encoder.encode(privateKeyString) );
    return {
      encryptedData: new Uint8Array(encrypted),
      iv: iv,
      salt: salt,
    };  }
  async function savePrivateKeyToIndexedDB(encryptedData, iv, salt) {const db = await openDB();
const tx = db.transaction("keys", "readwrite");
const store = tx.objectStore("keys");
const encryptedPackage = {
  encryptedData: Array.from(encryptedData),
  iv: Array.from(iv),
  salt: Array.from(salt),  };
await new Promise((resolve, reject) => {
  const request = store.put(encryptedPackage, "privateKey");
  request.onsuccess = () => {
    console.log("Зашифрований приватний ключ збережено в IndexedDB");
    resolve();  };
  request.onerror = (event) => {
    console.error("Помилка збереження ключа в IndexedDB:", event.target.error);
    reject(event.target.error); };  });}
```

```
   async function deriveKeyFromPassword(password, salt) {
    const encoder = new TextEncoder();
    const keyMaterial = await crypto.subtle.importKey(
     "raw",
     encoder.encode(password),
     "PBKDF2",
     false,
     ["deriveKey"]);
    return await crypto.subtle.deriveKey(
     {name: "PBKDF2",
      salt: salt,
      iterations: 100000,
      hash: "SHA-256",  },
     keyMaterial,
     { name: "AES-GCM", length: 256 },
     false,
     ["encrypt", "decrypt"]   );  }
  async function decryptPrivateKey(encryptedData, iv, salt, password) {
const encData = new Uint8Array(encryptedData);
const vector = new Uint8Array(iv);
const saltBytes = new Uint8Array(salt);
    const key = await deriveKeyFromPassword(password, saltBytes);
    try {
    const decrypted = await crypto.subtle.decrypt(
     { name: "AES-GCM", iv: vector },
     key,
     encData);
    const decoder = new TextDecoder();
    return decoder.decode(decrypted);}
    catch(err){      } }
  async function loadEncryptedPrivateKeyFromIndexedDB() { const db = await openDB();
const tx = db.transaction("keys", "readonly");
const store = tx.objectStore("keys");
return new Promise((resolve, reject) => {
  const getRequest = store.get("privateKey");
  getRequest.onsuccess = function () {
    const result = getRequest.result;
    if (result) {
     resolve({
      encryptedData: new Uint8Array(result.encryptedData),
      iv: new Uint8Array(result.iv),
      salt: new Uint8Array(result.salt),  }); } else {
```

```
    resolve({});  } };
getRequest.onerror = function (event) {
  reject(event.target.error); };  });}
async function fetchUsersList() {
  const savedToken = sessionStorage.getItem("token");
  if (!savedToken) {
    console.log("Спочатку увійдіть у систему");
    return;}
  try {
    const res = await fetch(`${API_BASE}/users`, {
      method: "GET",
      headers: {
        Authorization: `Bearer ${savedToken}`,},});
    if (!res.ok) {
      const error = await res.json();
      console.log("Помилка:", error.message);
      return;}
    const users = await res.json();
    const list = document.getElementById("userList");
    list.innerHTML = "";
    listHeader.style.display = "block";
    users.forEach((user) => {
      const li = document.createElement("li");
      li.textContent = `${user.username} — роль: ${user.role}`;
      const deleteButton = document.createElement("button");
      deleteButton.style.display = "none";
      deleteButton.classList.add("delete-button");
      if (currentUser.role === "admin" && user.role !== "admin") {
        deleteButton.style.display = "inline-block";
        deleteButton.textContent = "Видалити";
        deleteButton.addEventListener("click", async () => {
          if (!confirm("Ви впевнені, що хочете видалити цього користувача?"))
            return;
          const userId = user._id;
          try {
            const response = await fetch(`/users/${userId}`, {
              method: "DELETE",
              headers: {
                Authorization: `Bearer ${token}`,},});
            if (response.ok) {
              console.log("Користувача", user.username, "видалено");
              alert("Користувача", user.username, "видалено");
```

```
            deleteButton.closest("li")?.remove(); // або видали з DOM інший елемент
          } else {
            const error = await response.text();
            console.error("Помилка видалення:", error);}} catch (err) {
          console.error("Помилка з'єднання:", err);}});
        li.appendChild(deleteButton);}
      list.appendChild(li);});  } catch (e) {
    console.log("Помилка з’єднання з сервером", e); }}
async function tryDecryptPrivateKey(password) {
 try {const { encryptedData, iv, salt } = await loadEncryptedPrivateKeyFromIndexedDB();
  if (!encryptedData || !iv || !salt) {
    throw new Error("Зашифрований ключ не знайдено.");}
loginForm.addEventListener("submit", async (e) => {
    e.preventDefault();
    loginInput.type = "password";
    const data = Object.fromEntries(new FormData(loginForm));
    try { const res = await fetch(`${API_BASE}/login`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(data), });
     const result = await res.json();
     if (res.ok) {
      sessionStorage.setItem("token", result.token);
      sessionStorage.setItem("role", result.role);
      token = result.token;
      const payload = JSON.parse(atob(result.token.split(".")[1]));
      currentUser = {
        id: payload.id,
        username: payload.username,
        role: payload.role,  };
      const { encryptedData, iv, salt } = await loadEncryptedPrivateKeyFromIndexedDB();
      if (!encryptedData || !iv || !salt) {
        console.warn("Приватний ключ не знайдено в IndexedDB.");} else {
        try {const privateKey = await tryDecryptPrivateKey(
          data.password);} catch (e) {
          console.error(
            "Не вдалося розшифрувати приватний ключ. Можливо, неправильний пароль.");
          return;}
 </script>
</body>
</html>
```

ДОДАТОК Б. Код серверної частини застосунку

```
const rateLimit = require("express-rate-limit");
const dotenv = require("dotenv");
dotenv.config();
const zxcvbn = require("zxcvbn");
const path = require("path");
const express = require("express");
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
const bodyParser = require("body-parser");
const mongoose = require("mongoose");
const { body, validationResult } = require("express-validator");
const cors = require("cors");
const forge = require("node-forge");
const app = express();
const PORT = process.env.PORT || 5000;
const SECRET_KEY = process.env.JWT_SECRET;
app.use(bodyParser.json());
app.use(cors());
mongoose
  .connect(process.env.MONGO_URI)
  .then(() => console.log("MongoDB підключено"))
  .catch((err) => console.error("Помилка підключення до MongoDB:", err));
const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  publicKey: { type: String, required: true },
  role: { type: String, default: "user" },});
const User = mongoose.model("User", userSchema);
const failedLoginByUsername = {};
const failedLoginByIP = {};
const MAX_FAILED_ATTEMPTS = 5;
const LOCK_TIME = 15 * 60 * 1000;
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 100,
  message: "Занадто багато запитів. Спробуйте пізніше.",});
const allowedOrigins = [
  "http://localhost:5000",
  "https://dp-jha0.onrender.com",];
function isBlocked(attemptInfo) {
```

```
  if (!attemptInfo) return false;
  if (attemptInfo.count < MAX_FAILED_ATTEMPTS) return false;
  const timePassed = Date.now() - attemptInfo.lastAttempt;
  if (timePassed > LOCK_TIME) {
    return false;}
  return true;}
function recordFailedAttempt(storage, key) {
  if (!storage[key]) {
    storage[key] = { count: 1, lastAttempt: Date.now() };
  } else {
    storage[key].count += 1;
    storage[key].lastAttempt = Date.now();}}
function resetFailedAttempts(storage, key) {
  if (storage[key]) {
    delete storage[key]; }}
function getRemainingAttempts(username) {
  const attemptInfo = failedLoginByUsername[username];
  if (!attemptInfo) return MAX_FAILED_ATTEMPTS;
  if (Date.now() - attemptInfo.lastAttempt > LOCK_TIME) {
    return MAX_FAILED_ATTEMPTS;  }
  return Math.max(0, MAX_FAILED_ATTEMPTS - attemptInfo.count);}
function checkPasswordStrength(password) {
  const result = zxcvbn(password);
  if (result.score < 4) {
    return {
      ok: false,
      message: "Пароль занадто слабкий. Спробуйте складніший.",
      feedback: result.feedback,};  }
  return { ok: true };}
function authorizeRoles(...allowedRoles) {
  return async (req, res, next) => {
    const authHeader = req.headers.authorization;
    if (!authHeader || !authHeader.startsWith("Bearer ")) {
      console.log("Немає токена або невірний формат заголовку");
      return res.status(401).json({ message: "Немає токена" }); }
    const token = authHeader.split(" ")[1];
    try {
      const decoded = jwt.verify(token, SECRET_KEY);
      const user = await User.findOne({ username: decoded.username });
      if (!user || !allowedRoles.includes(user.role)) {
        return res
          .status(403)
```

```
          .json({ message: "Доступ заборонено: недостатньо прав" }); }
      req.user = user; // прикріпимо користувача до запиту
      next();} catch (err) {
      console.error("Помилка авторизації:", err);
      res
        .status(403)
        .json({ message: "Недійсний токен або помилка перевірки" });}};}
function authenticateToken(req, res, next) {
  const token = req.headers.authorization?.split(" ")[1];
  if (!token) return res.sendStatus(401);
  jwt.verify(token, SECRET_KEY, (err, user) => {
    if (err) return res.sendStatus(403);
    req.user = user; // тут user має role
    next();  });}
app.use(
  cors({
    origin: (origin, callback) => {
      if (!origin || allowedOrigins.includes(origin)) {
        callback(null, true);
      } else {
        callback(new Error("Not allowed by CORS"));   }  },
    credentials: true,  }));
app.post(
  "/register",
  [ body("username")
      .isLength({ min: 3 })
      .withMessage("Ім’я користувача має містити щонайменше 3 символи"),
    body("password")
      .isLength({ min: 8 })
      .withMessage("Пароль має містити щонайменше 8 символів"),
    body("role")
      .optional()
      .isIn(["user", "admin"])
      .withMessage("Невідома роль"),
    body("publicKey").notEmpty().withMessage("Public key is required"),],
  async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });}
    const { username, password, publicKey } = req.body;
    const role = "user";
    const check = checkPasswordStrength(password);
```

```
    if (!check.ok) {
      return res
        .status(400)
        .json({ error: check.message, feedback: check.feedback });}
    try {const userExists = await User.findOne({ username });
      if (userExists) {
        return res.status(400).json({ message: "Користувач вже існує" });  }
      const hashedPassword = await bcrypt.hash(password, 10);
      const newUser = new User({
        username,
        password: hashedPassword,
        publicKey,
        role: "user",});
      await newUser.save();
      res.json({ message: "Реєстрація успішна" });} catch (err) {
      console.error("Помилка під час реєстрації:", err);
      res.status(500).json({ message: "Помилка сервера" });  }  });
app.post("/login", async (req, res) => {
  const { username, password } = req.body;
  const ip = req.ip;
  if (isBlocked(failedLoginByUsername[username])) {
    return res.status(429).json({
      message: `Користувач тимчасово заблокований. Спробуйте пізніше.`,  });}
  if (isBlocked(failedLoginByIP[ip])) {
    return res.status(429).json({
      message: `З вашої IP-адреси заблоковано надто багато спроб. Спробуйте пізніше.`,});}
  try {const user = await User.findOne({ username });
    if (!user) {
      recordFailedAttempt(failedLoginByUsername, username);
      recordFailedAttempt(failedLoginByIP, ip);
      const remaining = getRemainingAttempts(username);
      return res.status(400).json({
        message: "Користувача не знайдено",
        remainingAttempts: remaining,});  }
    const isPasswordValid = await bcrypt.compare(password, user.password);
    if (!isPasswordValid) {
      recordFailedAttempt(failedLoginByUsername, username);
      recordFailedAttempt(failedLoginByIP, ip);
      const remaining = getRemainingAttempts(username);
      return res
        .status(401)
        .json({ message: "Неправильний пароль", remainingAttempts: remaining });}
```

```
    resetFailedAttempts(failedLoginByUsername, username);
    resetFailedAttempts(failedLoginByIP, ip);
    const payload = { username: user.username, role: user.role };
    const token = jwt.sign(payload, SECRET_KEY, { expiresIn: "1h" });
    res.json({ message: "Успішний вхід", token, role: user.role });  } catch (err) {
    console.error("Помилка під час входу:", err);
    res.status(500).json({ message: "Помилка сервера" }); }});
app.get("/admin", authorizeRoles("admin"), (req, res) => {
  res.json({ message: `Ласкаво просимо, адміністраторе ${req.user.username}` });});
app.get("/users", authorizeRoles("admin", "user"), async (req, res) => {
  try {
    const users = await User.find({}, "username role").exec();
    res.json(users);  } catch (err) {
    console.error("Помилка отримання користувачів:", err);
    res.status(500).json({ message: "Помилка сервера" });}});
app.delete("/users/:id", authenticateToken, async (req, res) => {
  const currentUser = req.user;
  if (currentUser.role !== "admin") {
    return res.status(403).json({ message: "Доступ заборонено" });  }
  const id = req.params.id;
  if (req.user.id === id) {
    return res.status(400).json({ message: "Неможливо видалити самого себе" });}
  if (req.user.role !== "admin") {
    return res
      .status(403)
      .json({ message: "Доступ заборонено. Ви не адміністратор" });  }
  try {const result = await User.deleteOne({ _id: id });
    if (result.deletedCount === 0) {
      return res.status(404).send("Користувача не знайдено");}
    res.status(200).send("Користувач видалений");} catch (err) {
    console.error(err);
    res.status(500).send("Помилка сервера");}});
app.post("/logout", (req, res) => {
  res.json({ message: "Вихід успішний" });});
app.use(limiter);
app.use(express.static(path.join(__dirname, "public")));
app.get("/", (req, res) => {
  res.sendFile(path.join(__dirname, "public", "index.html"));});
app.listen(PORT, () => {
  console.log(`Сервер запущено на http://localhost:${PORT}`);});
```

ДОДАТОК В. Слайди мультимедійної презентації

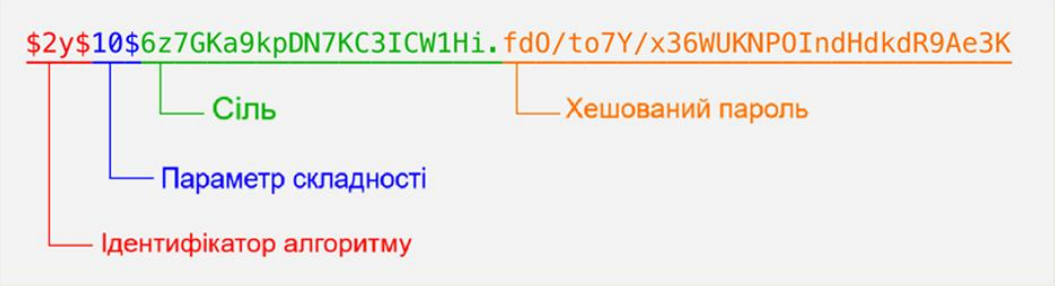
Розробка web-застосунку для аутентифікації користувачів з використанням методів криптографії

Ярошенко Аліна 4КБ-02

Криптографічні методи

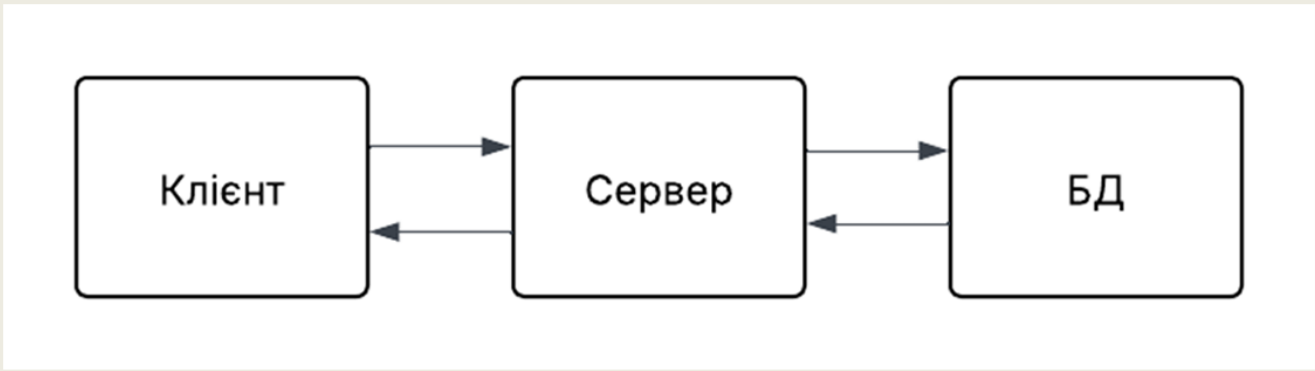
Асиметричне шифрування

Симетричне шифрування

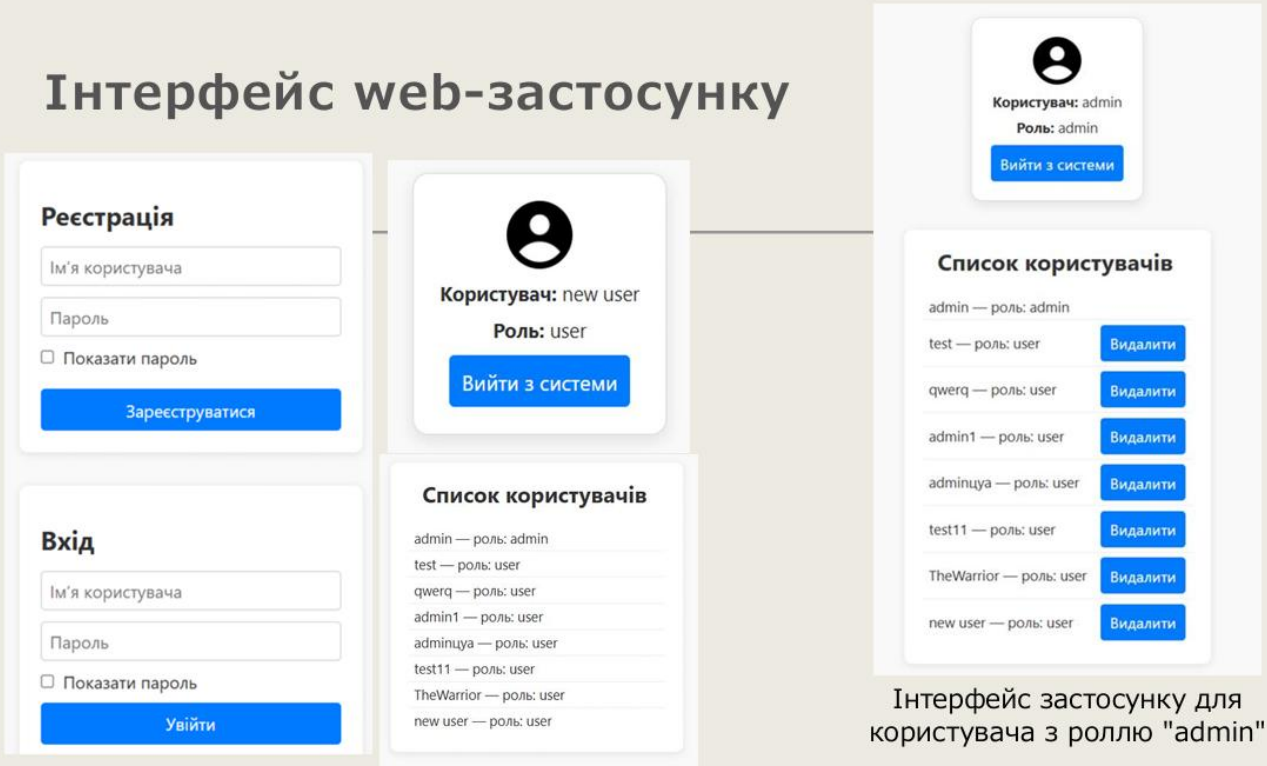


Bcrypt

Архітектура web-застосунку

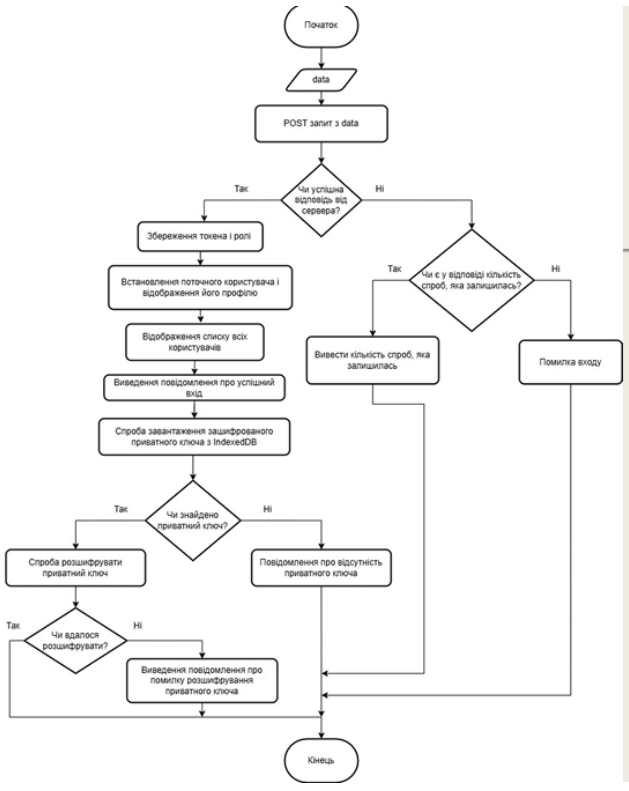


Інтерфейс web-застосунку



Інтерфейс застосунку для користувача з роллю "admin"

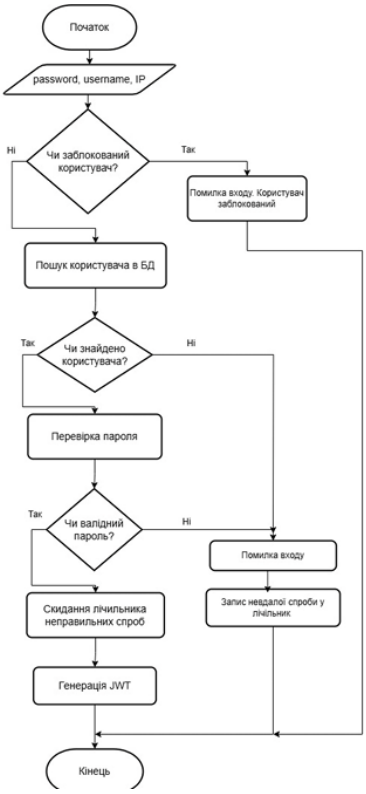
Блок-схема алгоритму реєстрації на клієнтській частині

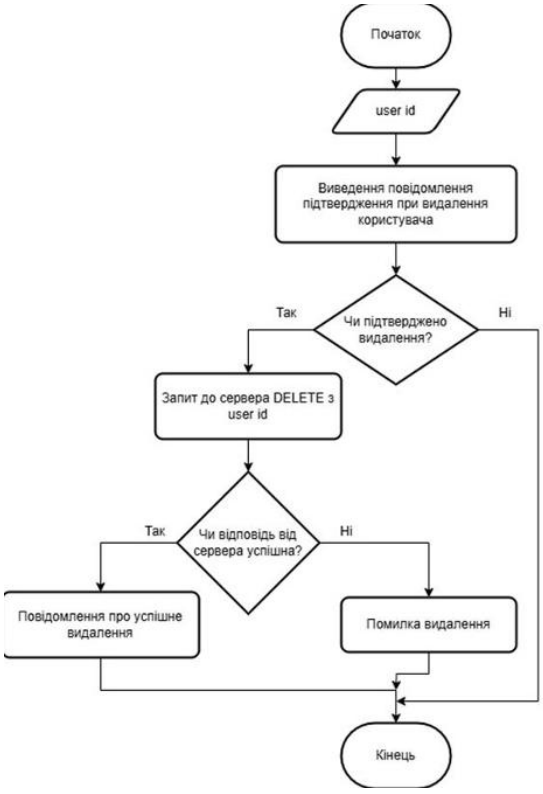


Блок-схема алгоритму входу на клієнтській частині

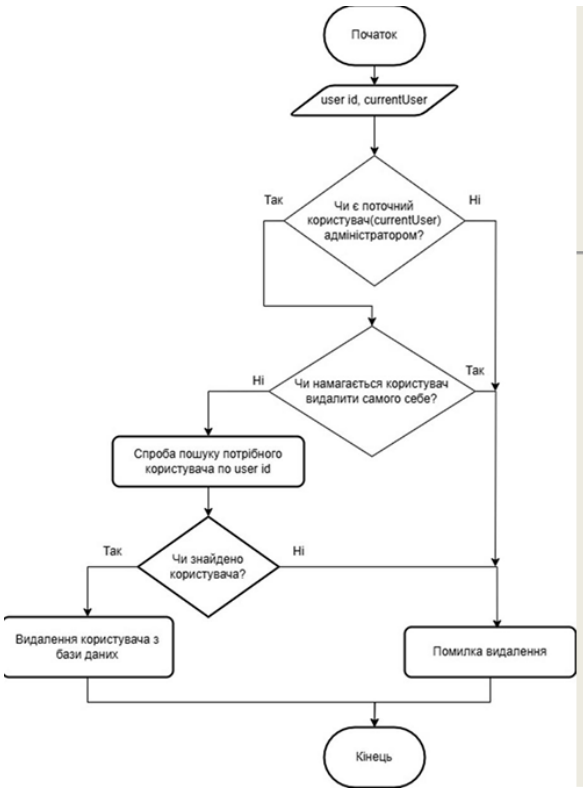
Блок-схема алгоритму реєстрації на серверній частині

Блок-схема алгоритму входу на серверній частині



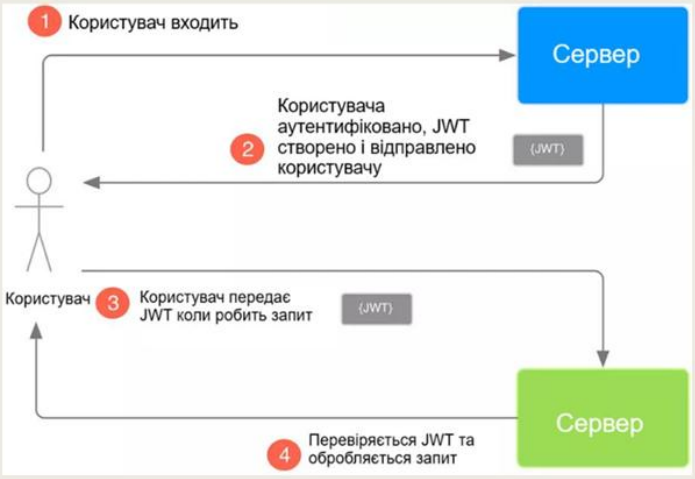


Блок-схема алгоритму видалення користувача на клієнтській частині



Блок-схема алгоритму видалення користувача на серверній частині

JWT токен



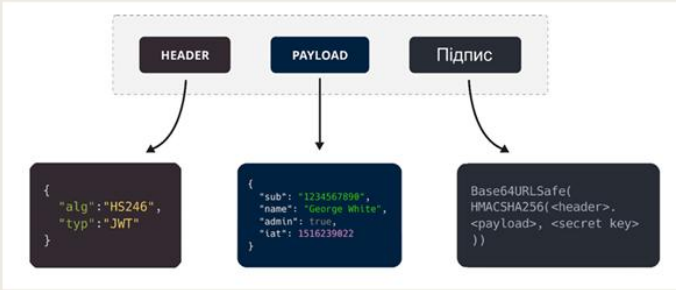


Схема атаки XSS (Cross-Site Scripting)

Схема атаки MITM (Man-In-The-Middle)

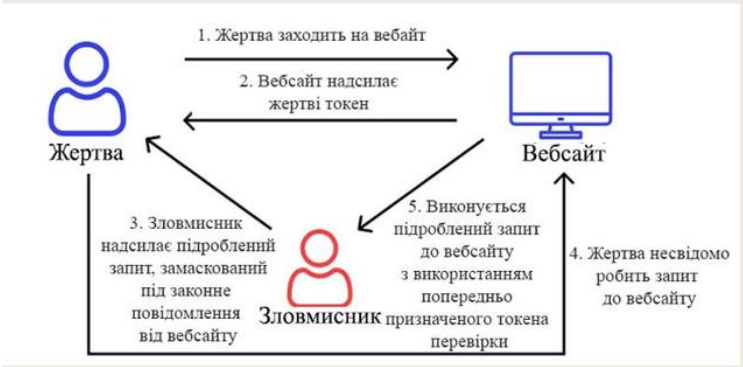


Схема атаки CSRF (Cross-Site Request Forgery)

Схема атаки розпиленням паролів (Password Spraying)

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Ярошенко Аліні Анатоліївні

(прізвище, ім`я та по батькові)

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Безпека комп'ютерних систем і мереж»*

Керівник дипломного проекту (роботи) *Гаджиєв Матін Магсудович*

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка web-застосунку для аутентифікації користувачів з використанням методів криптографії*

Обсяг розрахунково-пояснювальної записки *89* сторінок

Обсяг графічної (презентаційної) частини *12* аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту завданню

Представлений дипломний проєкт відповідає затвердженій темі та виконаний відповідно технічному завданню. Дипломний проект присвячений створенню web-застосунку для аутентифікації користувачів з використанням методів криптографії і складається з пояснювальної записки та мультимедійної презентації з відповідними схемами.

б) характеристика виконання кожного розділу дипломного проекту

Пояснювальна записка складається з основного розділу, економічного розділу, розділу охорони праці та додатків. Перелічені розділи поетапно охоплюють розробку, виконані докладно та обґрунтовано.

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту

Графічна частина складається з 12 слайдів мультимедійної презентації, виконаної у програмному продукті MS PowerPoint, які містять структурні, принципові та функціональні схеми, структурні моделі, блок-схеми алгоритмів, передбачені технічним завданням. Пояснювальна записка виконана акуратно та у відповідності до норм. Якість виконання пояснювальної записки відмінна, розробку виконано у повному обсязі.

г) перелік позитивних якостей дипломного проекту *Вичерпний захист від основних атак. Перевірка надійності пароля. Чітка структура: окремі форми реєстрації/входу, API-роути, моделі Mongoose та обробники логіки. Використання сучасного стеку Node.js + Express + MongoDB + JWT + bcrypt + indexedDB дають багатофункціональну та підйомну платформу. Захищена передача даних*

д) основні недоліки дипломного проекту *Генерація 2048-бітних RSA-ключів у браузері (node-forge) дуже повільна й непотрібна для простої аутентифікації. У моделі реєстрації сервер зберігає публічний ключ, але жодного шифрування на його основі не відбувається. Відсутність HTTPS-налаштувань у коді*

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові рецензента *к.т.н. Рудніченко Микола Дмитрович*

Місце роботи і посада рецензента *Національний університет «Одеська політехніка», доцент кафедри інформаційних технологій*

Підпис: _____

«___» _____ 2025 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проєкт здобувача (здобувачки) освіти

відділення комп'ютерних систем

Ярошенко Аліни Анатоліївни

(прізвище, ім`я та по батькові)

Спеціальність: *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма: *«Безпека комп'ютерних систем і мереж»*

Тема дипломного проєкту: *Розробка web-застосунку для аутентифікації користувачів з використанням методів криптографії*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЄКТУ

а) обсяг і якість виконання проєкту (графічного матеріалу і розрахунково-пояснювальної записки): обсяг і якість виконання проєкту повністю відповідають визначеним критеріям для кваліфікаційних робіт. Розрахунково-пояснювальна записка характеризується оптимальним обсягом, чіткою логічною структурою та коректним оформленням. Графічний матеріал поданий у зрозумілій та інформативній формі, що дозволяє ефективно відобразити ключові етапи процесу проєктування, впровадження та функціонування розробленої системи.

б) самостійність роботи над проєктом: у ході виконання дипломного проєкту студент продемонстрував високий рівень самостійності, особисто займався розробкою програмного забезпечення, приймав технічні рішення та здійснював тестування системи. Така робота свідчить про здатність результативно організовувати власну діяльність, ефективно працювати і проявляти ініціативу у вирішенні професійних завдань.

в) теоретична підготовка випускника (випускниці): випускник упевнено володіє спеціалізованою термінологією, демонструючи ґрунтовне розуміння основ криптографії, методів автентифікації, принципів розробки вебзастосунків та захисту інформації. У процесі виконання проєкту була проявлена здатність до самостійного аналізу джерел і ефективного впровадження теоретичних знань у практичну діяльність.

г) вміння розв'язувати виробничі та конструкторські питання: здатність вирішувати виробничі й конструкторські завдання проявилася у добре продуманих технічних рішеннях, належній організації серверної та клієнтської частин застосунку, а також у впровадженні дієвих механізмів шифрування, контролю доступу та захисту від атак. Проєкт демонструє практичну цінність і може слугувати базою для реалізації у реальних системах із високими вимогами до безпеки.

Оцінка розрахункової частини 5 (п'ять)

Оцінка графічної частини 5 (п'ять)

Загальна оцінка 5 (п'ять)

Прізвище, ім'я, по батькові керівника дипломного проєкту

д. т. н., проф. *Гаджиєв Матин Магсудович*

Місце роботи і посада керівника дипломного проєкту

ДДТУ, зав. каф. ІПЗ

Підпис

«18» 06 2025 р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
(ДИПЛОМНОГО ПРОЕКТУ)
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Ярошенко Аліна Анатоліївна
здобувач освіти гр. 4КБ-02, та

Гаджиєв Матин Магсудович,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до дипломного проекту фахового молодшого бакалавра на тему:

«Розробка web-застосунку для аутентифікації користувачів з використанням методів криптографії» (автор роботи –Ярошенко А.А., керівник роботи –Гаджиєв М.М.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2025 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Ярошенко А.А. /

Керівник _____ / Гаджиєв М.М. /

«12» червня 2025 р.

ДОВІДКА

циклової комісії КТ та ПІ
про допуск до захисту дипломного проєкту
здобувача (здобувачки) освіти IV курсу
відділення комп'ютерних систем групи 4КБ-02

Ярошенко Аліни Анатоліївни

на тему *Розробка web-застосунку для аутентифікації користувачів з використанням методів криптографії*

Висновок відповідальної особи за проведення нормоконтролю:

пояснювальна записка до дипломного проєкту виконана з некритичними порушеннями ДСТУ та оформлена відповідно до вимог Положення про дипломне проєктування

(підпис)	*18.06.2025* (дата)	*Петрашова В.І.* (П.І.Б.)

Висновок відповідальної особи за перевірку роботи на наявність академічного плагіату *згідно звіту про перевірку від 18.06.2025 р. значення коефіцієнту подібності в роботі становить 16,87%, коефіцієнт цитування – 2,62%.*

(підпис)	*18.06.2025* (дата)	*Краснокутська К.Г.* (П.І.Б.)

Попередня експертиза (малий захист) дипломного проєкту

здобувача (здобувачки) освіти *Ярошенко А.А.* (П.І.Б.)

проведена «*18*» *червня* 2025 р.

Висновки *Пояснювальна записка до дипломного проєкту виконана у повному обсязі. Випускна кваліфікаційна робота (дипломний проєкт) відповідає вимогам Положення про дипломне проєктування та рекомендована до захисту.*

Голова ЦК КТ та ПІ _____ (підпис) *Кривченко Ю.В.* (П.І.Б.)

StrikePlagiarism.com

Дата звіту 6/18/2025
Дата редагування 6/18/2025

Документ прийнятий

Звіт подібності

метадані

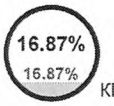
Назва організації
Odesa Technical Professional College of Odesa National University of Technology
Заголовок
Розробка web-застосунку для аутентифікації користувачів з використанням методів криптографії

Автор Науковий керівник / Експерт
Ярошенко Аліна АнатоліївнаГаджиєв Матин Магсудович

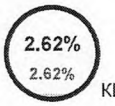
підрозділ
Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету"

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1



КЦ

25	17824	145804
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		0
Інтервали		0
Мікропробіли		0
Білі знаки		0
Парафрази (SmartMarks)	a	129

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://cybercalm.org/novyny/shho-take-ataka-man-in-the-middle-ta-yak-sebe-zahystyty/	66 0.37 %
2	https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download	63 0.35 %
3	https://card-file.ontu.edu.ua/bitstreams/8999d5af-6274-44f4-ae78-d23e08048d38/download	61 0.34 %
4	СТВОРЕННЯ СИСТЕМИ ШИФРУВАНЬ ПОВІДОМЛЕНЬ [redacted] 3/8/2025 Lesya Ukrainka Volyn National University (Кафедра комп'ютерних наук та кібербезпеки)	55 0.31 %

5	https://elartu.tntu.edu.ua/bitstream/lib/46054/1/Bachelor_Thesis__Siushko_2024.pdf	54 0.30 %
6	СТВОРЕННЯ СИСТЕМИ ШИФРУВАНЬ ПОВІДОМЛЕНЬ [redacted] 3/8/2025 Lesya Ukrainka Volyn National University (Кафедра комп'ютерних наук та кібербезпеки)	51 0.29 %
7	РОЗРОБКА ВЕБ ДОДАТКУ З УПРАВЛИННЯ ВЛАСНИМИ ФІНАНСАМИ ТА СТВОРЕННЯ ЗАМІТОК [redacted] 6/10/2021 National University Chernihiv Politechnika (NUCP) 2 (Дипломні роботи)	46 0.26 %
8	https://card-file.ontu.edu.ua/bitstreams/6cf43324-8f08-4031-ba42-f80b18efbbc8/download	40 0.22 %
9	https://card-file.ontu.edu.ua/bitstreams/55e2b8f2-7d3c-4235-99fc-2be51199b96d/download	39 0.22 %
10	https://card-file.ontu.edu.ua/bitstreams/bbaf3f38-16a8-4070-bead-5562769b7c71/download	37 0.21 %

з домашньої бази даних (0.03 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності [redacted] 6/12/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	6 (1) 0.03 %

з програми обміну базами даних (2.43 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	СТВОРЕННЯ СИСТЕМИ ШИФРУВАНЬ ПОВІДОМЛЕНЬ [redacted] 3/8/2025 **Lesya Ukrainka Volyn National University (Кафедра комп'ютерних наук та кібербезпеки)**	**211 (8) 1.18 %**
2	РОЗРОБКА ВЕБ ДОДАТКУ З УПРАВЛИННЯ ВЛАСНИМИ ФІНАНСАМИ ТА СТВОРЕННЯ ЗАМІТОК [redacted] 6/10/2021 **National University Chernihiv Politechnika (NUCP) 2 (Дипломні роботи)**	**70 (3) 0.39 %**
3	КПІ_2025_123_Сігаєв [redacted] 6/4/2025 Ukrainian national aviation university (Криворізький Фаховий коледж)	30 (5) 0.17 %
4	Məhbusların əl əməyinin online bazara çıxarılması [redacted] 7/10/2019 Azerbaijan Technical University (ATU) (İnformasiya texnologiyaları və proqramlaşdırma)	28 (2) 0.16 %
5	«Создание интернет магазина радиоуправляемой техники» [redacted] 4/25/2024 Kirgiz Economic University na M. Ryskulbekov (Кафедра "Цифровая экономика и программирование")	19 (2) 0.11 %
6	МЕТОДИ ЗАХИСТУ КОНФІДЕНЦІЙНИХ ДАНИХ ЗА ДОПОМОГОЮ [redacted] 4/29/2025 Lesya Ukrainka Volyn National University (Кафедра комп'ютерних наук та кібербезпеки)	11 (1) 0.06 %
7	Гамов_ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ ЗАХИСТУ ІНФОРМАЦІЇ У СУЧАСНИХ СИСТЕМАХ ПЕРЕДАЧІ ДАНИХ.docx [redacted] 6/15/2023 Kyiv International University (KIU) (факультет інформаційних технологій)	10 (1) 0.06 %

8	Методи захисту інформації при використанні комп'ютерних мереж. [redacted] 12/19/2023 National University Chernihiv Politechnika (NUCP) 2 (Інформаційний центр запобігання та виявлення плагіату)	9 (1) 0.05 %
9	Шклярський Р. А. (КБУІ-21 2024) [redacted] 12/2/2024 National University "Lviv Politechnika" (NULP2)	8 (1) 0.04 %
10	Дослідження та удосконалення стандартних методів захисту web-додатків [redacted] 12/16/2021 Ternopil Ivan Puľuj National Technical University (кафедра кібербезпеки)	8 (1) 0.04 %
11	2021_М_БІТ_Лібін_С_Є [redacted] 5/31/2024 Kharkiv National University of Radio Electronics (Kharkiv National University of Radio Electronics)	7 (1) 0.04 %
12	Аналіз методів та засобів захисту веб-сайту від несанкціонованого доступу та програмна реалізація захищеного веб-сайту [redacted] 12/10/2022 Kharkiv National University of Economics named after S.Kuznets (KNUE) (KNUE)	6 (1) 0.03 %
13	Модернізація систем захисту інформаційно-телекомунікаційної мережі підприємства стратегічного призначення [redacted] 5/18/2021 State University of Telecommunications (HHI3I)	6 (1) 0.03 %
14	Дослідження шляхів та розроблення рекомендацій щодо застосування криптографічних алгоритмів з відкритим та закритим ключем в інформаційній системі організації [redacted] 5/25/2023 State University of Telecommunications (HHI3I)	5 (1) 0.03 %
15	Implementarea, folosind un API cunoscut, a examenului de admitere dintr-o instituție de învățământ superior [redacted] 6/8/2024 Universitatea Titu Maiorescu (Facultatea de Informatica)	5 (1) 0.03 %

з Інтернету (14.41 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/server/api/core/bitstreams/44c16132-5f53-48e2-b6c0-61e9a2f0fd75/content	602 (50) 3.38 %
2	**https://card-file.ontu.edu.ua/bitstreams/538ada8a-2c79-4b1e-b7d2-b0c97f68bc1c/download**	**195 (16) 1.09 %**
3	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/8da72e29-656f-4ee4-9b22-716dedf53ff5/content**	**156 (11) 0.88 %**
4	**https://cybercalm.org/novyny/shho-take-ataka-man-in-the-middle-ta-yak-sebe-zahystyty/**	**136 (6) 0.76 %**
5	**https://elartu.tntu.edu.ua/bitstream/lib/46054/1/Bachelor_Thesis__Siushko_2024.pdf**	**105 (4) 0.59 %**
6	**https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download**	**95 (3) 0.53 %**
7	https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download	88 (9) 0.49 %
8	**https://dev.to/siddhant_teotia/mastering-authentication-in-nodejs-and-express-a-comprehensive-guide-5eoo**	**85 (7) 0.48 %**
9	**https://card-file.ontu.edu.ua/bitstreams/55e2b8f2-7d3c-4235-99fc-2be51199b96d/download**	**84 (6) 0.47 %**
10	http://chtyvo.org.ua/authors/Dzhonson_Pol/Istoriia_ievreiv.pdf	81 (13) 0.45 %
11	**https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download**	**63 (5) 0.35 %**

12	**https://card-file.ontu.edu.ua/bitstreams/8999d5af-6274-44f4-ae78-d23e08048d38/download**	**61 (1) 0.34 %**
13	**https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download**	**61 (3) 0.34 %**
14	**https://card-file.ontu.edu.ua/bitstreams/bbaf3f38-16a8-4070-bead-5562769b7c71/download**	**60 (3) 0.34 %**
15	https://card-file.ontu.edu.ua/bitstreams/aed610a6-43ef-47e0-9066-e85c89456f3e/download	51 (4) 0.29 %
16	**https://card-file.ontu.edu.ua/bitstreams/6cf43324-8f08-4031-ba42-f80b18efbbc8/download**	**40 (1) 0.22 %**
17	https://github.com/axios/axios/issues/4907	36 (4) 0.20 %
18	https://elartu.tntu.edu.ua/bitstream/lib/41775/1/2023_KRB_SN-41_Ternovyi_OV.pdf	36 (6) 0.20 %
19	https://dev.to/kosa12/creating-and-securing-jwt-authentication-in-a-web-app-5cni	35 (3) 0.20 %
20	https://ijisrt.com/design-and-implementation-of-a-multialgorithm-encryption-and-decryption-framework	35 (3) 0.20 %
21	**https://ami.lnu.edu.ua/wp-content/uploads/2022/06/Cryptology10.pdf**	**34 (1) 0.19 %**
22	https://ela.kpi.ua/bitstreams/de3beaa5-4636-40dd-8a87-8a8f2afbb7a0/download	34 (4) 0.19 %
23	https://oda.zht.gov.ua/wp-content/uploads/2021/05/Zvit-za-I-kvartal-2021-roku.pdf	31 (5) 0.17 %
24	**https://er.nau.edu.ua/bitstream/NAU/45189/1/%D0%9D%D0%9D%D0%86%D0%9E%D0%A2_2020_122_%D0%9E%D1%85%D1%80%D1%96%D0%BC%D0%B5%D0%BD%D0%BA%D0%BE_%D0%A1%D0%92.pdf**	**28 (1) 0.16 %**
25	https://docs.pingcode.com/baike/2267651	28 (3) 0.16 %
26	https://ichi.pro/th/srang-fil-xap-hold-dawn-hold-fang-kchan-dwy-kar-saedng-tawxyang-rupphaph-114743164138728	25 (3) 0.14 %
27	https://blog.lebara.co.uk/uk/%D1%8F%D0%BA-%D0%B1%D1%80%D0%B8%D1%82%D0%B0%D0%BD%D1%81%D1%8C%D0%BA%D1%96-%D0%BC%D0%BE%D0%B1%D1%96%D0%BB%D1%8C%D0%BD%D1%96-%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D1%96-%D0%B7%D0%B0%D1%85%D0%B8%D1%89%D0%B0/	25 (4) 0.14 %
28	https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download	25 (3) 0.14 %
29	http://www.datadisk.co.uk/html_docs/mern_redux/mern_2.html	22 (3) 0.12 %
30	https://ev.nmu.org.ua/docs/2019/3/EV20193_146-159.pdf	21 (3) 0.12 %
31	https://card-file.ontu.edu.ua/bitstreams/12d5c0ab-e979-48f2-a8ec-d5fc31f71fd5/download	18 (1) 0.10 %
32	https://journals.dut.edu.ua/index.php/dataprotect/article/download/2612/2511/	18 (2) 0.10 %
33	https://essuir.sumdu.edu.ua/bitstream/123456789/97630/1/Dryzhov_mag_rob.pdf	17 (1) 0.10 %
34	https://card-file.ontu.edu.ua/bitstreams/c1f3e592-1123-419d-b14a-4c28662f0f1e/download	17 (3) 0.10 %
35	https://stackoverflow.com/questions/70045203/getting-unauthorized-401-error-in-mern-application	14 (2) 0.08 %
36	https://card-file.ontu.edu.ua/bitstreams/361286d7-8a03-4221-ad05-db5133ab5f79/download	14 (1) 0.08 %
37	https://ela.kpi.ua/bitstreams/8fa63d66-50c3-4b15-9c39-22ed9f54e9af/download	13 (1) 0.07 %
38	https://www.ij.kubg.edu.ua/images/phocagallery/Podii2020/iba_docx/Informatsiini_texnologii_v_dokument oznavstvi.pdf	13 (2) 0.07 %
39	https://er.nau.edu.ua/bitstreams/f6b88457-1f80-4220-afc3-e4a8d71926df/download	12 (1) 0.07 %
40	https://cyberleninka.ru/article/n/elektronni-resursi-yak-skladova-bibliotechno-informatsiynogo-potentsialu-dnipra	10 (1) 0.06 %

41	https://card-file.ontu.edu.ua/bitstreams/62baa43e-b968-4993-bb54-8cf8761a89b2/download	10 (1) 0.06 %
42	https://github.com/graphql-compose/graphql-compose-mongoose/issues/158	7 (1) 0.04 %
43	https://zvitmarket.com/news/sho-take-i-yak-pracyuyut-tokeni-autentifikaciyi	6 (1) 0.03 %
44	http://www.ijeit.com/Vol%204/Issue%2012/IJEIT1412201506_03.pdf	6 (1) 0.03 %
45	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	5 (1) 0.03 %
46	https://www.codingdeft.com/posts/react-authentication-mern-node-passport-express-mongo/	5 (1) 0.03 %
47	https://card-file.ontu.edu.ua/bitstreams/bcb0d6f9-f464-4578-bda6-b5b2ce2349bb/download	5 (1) 0.03 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»
Освітньо-професійна програма: **«Безпека комп'ютерних систем і мереж»** **Група:** 4КБ- 02

Дипломний **проект здобувача освіти денної форми навчання КБ.** 02.25.000.ДП

ЯРОШЕНКО
АЛІНИ АНАТОЛІЇВНИ

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»
Освітньо-професійна програма: «Безпека комп'ютерних систем і мереж»
Група: 4 КБ-02

ПОЯСНЮВАЛЬНА ЗАПИСКА
до дипломного проекту на тему: _____
_____ Проектний матеріал складається з пояснювальної записки на _____ сторінках та графічного (презентаційного) матеріалу на _____ аркушах (слайдах) Дипломник _____(Ярошенко А.А.)
_____ Керівник _____ (Гаджиєв М.М.) Консультанти: з економічного розділу_____(Копайгородська Т.Г.) з розділу охорони праці та техніки безпеки_____ (Чорновол Н.І.) з нормоконтролю _____(Петрашова В.І.) старший консультант _____ (Кривченко Ю.В.)

До захисту допущений Голова циклової комісії _____ (Кривченко Ю.В.) Завідувач відділення _____(Краснокутська К.Г.)

_____ **Захист «____»_____ 2025 р. Протокол ЕК No_____ Оцінка** ЕК _____

Секретар ЕК _____
Розробка web-застосунку для аутентифікації користувачів
з використанням методів криптографії

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»