Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
завідувачка кафедри кібербезпеки
та захисту інформації
_____Наталія ЛУКОВА-ЧУЙКО
«14» червня 2022р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

дипломної роботи
бакалавра
(назва освітнього ступеня)

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітня програма Кібербезпека
(назва освітньої програми)

на тему: «Механізм безпечного зберігання паролів»

Виконавець: студентка IV курсу, групи КБ-41

_____ **Юлія СТЕПАНЕНКО**
(підпис) (ім'я прізвище)

	Прізвище, ініціали	**Підпис**
Керівник	Лариса МИРУТЕНКО	

Нормоконтроль	Олександр ТОРОШАНКО	

Київ 2022

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
завідувачка кафедри кібербезпеки
та захисту інформації
_____Наталія ЛУКОВА-ЧУЙКО
«01» листопада 2021 р.

ЗАВДАННЯ
на виконання дипломної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої програми)

Студентці **КБ-41** **Юлії СТЕПАНЕНКО**
(група) (ім'я, прізвище)

Тема дипломної роботи Механізм безпечного зберігання паролів

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема дипломної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №5 від 29.10.2021 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Алгоритми гешування, процес ідентифікації, автентифікації та авторизації, алгоритми та засоби розробки мобільних додатків

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Підхід до забезпечення безпеки даних користувачів, баз даних та серверів, рекомендації із забезпечення безпеки паролів, концепції зберігання паролів, розповсюджені геш-функції, відмінності гешування від шифрування, формування основних вимог до механізму зберігання паролів та його реалізація.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність розробка процесу авторизації користувачів, який у перспективі можна імплементувати в повноцінний мобільний додаток.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 01 листопада 2021 року

Завдання видала _____ (підпис) Лариса МИРУТЕНКО (ім'я прізвище)

Завдання прийняла до виконання _____ (підпис) Юлія СТЕПАНЕНКО (ім'я прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	01.11.2021 – 28.01.2022	*виконано*
2	Аналіз літератури	29.01.2022 – 13.02.2022	*виконано*
3	Аналіз підходу до забезпечення безпеки даних користувачів	14.02.2022 – 26.02.2022	*виконано*
4	Аналіз концепцій зберігання паролів	27.02.2022 – 20.03.2022	*виконано*
5	Аналіз геш-функцій, їх особливостей	21.03.2022 – 29.04.2022	*виконано*
6	З'ясування переваги гешування над шифруванням	30.04.2022 – 04.05.2022	*виконано*
7	Формування вимог до механізму	05.05.2022 – 18.05.2022	*виконано*
8	Реалізація механізму зберігання паролів	19.05.2022 – 04.06.2022	*виконано*
9	Оформлення пояснювальної записки	05.06.2022 – 06.06.2022	*виконано*
10	Підготовка до захисту	07.06.2022 – 13.06.2022	*виконано*

Завдання видала _____ (підпис) Лариса МИРУТЕНКО (ім'я прізвище)

Завдання прийняла до виконання _____ (підпис) Юлія СТЕПАНЕНКО (ім'я прізвище)

Термін подання дипломної роботи до ЕК 06 червня 2022 року

РЕФЕРАТ

Пояснювальна записка дипломної роботи складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел та додатків. Основний текст займає 70 сторінок, включає в себе зміст, вступ, три розділи дипломної роботи, висновки, список джерел та 1 додаток із загальною кількістю сторінок 4. У пояснювальній записці дипломної роботи міститься 26 рисунків і 4 таблиці.

Метою роботи є побудова механізму безпечного зберігання паролів.

Для досягнення зазначеної мети поставлено наступні завдання:

- проаналізувати підхід до забезпечення безпеки даних користувачів;
- проаналізувати концепції зберігання паролів та критерії створення надійних паролів;
- проаналізувати розповсюджені геш-функції, визначити їх особливості;
- сформувати основні вимоги до механізму безпечного зберігання паролів;
- побудувати механізм безпечного зберігання паролів.

Об'єктом дослідження є процес захисту персональних даних користувачів в інформаційно-комунікаційних системах.

Предметом дослідження є система зберігання паролів за допомогою криптографічної геш-функції.

Практичною цінністю отриманих результатів є розробка процесу авторизації користувачів, який у перспективі можна імплементувати в повноцінний мобільний додаток.

Ключові слова: пароль, алгоритм гешування, геш-функція, сіль, ітерації, база даних, облікові дані, додаток.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

AES – Advanced Encryption Standard
API – Application Programming Interface
DES – Data Encryption Standard
DoS/DDoS – (Distributed) Denial-of-service attack
FISMA – Federal Information Security Management Act
GDPR – General Data Protection Regulation
GLBA – The Gramm-Leach-Bliley Act
GPU – Graphics Processing Unit
HIPAA – Health Insurance Portability and Accountability Act
HMAC – hash-based message authentication code
HSM – Hardware Security Module
MD5 – Message Digest 5
MDC – Modification Detection Code
MHF – memory hard functions
MIC – Message Integrity Check
NIST – National Institute of Standards and Technology
PBKDF2 – Password-Based Key Derivation Function
PCI DSS – Payment Card Industry Data Security Standard
SHA – Secure Hash Algorithm
SOX – Sarbanes-Oxley Act
ОС – операційна система

ЗМІСТ

ВСТУП

Сучасність важко уявити без інформаційних технологій. Вони впевнено заполонили світ, стали частиною нашого життя. Якщо раніше різного роду девайси можна було побачити лише в окремих людей, то зараз це необхідна умова для повноцінного існування абсолютно кожного. Телевізори, комп'ютери, смартфони та Інтернет стали нашими повсякденними супутниками. Люди стали настільки залежними від технологій різного роду, що без них ми майже нічого не можемо зробити.

Інформаційні технології надзвичайно важливі, бо вони використовуються у всіх сферах життя. Гаджети використовуються у багатьох буденних справах – від будильника зранку і спілкування у соціальних мережах до корпоративної пошти на роботі й послуг онлайн-банкінгу.

Отже, наше суспільство уже давно перетворилося з індустріального на інформаційне, де інформація та інформаційні технології стали одними із найважливіших цінностей людства. Починаючи з кінця XX – початку XXI століття вони посідають дедалі значніше місце в житті кожної людини як члена соціуму.

Кількість електронних пристроїв щороку збільшується на тисячі й сотні мільйонів, а задачі, які вони допомагають вирішувати, стають більш різноманітними та складними. Якщо подумати, девайси знають про нас більше за наших родичів та друзів, адже саме їм ми довіряємо наші персональні дані. Паролі від облікових записів у соціальних мережах, інтернет-банкінгу, наші нотатки, розклад дня, медична картка – усе зберігається в електронних пристроях.

Усі ми у певному сенсі, самі того не помічаючи, стали заручниками діджиталізації. Наприклад, мобільний телефон із зрозумілих причин став нашою десницею або портативним помічником у всіх справах – від телефонних дзвінків та меседжингу до керування розумними пристроями і, забувши мобільний телефон вдома, неможливо викликати таксі чи подивитися геолокацію в онлайн картах.

Ми хвилюємося, коли втрачаємо доступ до власних облікових записів або коли дізнаємося, що в компанії, клієнтами якої ми є, стався витік даних

користувачів. Наприклад, у 2014 році один з найбільших інтернет-магазинів світу eBay став жертвою потужної кібератаки. Тоді в результаті постраждало 145 мільйонів активних користувачів вебсайту. До зловмисників потрапили персональні дані клієнтів, а саме імена, адреси місць проживання та поштові скриньки, а також зашифровані паролі, які потім могли використовуватись у спробах обману потенційних жертв [1].

Наразі саме паролі є найпоширенішим способом авторизації, хоча вже придумали більш надійні та сучасні методи, наприклад, авторизація за допомогою біометрії. Проте все ж таки у більшості ресурсів для підтвердження особистості використовують саме паролі. Потрапляння паролів від облікових записів до рук зловмисників може спричинити витік персональних даних користувачів. Саме тому паролі мають зберігатися особливим чином, що обумовлює актуальність обраної теми.

Метою дипломної роботи є побудова механізму безпечного зберігання паролів.

Для досягнення мети поставлені наступні завдання:

- проаналізувати підхід до забезпечення безпеки даних користувачів;
- проаналізувати концепції зберігання паролів та критерії створення надійних паролів;
- проаналізувати розповсюджені геш-функції, визначити їх особливості;
- сформувати основні вимоги до механізму безпечного зберігання паролів;
- побудувати механізм безпечного зберігання паролів.

Об’єктом дослідження є процес захисту персональних даних користувачів в інформаційно-комунікаційних системах.

Предметом дослідження є система зберігання паролів за допомогою криптографічної геш-функції.

Методи дослідження:

- аналіз відкритих інформаційних джерел;
- порівняння концепцій зберігання паролів;
- моделювання механізму зберігання паролів.

Новизна дипломної роботи полягає в розробці повноцінного процесу ідентифікації, автентифікації та авторизації користувача в інформаційно-комунікаційній системі зі збереженням його парольних даних відповідно до сучасного механізму забезпечення їх безпеки.

Практичне значення отриманих результатів полягає є розробці процесу авторизації користувачів, який у перспективі можна імплементувати в повноцінний мобільний додаток.

Основні результати дипломної роботи були представлені на XI Міжуніверситетській конференції студентів, аспірантів та молодих вчених «Engineer of XXI Century» (Бельсько-Бяла, 2021).

РОЗДІЛ 1
АНАЛІЗ ПІДХОДІВ ДО ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ДАНИХ КОРИСТУВАЧІВ

1.1 Безпека баз даних

З метою запобігання витоку даних й збереження конфіденційності інформації необхідно забезпечити передусім надійний захист системи, де ці дані зберігаються, оскільки наслідки витоку можуть бути непередбачувані.

Працюючи над забезпеченням безпеки бази даних, необхідно забезпечити захист кожного рівня – від фізичного середовища, де знаходиться сервер бази даних, до аудиту системи. База даних насамперед відноситься до інструментів, покликаних забезпечити цілісність, конфіденційність та доступність даних, які зберігаються. Проте при компрометації бази даних перш за все порушується саме конфіденційність [2].

Безпека бази даних повинна стосуватися також наступних понять:

- даних, що зберігаються в базі даних;
- системи управління базою даних;
- програми, пов’язані з базою даних;
- мережа, яка використовується для доступу до бази даних;
- сервер бази даних.

Проблема забезпечення і підтримки безпеки бази даних досить складна та багатогранна. Речник Департаменту труда та пенсій Великобританії каже про невелику кількість зафіксованих порушень (несанкціонованих доступів чиновників до баз даних), виявлених досить швидко, хоча й за допомогою лише вибіркових перевірок системи [3]. Можна зробити припущення, що порушень даного роду було б виявлено набагато більше при умові, що система була досліджена повністю, а не вибірково. Особливо це має велике значення, якщо справа стосується державних сервісів та додатків. Професор Росс Андерсон з Кембриджського університету та

NO2ID стверджують, що неможливо попередити подібні зловживання, маючи справу з великою базою даних. Правило Андерсона звучить наступним чином: неможливо побудувати велику функціональну та захищену базу даних, яка буде зручною у використанні, тому що чим зручнішу систему ви намагаєтесь спроектувати, тим вразливішою вона буде, і навпаки – захищеною системою складніше користуватись [3].

Компрометація бази даних ставить під загрозу перш за все конфіденційність інформації організації та підприємства, а саме інтелектуальної власності (комерційні таємниці, винаходи), що може підірвати конкуретоспроможність на ринку. Це однозначно вплине на репутацію бренду – клієнти та партнери можуть не захотіти співпрацювати, якщо не відчувають, що можуть довіряти свої дані. Під загрозу ставиться безперервність бізнесу, оскільки на відновлення може знадобитися час, людські та матеріальні ресурси [2]. Також може настати відповідальність за витік даних користувачів. Деякі закони та стандарти у сфері інформаційної безпеки, які вимагають контроль доступу, відслідковування розкриття та зміну чутливої інформації [4]:

- Федеральний акт про управління інформаційною безпекою (FISMA). FISMA вимагає від федеральних агенцій у Сполучених Штатах розробки та реалізації плану інформаційної безпеки в масштабі всієї організації [5].
- Загальний регламент із захисту даних (GDPR). GDPR встановлює правила захисту даних для всіх компанії, якими володіють іноземці, які обробляють дані мешканців Європейського Союзу [6].
- Акт про мобільність та підзвітність медичного страхування (HIPAA). HIPAA визначає вимоги до організацій охорони здоров'я до підтримки безпеки та конфіденційності даних пацієнтів [7].
- Закон Сарбейнса-Окслі (SOX). SOX визначає правила для фінансової звітності компаній [8].
- Закон Грамма-Ліча-Блайлі (GLBA). GLBA встановлює бачення щодо забезпечення захисту фінансової інформації споживачів [9].

- Стандарт безпеки індустрії платіжних карток (PCI DSS). PCI DSS визначає структуру для безпечної обробки інформації про кредитну картку.

Під загрозою безпеки розуміють потенційно можливі події, впливи, процеси або явища, які прямо чи побічно можуть нанести шкоду (збиток) інтересам об’єктів інформаційних відносин [10]. Серед загроз безпеці бази даних виділяють наступні [2]:

- людський фактор;
- інсайдер;
- використання дірок у безпеці програмного забезпечення бази даних;
- SQL-ін’єкції [11];
- атака типу переповнення буферу;
- DoS/DDoS атаки;
- шкідливе програмне забезпечення;
- атаки на резервні копії.

Безпека бази даних повинна виходити за межі лише-но однієї бази даних, оскільки загроза може виникнути в будь-якій частині мережевої інфраструктури, через яку база даних доступна [2].

Для захисту бази даних необхідно комплексно підійти до вирішення цієї задачі та забезпечити:

- фізичну безпеку: сервер бази даних має бути розміщений у захищеному приміщенні [4];
- контроль доступу: доступ до бази даних має бути в обмеженого кола користувачів з мінімальними правами;
- безпеку облікового запису користувача: необхідно спостерігати за тим, як і коли використовуються дані;
- шифрування: усі дані, що зберігаються у базі даних, мають шифруватись, будучи у стані спокою та передачі;
- безпеку програмного забезпечення, що взаємодіє з базою даних: завжди необхідно використовувати останні версії програмного забезпечення;

- тестування безпеки всіх додатків, що взаємодіють з базою даних [2];
- безпеку резервних копій: резервні копії мають бути захищені так само, як і сама база даних;
- реєстрацію усіх операцій та подій, що виконуються над даними та регулярний аудит безпеки [4].

1.2 Безпека серверного обладнання

Спеціалісти з інформаційної безпеки, маючи на меті забезпечити надійний захист даних користувачів системи, обов'язково повинні пам'ятати про безпеку серверного обладнання, оскільки з кожним днем кіберзлочинці розробляють все більш витончені атаки. До того ж, захищена база даних не буде означати абсолютно нічого, якщо не буде забезпечений захист сервера.

Сервер надає широкий спектр послуг внутрішнім і зовнішнім користувачам, зберігають та обробляють конфіденційну для організації інформацію [12].

Деякі з найпоширеніших типів серверів виділяють [13]:

- веб-сервер;
- email;
- сервер баз даних;
- управління інфраструктурою;
- файлові сервери.

Наприклад, сервер баз даних, який надає послуги бази даних.

Сервери часто стають мішенню для зловмисників через цінність даних, які вони зберігають. Наприклад, на сервері може міститись інформація, яка може бути використана для викрадення особистих даних користувачів [12].

Для того, аби підвищити загальну безпеку сервера, необхідно, перш за все, застосовувати основні рекомендації з забезпечення захисту, але й комбінувати їх із розширеними заходами для усунення вразливостей у серверному програмному забезпеченні.

Найважливішим аспектом розгортання надійно захищеного сервера є ретельне планування. Це гарантія того, що сервер буде максимально безпечним і відповідним усім вимогам й політикам. Можна простежити багато проблем безпеки та продуктивності сервера через відсутність відповідного планування або якісного управління. Але не варто занижувати важливість цих етапів. Розробка плану дає змогу знайти компроміс між ефективністю системи, зручністю її використання та ризиком [12].

У багатьох організаціях структура IT-підтримки дуже фрагментована. Ця фрагментація призводить до проблем з комунікацією й появи невідповідностей у налаштуванні обладнання, що потенційно може призвести до виникнення вразливостей безпеки. Оскільки безпека сервера тісно переплетена із загальною безпекою інформаційної системи організації, до планування сервера, його розгортання та адміністрування має бути залучений персонал IT-відділу.

Організації та компанії повинні впроваджувати певні методи управління безпекою та засоби контролю для підтримки та належної експлуатації сервера. Необхідно передусім розуміти, якими інформаційними активами володіє організація, з'ясувати, які ризики для них існують, та забезпечувати безпеку відповідно до існуючих політик, стандартів та інструкцій. Дана документація допомагає забезпечити конфіденційність, цілісність та доступність ресурсів інформаційної системи. Наразі у глобальній мережі Інтернет є вся необхідна інформація по захисту серверів. Передусім необхідно слідувати рекомендаціям Національного інституту стандартів та технологій (NIST), які викладені у «Посібнику із загальної безпеки сервера». При цьому важливо розуміти, що даний посібник носить рекомендаційний характер, що означає, що досвідчені спеціалісти з інформаційної безпеки можуть впроваджувати свої методи й засоби забезпечення й підтримки безпеки серверного обладнання, однак рекомендації NIST перевірені часом.

Для забезпечення безпеки сервера та допоміжної мережевої інфраструктури, слід запровадити такі методи:

- політику інформаційної безпеки організації;

- контроль та керування змінами;
- оцінку ризиків та управління ними;
- використання ліцензійного програмного забезпечення, що задовольняє політику інформаційної безпеки організації;
- обізнаність персоналу у сфері інформаційної безпеки, навчальні семінари, консультації;
- планування на випадок надзвичайних ситуацій, забезпечення безперервності діяльності та планування аварійного відновлення;
- сертифікацію та акредитацію.

Існують загальні принципи інформаційної безпеки, які необхідно враховувати при розгортанні захищеного сервера [12, 14, 15]:

- простота (Simplicity) – реалізація механізмів безпеки має бути інтуїтивно зрозумілою та простою у використанні, адже складна та заплутана структура (або взагалі її відсутність) може стати причиною проблем безпеки;
- відмовостійкість (Fail-Safe) – збереження працездатності системи у разі збою або відмови складових частин [16];
- відкритий дизайн (Open Design) – безпека не має залежати від таємності реалізації чи компонентів системи;
- розділення привілеїв (Separation of Privilege) – функції мають бути розділені, наскільки це можливо (наприклад, роль системного адміністратора краще відокремлювати від ролі адміністратора бази даних);
- принцип найменших привілеїв (Least Privilege) – користувач має володіти мінімальними правами та доступами, які необхідні для виконання його роботи;
- психологічна прийнятність (Psychological Acceptability) – персонал має розуміти необхідність інформаційної безпеки, що можна досягти через навчання та створення зручного у використанні механізму безпеки;
- найменш повторюваний механізм (Least Common Mechanism) – надання функцій складовим системи таким чином, щоб за певну функцію відповідав єдиний процес або сервіс;

- поглиблений захист (Defense-in-Depth) – одного механізму недостатньо для забезпечення безпеки, тому заходи мають застосовуватись на кожному рівні так, щоб компрометація одного механізму захисту була недостатньою для компрометації хоста чи мережі;
- фактор роботи (Work Factor) – спеціалісти з інформаційної безпеки мають розуміти, що необхідно для зламу їх системи, адже обсяг роботи зловмисника та її вартість має перевищувати цінність інформації, яку він може отримати в результаті компрометації;
- запис подій (Compromise Recording) – усі інциденти безпеки, атаки мають бути задокументовані, зберігатися та бути доступні, оскільки цю інформацію можна використати для побудови більш захищеної системи захисту, а також визначення нападника.

Найчастіше сервери працюють на операційних системах загального призначення. Багато проблем із безпекою можна уникнути, якщо ці ОС налаштовані належним чином.

Для захисту ОС необхідно виконати наступні основні кроки [12]:

- вчасно виправляти помилки та оновлювати ОС;
- встановити та налаштувати необхідні програмні засоби захисту ОС;
- видалити усі непотрібні сервіси та програми;
- налаштувати доступ користувачів;
- розробити парольну політику;
- періодично перевіряти безпеку ОС, щоб переконатися, що попередні кроки вирішують питання забезпечення захисту.

Після розгортання сервера адміністратори повинні постійно підтримувати його безпеку. До важливих заходів підтримки безпеки сервера відноситься:

- обробка і аналіз файлів логування;
- виконання регулярного резервного копіювання сервера;
- відновлення після компрометації сервера;
- регулярне тестування безпеки та безпечне віддалене адміністрування [17].

1.3 Сучасні вимоги до безпеки паролів користувачів

Дослідження Yubico та Ponemon «State of Password and Authentication Security Behaviors», опубліковане у 2019 році [18], показало, що:

- двоє з трьох респондентів (69%) діляться своїми паролями з колегами для доступу до облікових записів;
- 51% респондентів використовують паролі від акаунтів повторно;
- 57% респондентів, що зазнали фішингової атаки, не змінили свій підхід до захисту парольних даних;
- 67% респондентів не використовують двофакторну автентифікацію для особистих облікових записів та 55% – для робочих;
- 57% респондентів надають перевагу методу входу, що не передбачає введення паролю.

«IT-спеціалісти чи ні, люди не хочуть бути обтяженими безпекою – вона повинна бути зручною, простою та працювати миттєво», – сказала Стіна Еренсверд, генеральна директорка та співзасновниця Yubico. «Протягом багатьох років досягнення балансу між високою безпекою та простотою використання було майже неможливим, але нові технології автентифікації, нарешті, заповнюють цей розрив. Завдяки доступності безпарольного входу та ключів безпеки настав час для компаній посилити свої параметри безпеки. Організації можуть працювати набагато краще, ніж паролі, насправді користувачі цього вимагають» [19].

Система часто дозволяє користувачам використовувати ненадійні паролі. До того ж, не всі сервіси підтримують двофакторну автентифікацію, що є значним недопрацюванням політики безпеки. Іноді користувачі змушені недбало поводитись зі своїми парольними даними через непослідовність встановлених для паролів правил [20]. Усе це загрожує порушенням цілісності, конфіденційності та доступності інформації, що зберігається в облікових записах користувачів.

Відповідальні за підтримку надійності паролів та захист цих даних саме розробники системи та спеціалісти з інформаційної безпеки, адже саме ці люди можуть або заохочувати використання надійних паролів або перешкоджати цьому.

Спеціалісти мають дотримуватись сучасних вимог до безпеки паролів користувачів (табл. 1.1) [20].

Таблиця 1.1

Вимоги до безпеки паролів користувачів

Правильно	Неправильно
Мати обмеження по мінімальній довжині пароля та дозволяти довгі паролі.	Мінімальна довжина пароля менше 8 символів.
Дозволити якомога більший можливий набір символів, наприклад UTF-8, включаючи емодзі.	Застосовувати довільні обмеження, наприклад, забороняти використання певних символів.
Показувати, що всі вимоги щодо пароля виконані або забезпечити безпечний генератор паролів.	Сказати користувачеві, що його пароль «надійний».
Перевіряйте нові паролі на витік у базі даних, та іншу інформацію облікового запису.	Дозволити використання P@ssw0rd та qwerty123 або інших поширених паролів.
Дозволити вставлення пароля.	Обрізати текстове поле пароля.
Підтримка багатофакторної автентифікації.	Підтримка лише багатофакторної автентифікації на основі SMS.
Мати надійні способи скидання пароля.	Використовувати секретні запитання.
Надсилати повідомлення про помилки, які не розкривають інформацію зловмисникові.	Давати підказки у повідомленнях про помилки, наприклад «неправильне ім’я користувача» або «неправильний пароль».
Відстежувати ненормальну поведінку облікового запису та спілкуватися з користувачами.	Примусово регулярно змінювати пароль за розкладом.
Дозволяти відновлення облікового запису ще деякий час після його видалення.	Давати користувачам можливість легко заблокувати себе назавжди.
Використовувати сіль і гешувати паролі в базі даних.	Придумати власний алгоритм гешування.
Шифрувати всі дані під час передачі та в стані спокою.	Зберігати будь-які паролі у вигляді простого тексту паролю.

Існують програми, розроблені спеціально для оцінювання надійності паролів [21]. Імплементуючи загальну концепцію таких програм у свій додаток або сервіс, розробники можуть значно полегшити роботу собі та запевнити користувачів у міцності створених ними паролів. Під час реєстрації користувача необхідно також давати зворотній зв'язок з метою надання можливості підвищити надійність створюваного пароля. Для цього можна вказувати мінімальні вимоги, наприклад:

- Пароль має бути мінімум 12 символів довжиною. Чим пароль довший – тим він надійніший [22]. Це може бути, наприклад, цитата з промови відомої людини або з фільму, рядок із вірша або пісні, важлива особисто для користувача фраза у формі абревіатури (перші літери кожного слова) [23].
- Використання літер верхнього та нижнього регістрів, чисел та символів, оскільки це значно ускладнює роботу для зловмисника. Варто заміняти деякі літери або цифри спеціальними символами, наприклад, @ ! #. Але не слід робити очевидні заміни, наприклад, заміняти і на 1 чи !, або а на @ [24].
- Не використовувати слова зі словників, імена осіб або продуктів та організацій. Пароль не буде надійним, якщо в ньому використовуються ініціали або псевдоніми користувача, його день народження, імена домашніх улюбленців або дітей, місце проживання, оскільки цю інформацію легко знайти в Інтернеті або соціальних мереж самого користувача. Також не варто використовувати очевидні слова, такі як password; прості послідовності літер або чисел, наприклад: abcdefg, 12345678, 11111111; та послідовності символів з квавіатури, як-от: qwerty, asdfgh, zxcvbn [23].
- Не використовувати пароль повторно, оскільки це може загрожувати зломом одразу декількох акаунтів. Для особливо важливих облікових записів однозначно варто придумувати унікальні паролі.
- Намагатися вигадати такий пароль, який ви зможете запам'ятати, але іншим користувачам (та можливим зловмисникам) буде складно здогадатися.
- Не використовувати пароль, який легко вгадати знайомим людям, або ті, в яких згадується загальнодоступна інформація з соціальних мереж користувача [23].

Необхідно нагадувати користувачам, що створення надійного пароля допоможе їм захистити свої персональні дані, зберегти конфіденційність їх файлів, листувань та інших матеріалів, а також попередити несанкціонований доступ до їх облікового запису та можливий витік інформації [23].

Надавайте користувачам рекомендації про поводження з парольними даними, яких варто дотримуватись, щоб захистити свій обліковий запис, адже користувачі можуть і не здогадуватись, що наражають на небезпеку свої дані:

- Не надавати свої парольні дані друзям, родичам та незнайомцям.
- Не передавати свої паролі електронною поштою або тими засобами зв'язку, в безпеці яких не можна бути впевненим.
- Варто створювати унікальні паролі для кожного з ресурсів або додатків, де реєструєтесь. Запам'ятати таку кількість інформації досить складно, тому можна порекомендувати користувачам довіряти свої паролі надійним та перевіреним менеджерам паролів, які будуть зберігати ці паролі в зашифрованому вигляді.
- Не можна записувати паролі на наліпках або аркушах паперу та класти їх у видне місце, оскільки це дуже підриває захищеність облікових записів. Якщо без цього не можна обійтися, краще замість того, аби записувати безпосередньо пароль, записати підказку до нього. Наприклад, якщо пароль «BaliJan#33Vacatio3n0» – можна скористатися підказкою «Улюблена подорож сестри».
- Необхідно змінювати пароль, якщо є підозра, що його викрили, або помітна стороння присутність в обліковому записі.
- За можливості треба використовувати багатофакторну автентифікацію, оскільки це створює додатковий рівень безпеки на випадок, якщо пароль буде викрадений [22].

Розробники системи повинні постійно відслідковувати аномальну поведінку в профілях користувачів, наприклад, за допомогою створення спеціального програмного забезпечення, яке б реагувало на такі дії. Перш за все, треба мати уявлення, як виглядає «нормальна» поведінка у вашому застосунку чи програмі. Хоча кожна програма є по-своєму унікальною, існують загальні «прапорці» для моніторингу поведінки користувачів:

- Активність. Слідкуйте, як часто користувачі користуються програмою та заходять у свій облікових запис. Якщо зазвичай це декілька разів на тиждень, але 10 разів за останні 25 хвилин, то таку поведінку необхідно позначити як анормальну.
- Географія. Відстежуйте географічне положення запитів на авторизацію. Якщо зазвичай користувач входив у систему з України, але раптово намагається зайти з Китаю, то така поведінка – ненормальна.
- Метод. З'ясуйте методи входу користувачів. Якщо користувач завжди авторизується з мобільного застосунку, але несподівано намагається зробити це іншим чином – оцінюйте таку поведінку як ненормальну.
- Невдалі спроби. Ненормальним вважається, якщо користувач робить багато невдалих спроб під час входу в систему.
- Незавершена багатофакторна автентифікація. Якщо користувач намагається пройти автентифікацію, а «другий фактор» його зупиняє, то варто оцінювати таку поведінку як аномальну.

Розробникам також варто пам'ятати, що ненормальна поведінка незавжди означає, що по той бік знаходиться зловмисник, але краще все одно реагувати на усе підозріле. В залежності від специфіки програми або додатку методи оцінки такої поведінки також можуть змінюватись.

У разі виявлення ненормальної поведінки в профілі користувача варто надіслати йому повідомлення з нейтральною інформацією про цю поведінку, яка не розкривала б важливої інформації. Наприклад: «Нещодавно ми помітили ненормальну поведінку у Вашому профілі (можна перелічити, що саме здалося незвичним). Якщо це Ваші дії, натисніть, будь ласка, на зелену кнопку. В іншому випадку – натисніть на червону кнопку, щоб оповістити наш відділ безпеки» (рис. 1.1) [20].

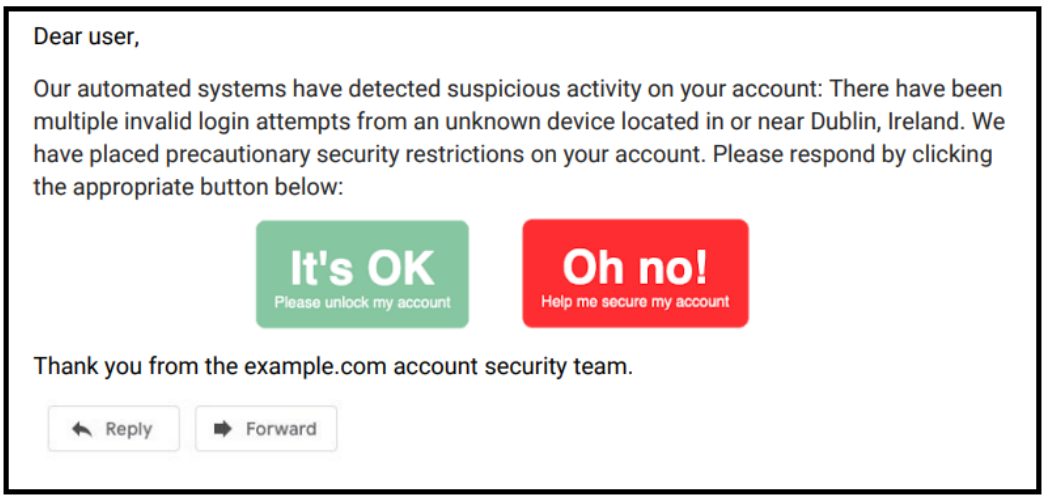

Рисунок 1.1 – Приклад повідомлення про ненормальну поведінку в обліковому записі

Якщо відповідь не надійде у певний проміжок часу – слід примусово скинути пароль або обмежити деякі функції, поки користувач не надасть відповідь. Наприклад, заборонити внесення змін до профілю або оплату, про що також варто повідомити.

Однак для того, щоб розробники системи та спеціалісти з інформаційної безпеки мали можливість вести діалог з користувачами та повідомляти про підозрілі дії в обліковому записі – користувач має додати резервний номер телефона та адресу поштової скриньки. Необхідно донести користувачеві, що це важливі складові забезпечення безпеки, оскільки їх можна використовувати для блокування доступу до облікового запису для сторонніх осіб, на них можна отримувати повідомлення від служби безпеки про підозрілі дії в профілі, а також відновити доступ до акаунту у разі втрати пароля [25].

Для забезпечення й підтримки безпеки власних акаунтів та даних, що там зберігаються та циркулюють, необхідно завжди вчасно оновлювати операційну систему, браузер або відповідні додатки, оскільки застарілі версії можуть мати певні вразливості та дірки в безпеці, а в останніх версіях ці вразливості можуть бути виправлені. Тільки співпрацюючи з користувачем, розробники можуть вберегти його дані від злому, тому ознайомлення користувачів з можливими ризиками та способами їх мінімізації – важливий етап забезпечення безпеки.

1.4 Концепції зберігання паролів

І існуючих додатках та іншому програмному забезпечення основним способом автентифікації користувачів є паролі, тому необхідно безпечно їх зберігати. Користувачі можуть реєструватися в додатках, автентифікуватися та за потреби змінювати паролі, а зловмисники не повинні розшифрувати збережені паролі, навіть якщо отримають доступ до бази даних, де ці паролі зберігаються. Для збору, передачі та зберігання парольних даних існує декілька концепцій (рис. 1.1) [26].

Рисунок 1.2 – Методи зберігання паролів

Найлегшим методом є зберігання паролів відкритим текстом безпосередньо у базі даних [26]. Це також найнебезпечніший метод, як мінімум, тому що адміністратори бази даних бачать паролі, які використовують користувачі. При компрометації бази даних зловмисником облікові дані будуть з легкістю вкрадені. Згідно з дослідженням Yubico та Ponemon за 2019 рік, 51% респондентів зізналися, що використовують паролі повторно [18], що буквально відкриває для зловмисників

доступ одразу до декількох облікових записів користувачів, наприклад, до GMail, Facebook і Twitter.

Досить легким та відносно небезпечним методом зберігання паролів є використання простого алгоритму гешування, наприклад, SHA-256 або MD5, які піддаються перебору за словником. Якщо зловмисники отримають доступ до бази даних, вони зможуть використати словник, де зібрані найбільш поширені паролі, та досить швидко встановити реальний пароль, оскільки в атаці такого типу геш пароля зі словника порівнюється з гешем з бази даних. Такий метод зберігання паролів також є досить небезпечним, оскільки дізнавшись пароль від одного облікового запису, зловмисник, вірогідно, зможе отримати доступ і до інших акаунтів цього користувача. Також не варто придумувати власний алгоритм гешування, оскільки він не перевірений та також може виявитись слабким до атак [26].

Більш складним та відносно безпечним методом зберігання паролів є використання солі разом з гешем. Сіль – це унікальний, випадково згенерований рядок, який додається до кожного пароля як частина процесу гешування [27]. Сіль може використовуватись одна й та сама, проте використання унікальних солей значно підвищує безпеку паролів [28]. Оскільки сіль є унікальною для кожного користувача, зловмисник повинен зламати геші по одному, використовуючи відповідну сіль, а не обчислювати геш один раз і порівнювати його з кожним збереженим гешем. Це значно ускладнює викриття великої кількості паролів, оскільки на таку атаку треба більше часу [27].

Методом, який підвищує безпеку зберігання паролів, є використання перцю як додаткового захисту разом із сіллю. Мета цього методу — запобігти зловмиснику зламати будь-який геш, якщо він має доступ лише до бази даних . Перець не є унікальним і не повинен зберігатися в базі даних, на відміну від солі. Його треба зберігати у секретних сховищах або, наприклад, апаратному модулі безпеки (HSM) [27].

Висновки за розділом 1

Проблема забезпечення і підтримки безпеки парольних даних користувачів досить складна та багатогранна, оскільки найпоширенішою є автентифікація саме на основі пароля, тож ці дані необхідно ретельно захищати. Щоб зберегти цілісність, конфіденційність та доступність інформації, необхідно дотримуватись комплексного підходу до захисту даних користувачів.

Перш за все, має бути максимально захищена база даних, у якій ці дані зберігаються, адже її компрометація загрожує витоком облікової інформації користувачів та подальшим зломом інших їх акаунтів, зважаючи на те, що кожний другий користувач використовує один пароль мінімум два рази. Сервер бази даних також має розташовуватись у безпечному та захищеному приміщені. Якщо використовується хмарний сервер – про безпеку обладнання має подбати постачальник послуг.

Відповідальними за безпеку паролів є розробник системи та інженер інформаційної безпеки, адже вжиті ними заходи будуть мати безпосередній вплив на безпеку використання та зберігання облікових даних користувачів. Тому спеціалісти мають дотримуватись перевірених практик та рекомендацій. Необхідно мати можливість створення довгого пароля з використанням широкого спектру символів, повідомляти користувачу про ступінь надійності створеного ним пароля, відслідковувати аномальну поведінку в обліковому записі, обов’язково шифрувати усі дані й під час передачі й у стані спокою. Найкращим методом зберігання паролів у базі даних є використання алгоритмів гешування у поєднанні з сіллю та перцем, що забезпечує надійний захист парольних даних та значно ускладнює роботу для зловмисника. Спеціалісти з інформаційної безпеки організації мають визначити цінність інформації, яка циркулює по системі або зберігається в ній, розуміти ризики виникнення інцидентів безпеки та забезпечити такий захист, злом якого коштував би зловмиснику дорожче, ніж та інформація, на яку він полює.

РОЗДІЛ 2
ПОРІВНЯЛЬНИЙ АНАЛІЗ АЛГОРИТМІВ ГЕШУВАННЯ ПАРОЛІВ

2.1 Визначення, властивості та застосування геш-функцій

Серед усього різноманіття проблем інформаційної безпеки завдання забезпечення цілісності, конфіденційності та доступності переданої та збереженої інформації є одним із найголовніших. Це завдання може перетворитися на справжню проблему, враховуючи вимоги сучасності до інформаційно-телекомунікаційних систем. Завдання і проблеми такого роду можуть вирішуватись за допомогою криптографічних методів та засобів. Особливо актуально це у фінансовій сфері, мобільних додатках, оскільки забезпечення цілісності та конфіденційності даних є необхідною умовою надійного функціонування системи [29].

Одним із дієвих методів криптографічного захисту інформації є використання геш-функцій. За визначенням тлумачного словника англійської мови, гешування означає «розрізання на дрібні шматочки», щоб це виглядало як «заплутаний безлад» [30]. Це визначення безпосередньо стосується того, що являє собою гешування в обчислювальній роботі.

Використання односторонніх функцій (які можуть бути без стиснення) для автентифікації було зазначено в роботі Діффі та Хеллмана «Нові напрямки в криптографії» [31], хоча вони не використовували термінологію «геш-функція». Вони стверджували, що потребують опору прообразу (тобто односторонності) й опору другого прообразу. Термін «геш-функція» походить від Меркла [32] або Рабіна [33]. Обидвох особливо хвилювала тема автентифікації повідомлень, хоча не обов'язково через криптосистеми з відкритим ключем.

Гешуванням називається процес виконання одностороннього перетворення будь-якого рядка байтів в інший детермінований новий рядок символів [20]. «Детермінований» означає, що однакове вхідне повідомлення завжди буде давати однаковий (визначений) геш. Детермінована функція – це та функція, яка при

однакових вхідних даних завжди буде давати однакове вихідне значення. Це властивість, але відіграє ключову роль під час автентифікації, адже має бути гарантія, що один і той самий пароль кожного разу буде видавати один і той самий геш. Дана властивість робить геш-функції придатними до використання для перевірки парольних даних користувачів. Іншими словами, геш-функція є алгоритмом, що приймає на вхід довільний блок даних, але повертає рядок визначеного розміру, яке називається геш-значенням (рис. 2.1) [34], при якому будь-які модифікації вхідних даних змінять кінцеве значення гешу.

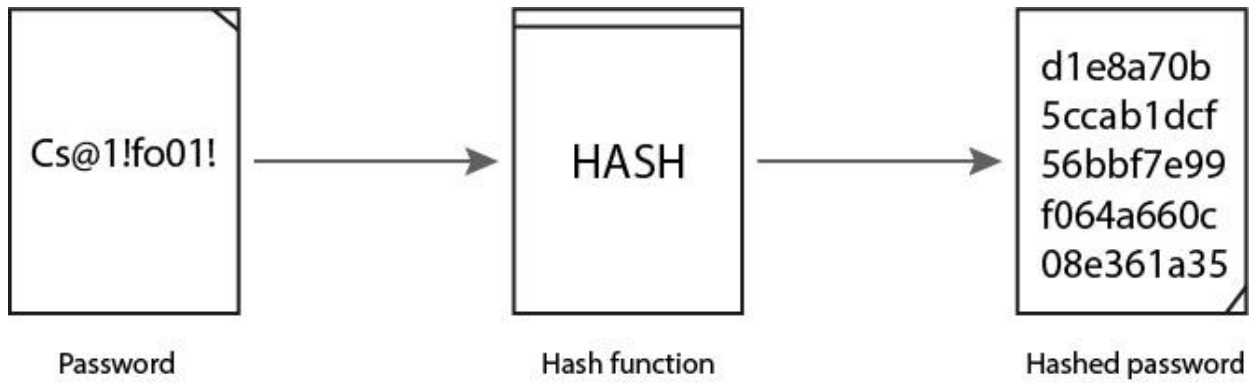

Рисунок 2.1 – Приклад роботи алгоритму гешування

Криптографія з відкритими ключами покликана була вирішити нагальні проблеми та завдання. Першим завданням, як зазначалося вище, є зберігання паролів. Але паролі ж можуть зберігатися не десь там у базі даних на віддаленому для користувача сервері, а й на самому комп’ютері, якщо мова йде про парольні дані від входу у систему цього комп’ютера під певним юзером. Така інформацію не може зберігатися у відкритому вигляді, тому що зловмисник, маючи доступ до диску комп’ютера, може прочитати цей пароль та використати для несанкціонованого доступу. Це найлегше буде зробити, наприклад, якщо зловмисник – системний адміністратор та має безпосередній доступ до обладнання.

Друге завдання виникло й стало актуальним з появою радіолокаторів та системи протиповітряної оборони. Це особливо важливо у наші часи. Працює це так, що при перетині кордону радіолокатор запитує пароль у літака. Якщо названий пароль є правильний – літак без перешкод може перетнути кордон, якщо ні – такий літак ідентифікують як «чужий». Проблема проявляється у тому що, це пароль

передається по відкритому каналу і може бути підслуханий зловмисником і використаний ним потім, як привід вважати його за «свого».

Третє завдання, яке покликана вирішувати криптографія з відкритим ключем, пов'язане з попередніми і виникає в комп'ютерних мережах із віддаленим доступом, наприклад, при спробі авторизації користувача в системі, оскільки після введення ідентифікатора (логіна) необхідно ввести секретний пароль.

Ці та інші проблеми та завдання можна вирішити за допомогою криптографічних методів та безпосередньо односторонньої функції (one-way function) [35].

«Одностороннє перетворення» означає те, що неможливо перетворити геш назад у початковий вигляд паролю, тобто легко перетворити у геш, але складно зробити обернену операцію. Використання односторонніх геш-функцій передбачається у генераторах псевдовипадкових чисел, електронних підписах, схемах ідентифікації та шифруванні з відкритим ключем [36]. Загальні інструменти односторонніх функцій запропонували Діффі й Хеллман, які започаткували криптографію з відкритим ключем ще у 1976 році [37].

Існує декілька визначень односторонньої функції – сильна, слабка та детермінована:

- можна вважати сильною односторонньою функцією, якщо всі ефективні ймовірнісні алгоритми інверсивного перетворення досягають успіху з мізерно малою ймовірністю;
- можна вважати слабкою односторонньою функцією, якщо всі ефективні імовірнісні алгоритми інверсивного перетворення зазнають невдачі з ймовірністю, якою не можна знехтувати;
- одностороння функція вважається детермінованою, якщо вона стійка до детермінованих алгоритмів інверсивного перетворення.

Цікавим фактом про сильні та слабкі односторонні функції є те, що, хоча їх визначення не є еквівалентними, слабкі односторонні функції існують тоді й тільки тоді, коли існують сильні односторонні функції [38].

Особливості геш-функцій:

- зазвичай геш-значення є меншим, ніж вхідні дані;
- вхідні дані можуть бути будь-якого розміру, а геш-значення є фіксованим;
- розрахування гешів – це проста та швидка операція [39].

Існують деякі вимоги та властивості, яким має відповідати функція гешування, щоб її використання вважалося безпечним та зручним:

- опір прообразу: при даному значенні y неможливо віднайти таке x, що h(x) = y;
- опір другого прообразу: при даному y та x^1 таким, що $h(x^1) = y$, повинно бути неможливим знаходження $x^2 \neq x^1$ такому, що $h(x^2) = y$;
- стійкість до колізій: неможливо знайти два будь-які x^1 та x^2, при яких їх значення гешів були б однакові $h(x^1) = h(x^2)$ [40];
- неможливо змінити початкове повідомлення без зміни його геш-значення;
- швидкість роботи: обчислення геш-значення будь-якого за розміром тексту має бути швидким;
- кінцевий результат має залежати від кожного байту відкритого текста та від їхнього взаємного розташування (рис. 2.2) [26];

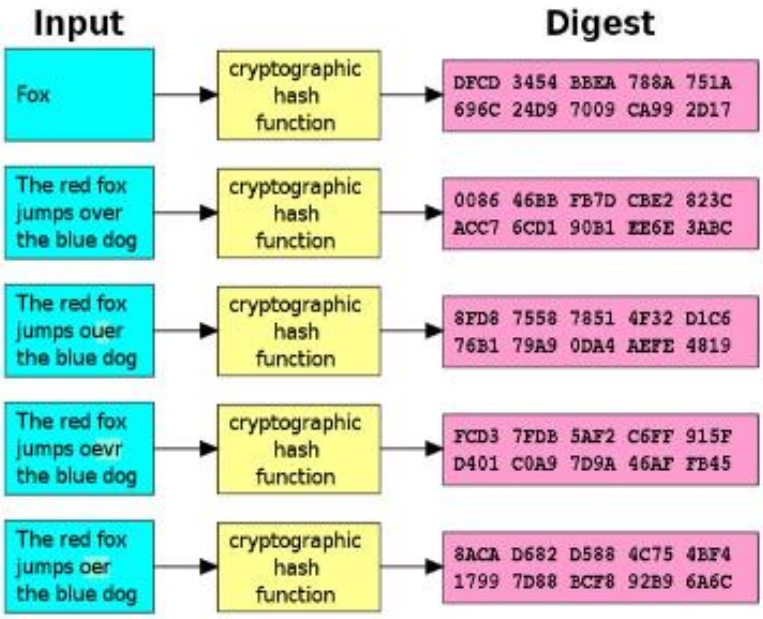

Рисунок 2.2 – Вплив модифікації початкового тексту на геш-значення

Геш-функції широко застосовуються для розв’язування питань, пов’язаних із забезпеченням захисту даних, наприклад, для гешування паролів користувачів з метою подальшого їхнього зберігання у базі даних. Цей метод застосовується в ОС Windows NT (використовується геш-функція MD4 разом з DES). Геш пароля не шифрується, коли зберігається у базу даних. Коли користувач намагається здійснити вхід в обліковий запис, модуль персоналізації бере наданий користувачем пароль, виконує аналогічне одностороннє гешування і порівнює результат операції зі значення гешу, що знаходиться у базі даних. Якщо геші збігаються – пароль введено правильно і вхід виконано успішно. Функція гешування може служити за криптографічну контрольну суму – код виявлення змін (MDC) або для перевірки цілісності повідомлення (MIC) [29].

Використання геш-функцій набуло такої розповсюдженості через їх важливу характеристику отримувати з відкритого тексту довільної множини значення гешу значно меншої довжини, що дозволяє скоротити мережевий трафік. Застосування геш-функцій надає можливість боротися з надлишковістю даних До того ж, геш-значення може підлягати подальшому шифруванню з метою забезпечення більш надійного захисту інформації у стані спокою та під час передачі. Враховуючи те, що геш-значення зазвичай менше, ніж початковий текст, це позитивно вплине на якість виконання подальших криптографічних перетворень [29].

Геш-функції можуть використовуватись з метою:

- захисту паролів під час їхнього передавання чи зберігання у базі даних;
- перевірки цілісності даних, їх модифікації за допомогою контрольних сум [41];
- формування електронного підпису для гарантування автентичності повідомлення та підтвердження авторства [29];
- виявлення повторювальних записів, оскільки однакові блоки тексту дають один і той самий геш [41];

Наразі криптографічні геш-функції настільки широко застосовуються та допомагають у «буденних» справах розробників системи та спеціалістів з

інформаційної безпеки, що вже реалізовані як вбудовані функції в стандартних бібліотеках сучасних мов програмування [26].

2.3 Порівняльний аналіз розповсюджених функцій гешування

Для збору, передачі та зберігання паролів потрібно застосовувати алгоритми гешування, які допоможуть забезпечити безпеку. У базі даних мають зберігатись криптографічно надійні геші паролів, які неможливо змінити [20].

Алгоритм гешування приймає пароль як відкритий текст і перетворює його в геш-значення, враховуючи три параметри:

- геш-функція;
- ітерації;
- сіль.

Головним параметром алгоритму гешування є використовувана геш-функція. Ітерації є необов'язковим параметром, що вказує на кількість повторень виконання функції для обчислення геш-значення. Наприклад, якщо алгоритм гешування використовує геш-функцію MD5 і кількість ітерацій дорівнює 50, тоді вона виконає 50 повторень функції MD5 для обчислення гешу пароля. Кількість ітерацій можна налаштувати так, щоб обчислення геш-значення займало як завгодно кількість обчислювального часу (також відомого як розтягування ключа). Таким чином, додавання такого параметру як ітерації можуть використовуватись для сповільнення атак підбору пароля.

Останній параметр – сіль – також не є обов’язковим [42]. Найчастіше, сіль додається на початок пароля. Потім нове значення відкритого тексту «сіль+пароль» підлягає подальшому гешуванню. Значно підвищує рівень безпеки використання унікальних солей для кожного пароля [28], оскільки райдужні таблиці (rainbow table) стають неефективними. Тобто зловмисник не буде знати наперед, що яке значення солі застосовується, і тому не може попередньо обчислити райдужну таблицю [42].

Для гешування паролів використовуються численні функції, зокрема: MD5, SHA1, SHA256 - SHA512, PBKDF2, BCRYPT, SCRYPT та Argon2.

Алгоритм MD5 розроблений у 1992 році для досить швидкої роботи на 32-розрядних машинах та є розширенням алгоритму MD4 [43].

MD4, розроблений Рональдом Рівестом у 1990 році, дав назву цілому набору геш-функцій, оскільки це була перша геш-функція такого роду. Для створення інших геш-функцій цієї родини була взірцем саме MD4. До них належать такі геш-функції як MD5, SHA-0, HAVAL, SHA-1, SHA-2 та інші.

Можна визначити наступні характеристики сімейства MD4 [44]:

- Вони побудовані на основі (варіантів) конструкції Меркле-Дамгарда.
- Функцію стиснення часто можна розглядати як блочний шифр у режимі Девіса-Меєра (отже, блок повідомлення є ключем, а ланцюжок input є блоком відкритого тексту блочного шифру) – однак цей «блочний шифр» у всіх випадках був розроблений спеціально для геш-функції, про яку йдеться.
- Функція стиснення складається з відносно великої кількості простих кроків, що кожне оновлення одного або кількох регістрів у стані, що містить від чотирьох до приблизно восьми регістрів.
- Кожен регістр є 32-розрядним або 64-розрядним значенням.

MD5 було розроблено, оскільки вважалося, що MD4 був взятий у використання швидше, ніж встигли оглянути його з критичної точки зору. Оскільки MD4 був розроблений як надзвичайно швидкий, з точки зору ризику успішної криптоаналітики він знаходиться «на межі». MD5, поступаючись трохи в швидкості, вважається більш безпечним [43]. Тим не менш, в одній із перших статей, у якій розкривається атака зіткнення (collision attack) функції MD5, ставиться під сумнів покращення, зроблені в MD5 порівняно з MD4 [45]. Атакою зіткнення називається спроба віднайти два прообрази, які будуть давати однакові значення гешу. Однак MD5 використовує чотири раунди замість трьох унікальних констант для кожного кроку, і нову булеву функцію, і вона набагато довше протистояла нищівним атакам, ніж MD4 [44].

MD5 приблизно на 33% повільніше від MD4, тому що додається ще один раунд. Як згадувалося, незабаром після створення було виявлено колізію в функції

MD5, але знадобилося чимало років, перш ніж було здійснено напад на функцію [44].

Орган стандартизації США NIST опублікував Secure Hash Standard у травні 1993 року. Геш-функція, що лежить в основі стандарту, отримала назву SHA, але сьогодні її частіше називають SHA-0.

SHA-0 також побудований на тих же принципах, що і MD4. Однак SHA-0 є 160-бітною геш-функцією, яка зберігає п'ять регістрів у стані: a, b, c, d та e. Ця функція замінений на SHA-1 у 1995 році. SHA-1, можливо, є найбільш поширеною криптографічною геш-функцією. Вона зустрічається у величезній кількості криптографічних стандартів, протоколів, схем тощо.

У 2017 році Google представив докази того, що функція гешування SHA-1 не є безпечною для використання з метою захисту даних [46]. Це було кульмінацією дворічних досліджень Інституту CWI в Амстердамі сумісно з Google, що виступає за припинення підтримки SHA-1 протягом багатьох років, особливо коли йдеться про підписання сертифікатів TLS. Уже в 2014 році команда Chrome оголосила, що вони поступово припинятимуть використання SHA-1 [47]. Усе через знайдену колізію (зіткнення) в даній геш-функції. Зіткнення відбувається, коли дві різні частини даних (документ, двійковий файл або сертифікат веб-сайту) гешуються в той самий дайджест (рис. 2.3) [46].

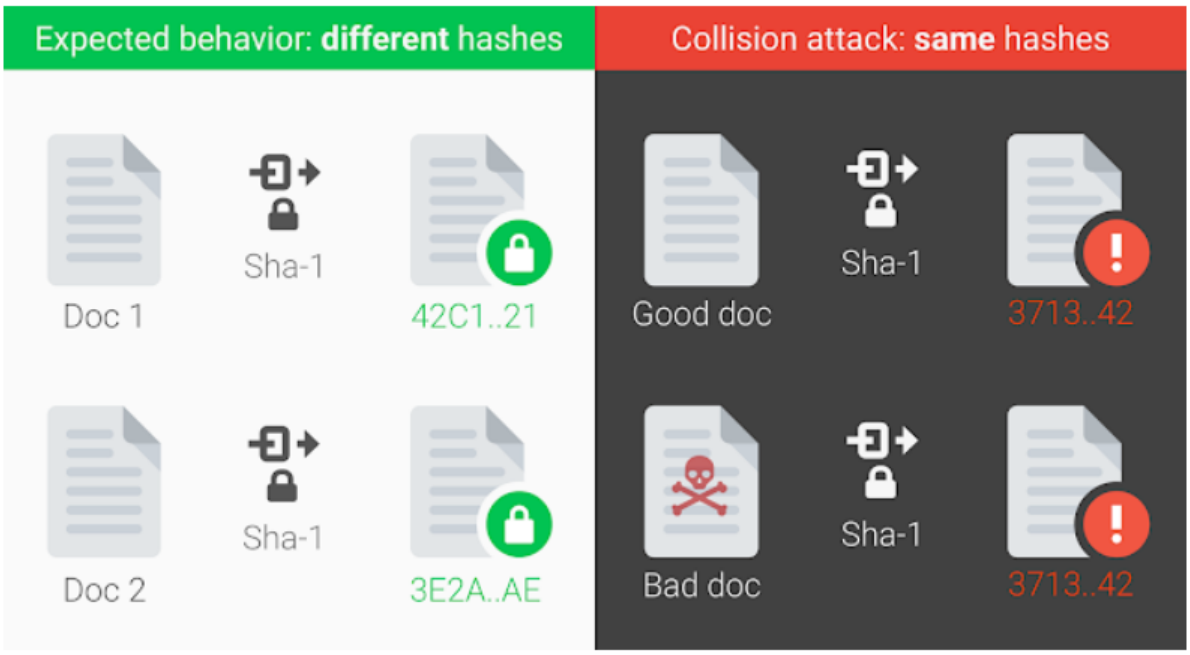

Рисунок 2.3 – Колізія SHA-1

На практиці ніколи не повинні відбуватися колізії для безпечних геш-функцій. Однак якщо алгоритм гешування має деякі недоліки, як SHA-1, добре фінансований зловмисник може створити зіткнення. Потім зловмисник може використати цю колізію, щоб обдурити системи, які покладаються на геші, щоб прийняти шкідливий файл замість його доброякісного аналога. Наприклад, два договори страхування з кардинально різними умовами.

З публікацією Advanced Encryption Standard [46] у 2001 році з'явилася потреба у нових геш-функції з більшими вихідними розмірами, щоб відповідати розмірам ключа AES. Це призвело до розробки трьох нових геш-функцій – SHA-256, SHA-384 і SHA-512, які спільно називають SHA-2, опубліковані в 2002 році [49]. SHA-256 і SHA-512 відрізняються розміром слова: SHA-256 використовує 32-розрядні слова, а SHA-512 використовує 64-розрядні слова. Кількість кроків у SHA-256 становить 64 і 80 в SHA-512 [44].

Перераховані геш-функції – MD5, SHA1, SHA256, SHA512 – не вимагають використання солі за замовчуванням. Таким чином, необхідно використовувати окрему функцію для створення солі під час процесу гешування. З іншого боку, решта геш-функцій внутрішньо генерують і використовують випадкову сіль під час обчислення геш-значень.

Існує багато прикладів більш нових гешів, які більше підходять для вирішення проблем за задач сьогодення, наприклад, їх можна застосовувати під час зберігання паролів. Це такі функції гешування як PBKDF2, Argon2, Scrypt та Bcrypt [20], порівняння яких міститься у таблиці 2.1.

Як згадувалось раніше, параметр ітерацій визначає кількість послідовних виконань використовуваної геш-функції, збільшуючи час обчислення геш-значення. З цієї причини геш-функції PBKDF2, BCRYPT, SCRYPT і Argon2 використовують ітерації за замовчуванням. «Геші, якщо використовуються для безпеки, мають бути повільними», – наголошував Джефф Етвуд, співзасновник Stack Overflow і Discourse [50]. Криптографічна геш-функція, яка використовується для гешування паролів, повинна обчислюватися повільно, оскільки швидко обчислюваний алгоритм може зробити атаки грубої сили більш результативними, особливо з урахуванням потужності сучасного обладнання, що стрімко розвивається.

Таблиця 2.1

Порівняння сучасних алгоритмів гешування

Password Hashing Algorithm	**PBKDF2**	**BCRYPT**	**SCRYPT**	**Argon2**
Originating Year	2000	1999	2009	2015
Memory Intensive	-	+	+	+
Computation or Time Intensive(CPU)	+	+	+	+
Recommended input parameters	Work factor = 20,000 to 100,000	Cost factor of 10 to 31	N=16384, r=8, p=1	Number of iterations =3 Memory usage = 2^{12} Kibibyte, Parallel threads =1
Known Attacks	Brute force is relatively easy with	Resistant to ASICs and GPUs but	Scrypt if it only uses 1MB memory is	Two successful attacks against Argon2i

Password Hashing Algorithm	PBKDF2	BCRYPT	SCRYPT	Argon2
	emergence of ASIC, FPGA and/or GPUs	vulnerable to to FPGAs	vulnerable to GPU attacks	version
Well known implementations	Apple IOS uses 20,000 iterations, LastPass, WPA2	OpenBSD, SuseLinux	Used in many Cryptocurrencies (LiteCoin), used in Chromium OS to protect user's vault keyset	Relatively new

Можна досягти цього, уповільнивши обчислення гешування, використовуючи багато внутрішніх ітерацій або завантаживши пам’ять обчислень. Повільна криптографічна геш-функція гальмує цей процес, але не зупиняє його, оскільки швидкість обчислення гешування впливає як на користувачів, які мають добрі наміри, так і на зловмисників, які переслідують мету викрасти конфіденційну інформацію. Геш-функція має бути повільна, але при цьому і зручна у використанні.

PBKDF2 є найпростішою функцією, оскільки використовує зазвичай SHA256 або SHA512, й рекомендована NIST. Дана функція гешування вимагає вибору внутрішньої функції. Робочий коефіцієнт (work factor) для PBKDF2 реалізується за допомогою підрахунку ітерацій, який встановлюється на основі використаного внутрішнього алгоритму гешування [27]:

- PBKDF2-HMAC-SHA1: 720 000 ітерацій;
- PBKDF2-HMAC-SHA256: 310 000 ітерацій;
- PBKDF2-HMAC-SHA512: 120 000 ітерацій.

Якщо PBKDF2 використовується з HMAC, а пароль довший за розмір блоку геш-функції (64 байти для SHA-256), пароль буде автоматично попередньо гешований. Наприклад, пароль «Цей пароль значно довший за 512 біт, які є

розміром блоку SHA-256» перетворюється на геш-значення 16d62a5de7dc8e762304fa6b5aa92e59f146b39b8bfce6126a3c24cda47702e2 (рис. 2.4).

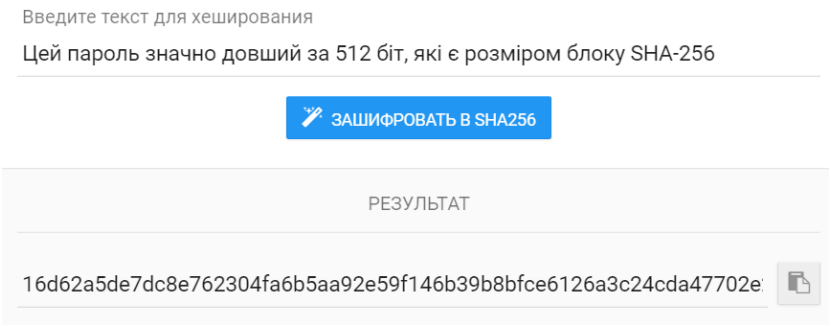

Рисунок 2.4 – Результат гешування паролю

Хороша реалізація PBKDF2 виконає цей крок перед достатньо коштовною ітераційною фазою гешування, але деякі реалізації виконують перетворення на кожній ітерації. Це може зробити гешування довгих паролів значно дорожчим, ніж гешування коротких паролів. Якщо користувач необмежений у довжині пароля, існує потенційна вразливість для відмови в обслуговуванні, наприклад, опублікована в Django в 2013 році [51]. Попереднє гешування може зменшити цей ризик, але вимагає додавання солі на етапі попереднього гешування.

З іншого боку, BCRYPT, який базується на алгоритмі шифрування blowfish використовує ітерації лише у функції налаштування ключа Blowfish та бере параметри сіль і пароль як вхідні дані. Дана функція гешування має обмеження по довжині вхідних даних – 72 байти. Аби не виникало вразливостей, користувач має бути обмежений у довжині пароля відповідно до 72 байтів (або менше, якщо реалізований алгоритм має ще менші обмеження). Але тут також можна скористатися попереднім гешуванням, оскільки це вирішить проблему з надмірністю даних та пришвидшує роботу алгоритму. Тобто спочатку гешувати введений користувачем пароль за допомогою, наприклад, SHA-256, а потім гешувати отриманий геш, використовуючи BCRYPT [42].

З іншого боку, SCRYPT і Argon2 належать до категорії геш-функцій, які називаються жорсткими функціями пам'яті (MHF) і призначені для використання довільно великого та настроюваного об’єму пам’яті порівняно з PBKDF2 і BCRYPT,

роблячи розмір і вартість апаратної реалізації геш-функції набагато дорожчими. Це у свою чергу впливає на вартість криптоаналізу з боку зловмисника.

Таблиця 2.2 ілюструє програмну реалізацію того чи іншого алгоритму гешування.

Таблиця 2.2

Програмна реалізація PBKDF2, BCRYPT, SCRYPT та Argon

Алгоритм гешування	Пароль (байти)	Сіль (байти)	Вивід (байти)	t_cost	m_cost	ROM	RAM	CPU
PBKDF2	24	8	64	1000	0	30	0	0,002024
PBKDF2	24	8	64	2048	0	30	0	0,004150
PBKDF2	24	8	64	4096	0	30	0	0,008141
PBKDF2	24	8	64	10000	0	30	0	0,019386
PBKDF2	24	8	64	1000000	0	30	0	1,908592
PBKDF2	24	8	64	16777216	0	30	0	32,969576
BCRYPT	12	16	54	12	0	27	492	2,668653
SCRYPT	8	32	64	5	0	182	450656	2,837654
Argon	32	32	32	3	0	82	192	0,008917

Подібно до BCRYPT, і SCRYPT, і Argon2 використовують ітерації в певних частинах алгоритму. SCRYPT був одним із перших запропонованих MHF [52], а в 2016 році алгоритм SCRYPT був опублікований IETF як стандарт (RFC 7914) [53].

Щодо точної кількості ітерацій для вищевказаних геш-функцій, рекомендації NIST радять використовувати PBKDF2 з мінімум 10000 ітерацій, тоді як автор SCRYPT рекомендує 16384 ітерації [52]. З іншого боку, немає офіційних рекомендацій для BCRYPT і Argon2. Однак мова програмування PHP за замовчуванням використовує BCRYPT з 1024 ітераціями [54]. PBKDF2 дозволяє компактну реалізацію, що робить цю геш-функцію придатною для обмежених можливостей [55].

Геш-функції MD5, SHA1, SHA256, SHA512 можуть виконуватися паралельно на багатопроцесорних системах, що значно підвищує ефективність атак підбору пароля. Ці алгоритми гешування досить поширені й досі використовуються

сьогодні, але необхідно уникати їх уникати, оскільки ці технології застаріли та їм на заміни вже прийшли інші [20].

Таблиця 2.3 демонструє обчислення GPU залежно від обраного алгоритму гешування [56].

Таблиця 2.3

Вимірювання GPU

Алгоритм гешування	**Гешів на секунду (H/s)**	**Модель GPU**
SHA1	794,600,000.00	ATI HD 6870
HMAC-SHA1	395,210,000.00	ATI HD 6870
SHA2	290,000,000.00	NVIDIA GRID K520
SHA3	113,116,250.00	NVIDIA GeForce GTX580
PBKDF2-SHA1	424,780.00	ATI HD 6870

продовження табл.2.3

PBKDF2-SHA2	219,480.00	NVIDIA Tesla C2070
Bcrypt	2868.00	NVIDIA GeForce GTX480
Scrypt	42,650.00	NVIDIA GeForce GTX480
Argon2	2.64	AMD Radeon HD 7900

Атаки на прості алгоритми гешування, що використовують тільки SHA1, SHA2, SHA3 або HMAC, можуть досягти 794–113 MH/s. Стандартизована схема PBKDF2 підвищує безпеку до 424–219 KH/s, Bcrypt знижує швидкість атаки до 2,8 KH/s. Коли використовується Scrypt із достатнім споживанням пам’яті, паралельну атаку можна обмежити до 0,37 H/s. Argon2 може досягти до 0,04 H/s.

Таким чином, PBKDF2 є стандартизованим алгоритмом гешування, який підходить для пристроїв з обмеженими обчислювальними можливостями, оскільки він має легшу реалізацію. Проте PBKDF2 передбачає використання солі в алгоритмі, що значно підвищує безпеку зберігання паролів та при цьому швидко обчислюється.

2.3 Переваги гешування паролів над шифруванням

І гешування, і шифрування забезпечують захист конфіденційних даних. Однак майже за всіх обставин паролі слід гешувати, а шифрувати.

Гешування є односторонньою функцією, таким чином неможливо «розшифрувати» геш і отримати початкове значення відкритого тексту. Тому гешування підходить зберігання паролів, адже навіть за умови, що зловмисник може отримати гешований пароль, він не може ввести його в поле пароля програми або застосунку та увійти як жертва. Шифрування ж навпаки є двосторонньою функцією. Це означає, що початковий відкритий текст можна отримати. Шифрування підходить для зберігання таких даних, як адреса користувача, оскільки ці дані можуть відображатися відкритим текстом у профілі користувача.

У контексті зберігання паролів шифрування слід використовувати лише в крайніх випадках, коли необхідно отримати оригінальний відкритий пароль. Це може знадобитися, якщо програмі потрібно використовувати пароль для автентифікації в іншій системі, яка не підтримує сучасний спосіб програмного надання доступу, наприклад OpenID Connect. Якщо можливо, слід використовувати альтернативну архітектуру, щоб уникнути необхідності зберігати паролі в зашифрованому
вигляді [27].

Висновки до розділу 2

За надійного захисту інформації користувачів необхідно застосовувати криптографічні методи, що забезпечують безпеку даних та ускладнюють для зловмисників задачу отримання несанкціонованого доступу до облікових записів. Для зберігання паролів у базі даних оптимальним є використання криптографічних геш-функцій.

Гешування – це процес виконання одностороннього перетворення рядка в інший детермінований новий рядок символів. Саме така властивість геш-функцій як

односторонність та детермінованість роблять їх придатними до забезпечення надійного зберігання паролів у базі даних.

Перша функція гешування була створена більше тридцяти років тому. Критичний аналіз та постійні спроби знайти вразливість призвели до того, що деяким застарілим геш-функціям приходили на зміну інші – більш надійні. Наразі необхідно уникати використання таких функцій як MD5 і SHA1, оскільки доведено, що ці ті інші геші з цього сімейства не відповідають основній умові надійності – вони не стійкі до колізій. Існує багато прикладів гешів, які підходять для використання під час зберігання паролів, наприклад, PBKDF2, Argon2, SCRYPT або BCRYPT. PBKDF2 передбачає використання солі в алгоритмі, таким чином зникає потреба створювати додатковий алгоритм, який забезпечував би створення унікальної солі. Даний алгоритм гешування підходить для реалізації на апаратурі з обмеженими обчислювальними можливостями.

Для забезпечення надійності збереження парольних даних не варто використовувати шифрування, оскільки воно поступається гешуванню у тому, що не використовує односторонні функції у своїх алгоритмах, які гарантують незворотність операції, тобто за наявності гешу неможливо відтворити початковий відкритий текст.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ МЕХАНІЗМУ БЕЗПЕЧНОГО ЗБЕРІГАННЯ ПАРОЛІВ

3.1 Формування основних вимог до механізму безпечного зберігання паролів

Важливо зберігати паролі таким чином, щоб зловмисник не зміг їх отримати, навіть за умови, якщо база даних компрометована. Сучасні мови програмування і фреймворки містять вбудований функціонал для безпечного зберігання паролів.

Перш за все, парольні дані користувачів мають зберігатися за допомогою криптографічної геш-функції у базі даних. Зберігання паролів у вигляді відкритого тексту не є безпечним методом. Також не варто шифрувати паролі, оскільки використовувані алгоритми є зворотними [27]. Проте за умови використання геш-функцій паролі не будуть відомі навіть адміністратору бази даних, який має безпосередній доступ до її вмісту. Якщо зловмисник отримає доступ до гешів, він завжди може спробувати віднайти паролі, тому основна задача розробників – уповільнити роботу зловмисника, наскільки це можливо. Для цього необхідно обирати такі алгоритми гешування, які будуть повільними та максимально ресурсомісткими. Варто знайти «золоту середину», адже алгоритмом має бути зручно користуватись самим розробникам системи.

Краще не використовувати такі функції гешування як MD5 і SHA1, оскільки вони були розроблені, щоб навпаки бути швидкими, та, як виявилось, не стійкі до колізій та не передбачають використання солі. Для збереження паролів можна скористатися такими функціями гешування: PBKDF2, Argon2, SCRYPT або BCRYPT– створення й застосування солі у них одразу передбачено [20].

Також можна використовувати додатковий параметр перець, щоб забезпечити більш глибокий захист.

Отже, основні вимоги до механізму безпечного зберігання паролів у базі даних можна сформулювати так:

- не зберігати паролі у вигляді звичайного тексту;
- не використовувати алгоритми шифрування;
- необхідно використовувати криптографічні геш-функції, такі як PBKDF2, Argon2, SCRYPT або BCRYPT, в яких також передбачена унікальна сіль;
- не використовувати застарілі алгоритми гешування, наприклад, MD5 і SHA1;
- використовувати перець для більш глибокого захисту;
- попіклуватися про достатню кількість ітерацій;
- надавати користувачам рекомендації зі створення надійного пароля.

3.2 Обґрунтування вибору програмних засобів реалізації механізму

Для реалізації механізму безпечного зберігання паролів використана мова програмування Javascript.

Javascript – об’єктно-орієнтована мова програмування високого рівня, що була представлена у грудні 1995 року компанією Netscape Communications Corporation [57]. Дана мова програмування підтримує імперативні та функціональні стилі, і зазвичай використовуються як вбудована мова для програмового доступу до об’єктів додатків і найбільше застосовується для надання веб-сторінкам у браузері інтерактивного вигляду. Скриптова мова Javascript має динамічну типізацію, прототипне наслідування та функції як об’єкти першого класу й відповідає стандарту ECMAScript [58].

Використання Javascript вийшло за межі лише одного веб-браузера. Механізми даної мови вбудовані тепер в інші програмні системи для розгортання веб-сайтів на сервері, а також додатків.

Додатки, написані на мові Javascript, можуть виконуватись на серверах за допомогою таких платформ як Jaxer, persevere-framework, Helma, v8cgi, gopherjs та Node.js. Буде використовуватись Node.js, оскільки дана платформа є найпопулярнішою.

3.3 Опис програмної реалізації механізму

Основна задача полягає у створенні мобільний додаток, що буде ілюструвати процес ідентифікації, автентифікації та авторизації користувача в системі. Це буде єдиний функціонал даного додатку, оскільки необхідно безпосередньо продемонструвати механізм безпечного зберігання паролів. Потрібно підключити базу даних, а також забезпечити зберігання парольних даних у гешованому вигляді за допомогою актуальних алгоритмів, з використанням унікальної солі та з достатньою кількістю ітерацій. З точки зору користувача даний додаток повинен надавати можливість введення нікнейму та паролю й показувати результат проходження авторизації, зважаючи на те, чи є вона успішною чи ні.

В перспективі цю частину можна імплементувати у повноцінний додаток, оскільки реєстрація користувача присутня зараз, певно, у всіх застосунках, тож вкрай важливо правильно налаштувати механізм безпеки даних.

Отже, для виконання практичної частини дипломної роботи використано мову програмування Javascript, середу виконання Node.js, базу даних PostgreSQL та Docker.

Крок 1. Спочатку необхідно створити API – програмовий інтерфейс додатку, тобто механізми, які дозволяють програмовим компонентам взаємодіяти один з одним, користуючись набором визначень та протоколів. Прикладом може бути система програмного забезпечення метеослужби, що містить щоденні дані про погоду. Додаток цієї метеослужби у смартфоні «спілкується» з цією системою через API та показує щоденні оновлення про погоду.

Архітектуру API відносять до клієнт-серверної. Тобто той додаток (у цьому контексті – будь-яке програмне забезпечення, що виконує певну функцію), що надсилає запит, називається клієнтом, а той, що надає відповідь, – сервером. У даному випадку, база даних – це сервер, а мобільний додаток у смартфоні – це клієнт.

Для створення файлу конфігурації package.json необхідно створити папку api, перейти у неї та запустити команду npm init (рис. 3.1)

```
stepanenkoyuliia@MacBook-Pro api % npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (api)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /Users/stepanenkoyuliia/Desktop/juli/api/package.json:

{
  "name": "api",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}


Is this OK? (yes)
```

Рисунок 3.1 – Запуск команди npm init для створення файлу конфігурації

Після чого API необхідно відкрити у VS Code. Тепер можна побачити файл index.js, з цього файлу буде запускатися API.

Крок 2. Отримання та обробка запитів вимагає використання фреймворку express, встановити його можна за допомогою команди npm install express (рис. 3.2).

```
stepanenkoyuliia@MacBook-Pro api % npm install express

added 57 packages, and audited 58 packages in 534ms

7 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
stepanenkoyuliia@MacBook-Pro api %
```

Рисунок 3.2 – Встановлення фреймворку express

Крок 3. Можна створити express додаток, який буде поки-що мати одну кінцеву точку і буде відправляти «Hello World» (рис. 3.3).

```
import express from 'express';

const app = express();
const port = 3000

app.get('/test', (req, res) => {
  res.send('Hello World!')
})

app.listen(port)

```

Рисунок 3.3 – Створення базового express додатку

Даний код демонструє імпорт пакету express. const app = express(), виклик функції express, яка створює новий додаток, після чого присвоює результат константі app. const port = 3000 означає, що додаток запускає сервер та слухає з'єднання на порту 3000. app.get – це функція, яка буде прослуховувати на GET запит (запит на отримання даних). /test – це URL, до якого буди виконуватись GET запит. (req, res) => {} – це функція, яка буде викликатися при зверненні до кінцевої точки. res.send буде відправляти відповідь користувачу, який зробив запит. Відповідно, коли користувач виконує GET запит до localhost:3000/test буде викликатися функція й відображуватиметься напис «Hello World!». Останнім кроком буде налаштування можливості запускати сервер. Для цього використовується метод listen та вказується порт [59].

Додаток можна запустити, скориставшись командою node index.js. Для перегляду необхідно відкрити сторінку http://localhost:3000/test та перевіряємо результат (рис. 3.4).

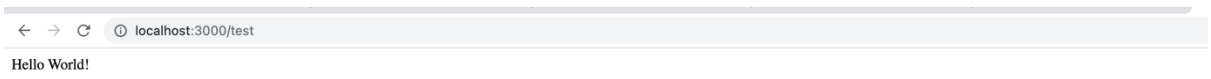

Рисунок 3.4 – Вивід «Hello World!»

Також можна скористатися Postman для тестування запитів (рис. 3.5).

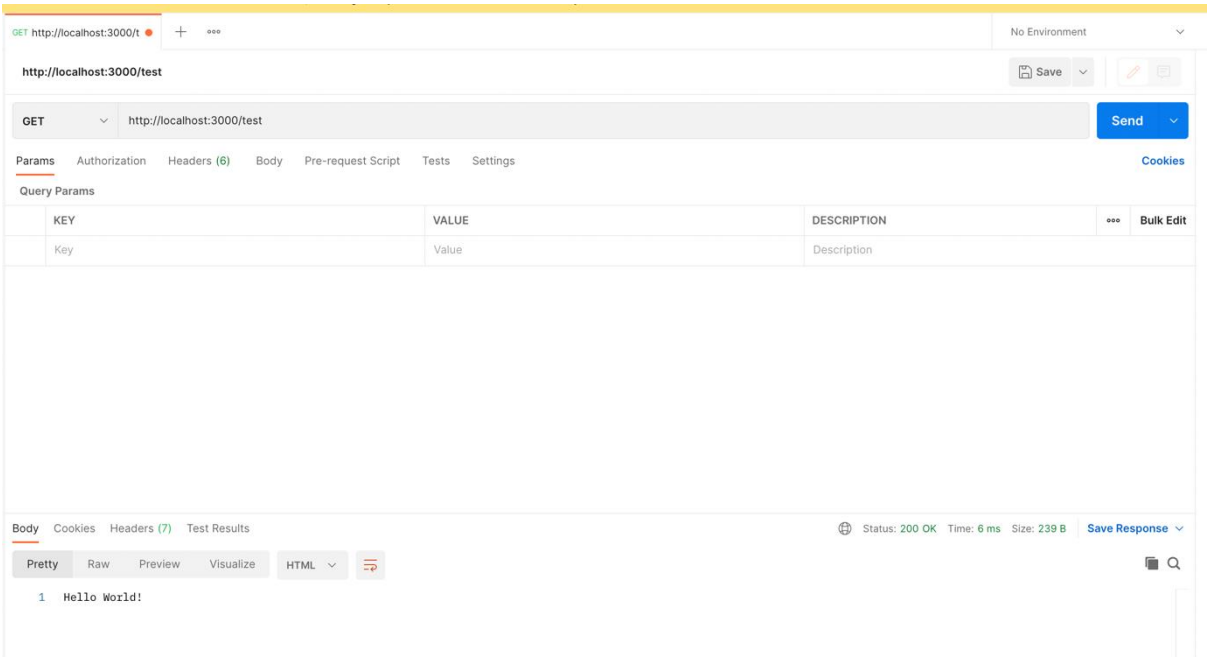

Рисунок 3.5 – Перевірка результату в Postman

Крок 4. Коректне функціонування додатку вимагає створення бази даних, для цього використовується Docker. В папці api створені Dockerfile і docker-compose.yml. У файлі Dockerfile описуватимуться всі дії, які потрібно виконати, щоб запустити API (рис. 3.6).

Рисунок 3.6 – Код файлу Dockerfile

У файлі Dockerfile реалізовані наступні команди:

- прописано, яку версію NodeJS використано;
- створено робочу папку;
- скопійовано усі файли в робочу папку;
- встановлено залежності;
- дозволено прослуховувати порт 3000;
- прописано запуск додатку.

У docker-compose.yml описана серверна інфраструктура (рис. 3.7).

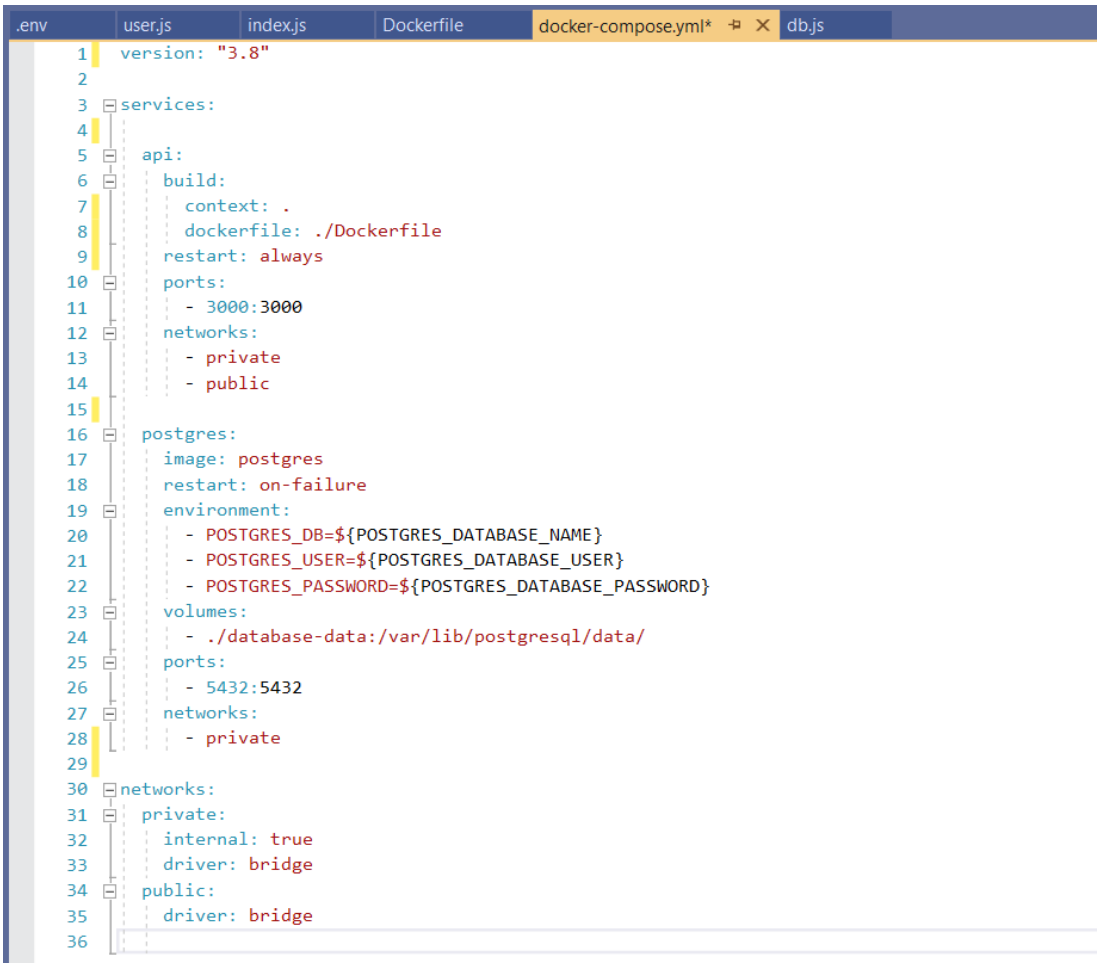

Рисунок 3.7 – Код файлу docker-compose.yml

У даному файлі необхідно прописати конфігурацію двох сервісів – додатку та бази даних. Відповідно до коду:

- додаток має запускатись з цієї ж директорії;
- є посилання на файл Dockerfile, звідти необхідно взяти налаштування;

- встановлено, що додаток буде перезавантажуватись завжди, не зважаючи на те, чи є помилки, чи ні;
- вказано внутрішній і зовнішній порти;
- вказано налаштування мережі – локальна та з виходом в Інтернет.

Відповідно до коду, що описує другий сервіс, використовується база даних postgres, що перезавантажується, якщо є якісь помилки, із файлу .env беруться значення змінних оточення (рис. 3.8). Вказано шлях до папки хоста, зовнішній та внутрішній порти та налаштування мережі – локальна. Це означає, що доступ до бази даних можна отримати тільки через локальну мережу.

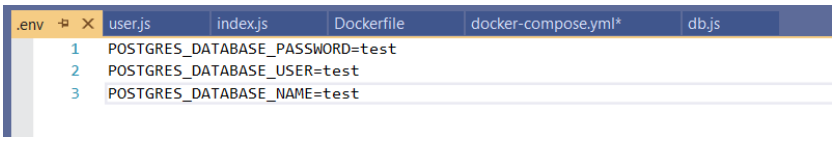

Рисунок 3.8 – Вміст файлу .env

У налаштуваннях мережі internal: true означає обмеження зовнішнього доступу до приватної мережі. Тут же вказано драйвер для управління мережею, який за замовчуванням bridge [60].

Крок 5. Налаштування файлу db.js (рис. 3.9).

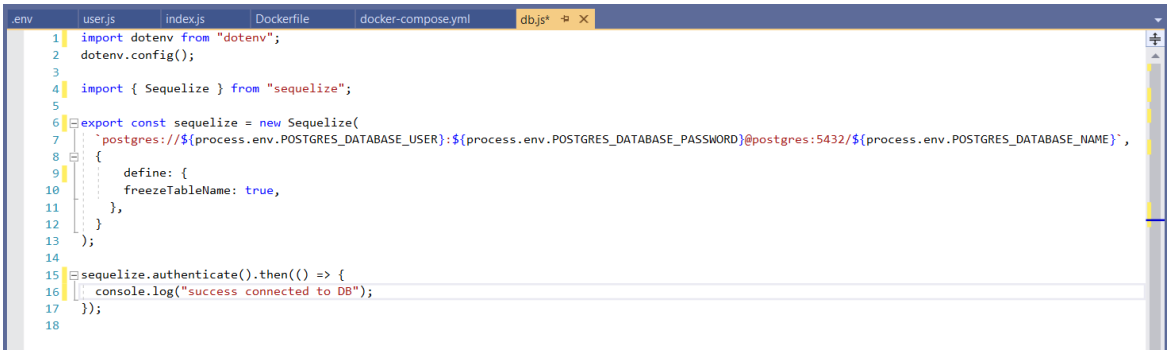

Рисунок 3.9 – Код файлу db.js

Тут необхідно пропарсити файл .env, тобто розібрати дані на частини для їх збереження у базі даних або для виконання запиту до бази даних, базуючись на окремих елементах даних [61]. Dotenv — це модуль нульової залежності, який завантажує змінні оточення з файлу .env [62]. Імпортовано систему управління

базою даних sequelize та побудовано URI для підключення цієї системи управління безпосередньо до бази даних [63]. За допомогою sequelize.authenticate() перевіряється чи можливо підключитися до бази даних. Якщо підключення буде успішне, то викликається функція .then() та буде виведено повідомлення в консоль «success connected to DB».

Крок 6. Налаштування файлу index.js, який умовно можна поділити на дві частини – реєстрацію (рис. 3.10) та логін.

```
import express from "express";
import { User } from "./user.js";
import { pbkdf2Sync, randomBytes } from "crypto";
import fs from "fs";

const app = express();
const port = 3000;

app.use(express.json());
app.use(express.urlencoded({ extended: false }));

const accessLogStream = fs.createWriteStream("./access.log", {
  flags: "a",
});

app.listen(port);

app.post("/signup", async (req, res) => {
  try {
    if (req.body.password.length < 8 || req.body.username.length < 3) {
      return res.status(400).send("error");
    }

    const salt = randomBytes(128);
    const passwordHash = pbkdf2Sync(req.body.password, salt, 5000, 128, "sha1");

    const user = await User.create({
      username: req.body.username,
      passwordHash,
      salt,
    });

    accessLogStream.write(
      new Date() + "    -       create " + user.id + " " + user.username + "\n"
    );
    return res.status(200).send("success");
  } catch {
    return res.status(500).send("error");
  }
});
```

Рисунок 3.10 – Налаштування реєстрації у файлі index.js

Для налаштування процесу реєстрації:

- імпортовано пакет express, модуль crypto для можливості застосування гешування та модуль fs для роботи з файловою системою, оскільки необхідно буде створити файл для логування;
- запущена функція, імпортована за допомогою const app = express();
- вказано порт, на який будуть отримуватись запити;
- пропарсений body запиту, де express. json() – це метод, вбудований у express, щоб розпізнати вхідний об’єкт запиту як об’єкт JSON; цей метод входить до

проміжного програмного забезпечення й викликається між опрацюванням запиту та відправленням відповіді в додатку; express.urlencoded() – це вбудований у вираз метод для розпізнавання вхідного об'єкта запиту як рядка або масиву й також входить у проміжне програмне забезпечення;

- створений файл для запису логів;
- запущений додаток за допомогою app.listen(port);
- використаний тип запиту POST для створення нового ресурсу;
- створена функція app.post, що буде обробляти POST запити, де /signup – це URL, який буде прослуховуватись, async (req, res) => {} – асинхронна функція, яка викликається з двома параметрами request і response і буде виконана, коли прийде запит;
- перевірено, якщо довжина пароля менше 8 символів або довжина імені користувача менше 3 символів, то повертається відповідь за статусом 400 та повідомленням «error»;
- створена функція формування унікальної солі довжиною 128 біт;
- створена функція гешування const passwordHash = pbkdf2Sync(req.body.password, salt, 5000, 128, "sha1"), де pbkdf2 – це алгоритм гешування, req.body.password – пароль користувача, salt – унікальна сіль, яку ми згенерували перед цим, 5000 – кількість ітерацій, 128 – довжина солі та sha1 – це внутрішній алгоритм гешування;
- створюється новий користувач та зберігаються його ім’я, геш пароля та сіль;
- додається у файл логування запис про створення нового користувача із вказанням дати, події, ID користувача та юзернейму;
- якщо під час реєстрації сталася якась непередбачувана помилка – вона повертається зі статусом 500 та «error».

Таким чином створюється новий користувач та додаються його облікові дані до бази даних. Паролі, як видно, ніде не фігурують, база даних зберігає лише геші.

Під час входу користувача у систему (рис. 3.11) також перевіряється довжина пароля та імені користувача. Якщо вони відповідають вимогам – в базі даних

відбувається пошук користувача з таким нікнеймом і якщо його знайдено – відбувається такий самий процес гешування введеного користувачем пароля та співставлення з тим, що збережений у базі даних. Якщо геші паролів співпадають, то у файл логів додається запис про логін користувача із вказанням дати, ID та юзернейму.

```
app.post("/login", async (req, res) => {
  try {
    if (req.body.password.length < 8 || req.body.username.length < 3) {
      return res.status(400).send("error");
    }

    const user = await User.findOne({
      where: {
        username: req.body.username,
      },
    });

    if (user) {
      const passwordHashToCheck = pbkdf2Sync(
        req.body.password,
        user.salt,
        5000,
        128,
        "sha1"
      );

      if (user.passwordHash.equals(Buffer.from(passwordHashToCheck))) {
        accessLogStream.write(
          new Date() +
            "    -       login " +
            user.id +
            " " +
            user.username +
            "\n"
        );
        return res.status(200).send("success");
      } else {
        return res.status(422).send("error");
      }
    } else {
      return res.status(404).send("error");
    }
  } catch {
    return res.status(500).send("error");
  }
});

```

Рисунок 3.11 – Налаштування логіну у файлі index.js

Крок 7. У файлі user.js необхідно визначити модель, що представляє таблицю в базі даних (рис. 3.12).

```
import { DataTypes } from "sequelize";
import { sequelize } from "./db.js";

export const User = sequelize.define(
  "User",
  {
    id: {
        primaryKey: true,
        type: DataTypes.UUID,
       defaultValue: DataTypes.UUIDV4,
    },
    username: {
      type: DataTypes.STRING,
      unique: true,
    },
    passwordHash: {
      type: DataTypes.STRING({ binary: true }),
    },
    salt: {
      type: DataTypes.STRING({ binary: true }),
    },
  },
  {
    sequelize,
  }
);

await User.sync({ force: true });
```

Рисунок 3.12 – Модель представлення таблиці в базі даних

Стовпці таблиці визначаються об’єктом, що задається в якості другого аргументу. Кожний ключ представляє стовпець. Це робиться за допомогою виклику sequelize.define(modelName, attributes, options), де modelName – це назва моделі, attributes – визначають стовпці таблиці, options поєднуються з параметрами define за замовчуванням та не є обов’язковими [64]. Відповідно до коду, назва моделі – «User», стовпцями таблиці є id, який є унікальним ідентифікатором кожного запису в таблиці, username строкового типу, passwordHash та salt (рис. 3.13).

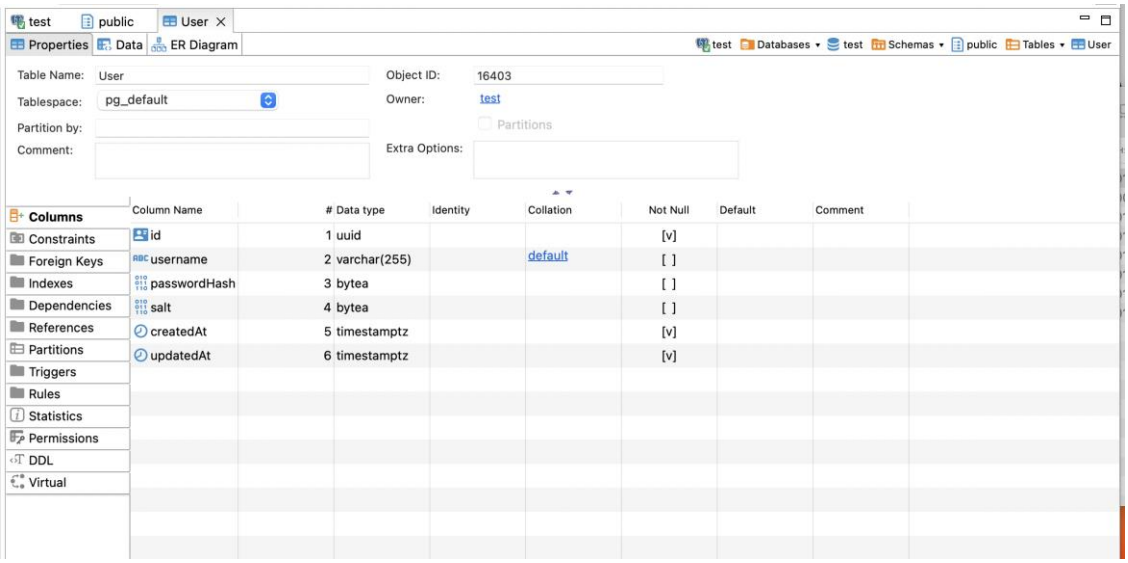

Рисунок 3.13 – Демонстрація наявних у таблиці бази даних стовпців

Крок 8. У файлі App.tsx необхідно визначити «зовнішній вигляд» застосунку, тобто це розробка інтерфейсу, який буде бачити користувач і з яким він буде безпосередньо взаємодіяти (Додаток А).

Тут вказуються стилі, які будуть застосовуватись при зміні певного значення. Наприклад, при зміні на екран логіну елементи будуть розміщуватись по центру, буде застосований градієнт та буде напис «Успішно!!!» та кнопка виходу (рис. 3.14).

```
if (isLogin) {
  return (
    <View
      style={{
        flex: 1,
        backgroundColor: "#fff",
        alignItems: "center",
        justifyContent: "center",
      }}
    >
      <LinearGradient
        colors={["rgba(69, 131, 255, 0.8)", "rgba(255, 221, 48, 0.8)"]}
        style={{
          position: "absolute",
          left: 0,
          right: 0,
          top: 0,
          bottom: 0,
        }}
      />
      <Text style={{ fontSize: 25, marginBottom: 30 }}>Успішно!!!</Text>
      <Button
        title="Вийти"
        onPress={() => {
          setIsLogin(false);
          setError("");
          setUsername("");
          setPassword("");
        }}
      />
    </View>
  );
```

Рисунок 3.14 – Розробка екрану логіну

Головний екран містить напис «Додаток», поля для вводу даних (рис. 3.15).

```
      <Text style={{ fontSize: 25, marginBottom: 30 }}>Додаток</Text>
      <TextInput
        style={{
          width: "90%",
          borderWidth: 2,
          borderColor: "grey",
          borderRadius: 5,
          height: 50,
          marginTop: 20,
          paddingHorizontal: 5,
          backgroundColor: "rgba(255, 255, 255, 0.8)",
        }}
        placeholder="Username"
        placeholderTextColor={"grey"}
        value={username}
        onChangeText={setUsername}
        autoCapitalize="none"
      />
```

Рисунок 3.15 – Стилі головного екрану додатку

Поля вводу даних (імені користувача та пароля) мають однакові налаштування та займають 90% ширини екрану, товщина рамки 2 px, її колір – сірий, скруглення – 5 px, висота блоку – 50 px, відступ зверху 20 px, відстань до тексту всередині 5 px (рис. 3.16).

```
<TextInput
  style={{
    width: "90%",
    borderWidth: 2,
    borderColor: "grey",
    borderRadius: 5,
    height: 50,
    marginTop: 20,
    marginBottom: 10,
    paddingHorizontal: 5,
    backgroundColor: "rgba(255, 255, 255, 0.8)",
  }}
  placeholder="Password"
  placeholderTextColor={"grey"}
  autoCapitalize="none"
  secureTextEntry={true}
  value={password}
  onChangeText={setPassword}
/>
```

Рисунок 3.16 – Стилі головного екрану додатку (продовження)

3.4 Використання програмної реалізації механізму

Головний екран додатку зображений на рисунку 3.17. Для того, щоб запустити додаток в симуляторі, необхідно прописати команду expo init [64].

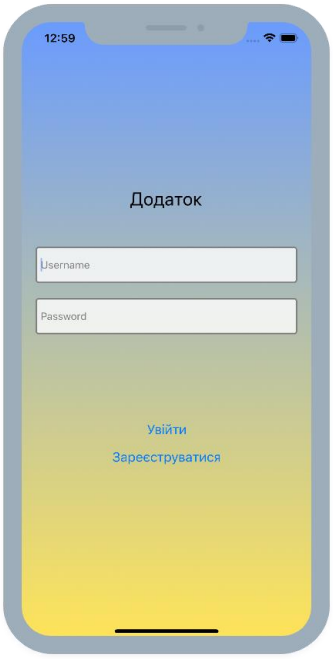

Рисунок 3.17 – Головний екран додатку

Наступний крок – реєстрація користувача. Для цього необхідно ввести нікнейм та пароль у відповідні поля на екрані додатка. Якщо довжина імені користувача або паролю менша за необхідну – виводиться повідомлення про помилку, тож необхідно повторно ввести дані, але з урахуванням вимог (рис. 3.18).

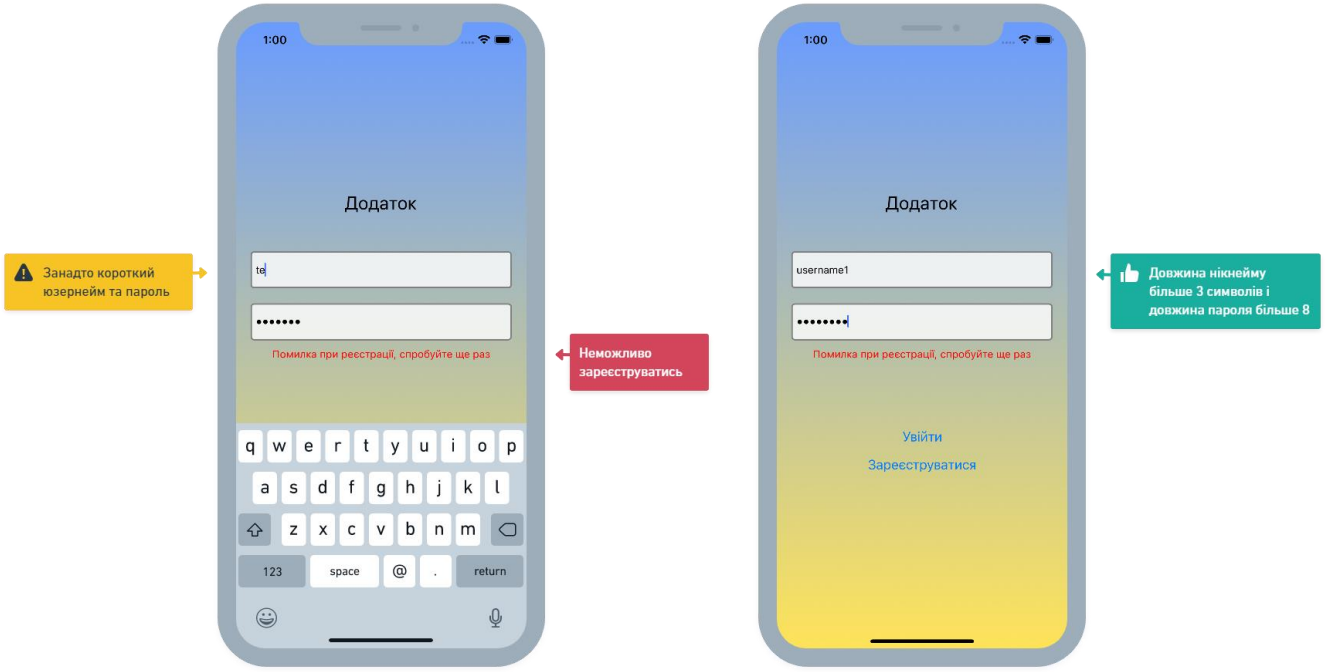

Рисунок 3.18 – Спроба реєстрації

Якщо користувач вже зареєстрований у системі – він може залогінитись. Якщо введене ім’я користувача не знайдене у базі даних або геш введеного пароля не співпадає з гешем пароля, що зберігається в базі даних, – на екран виводиться нейтральне повідомлення про помилку, оскільки якщо це зловмисник намагається зайти в чужий обліковий запис, необхідно максимально ускладнити йому задачу та не уточнювати, в яких саме даних він помилився, коли вводив їх. Якщо введені дані правильні – користувач потрапляє на новий екран з повідомленням про успішний вхід у систему та з кнопкою виходу звідти (рис. 3.19).

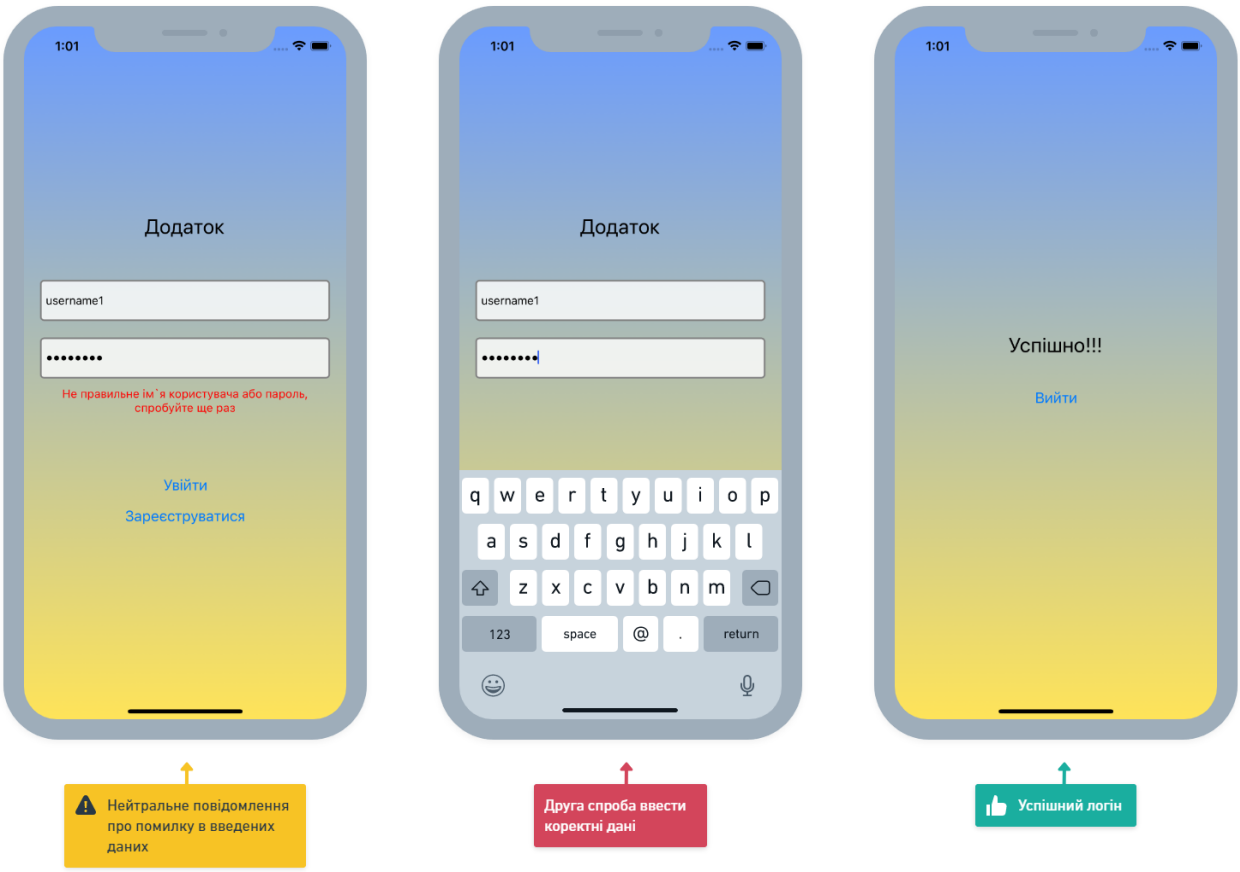

Рисунок 3.19 – Спроба ввійти в систему

Після реєстрації користувачів у додатку – їх дані зберігаються у базу даних, при цьому створюється новий рядок в таблиці (рис. 3.20).

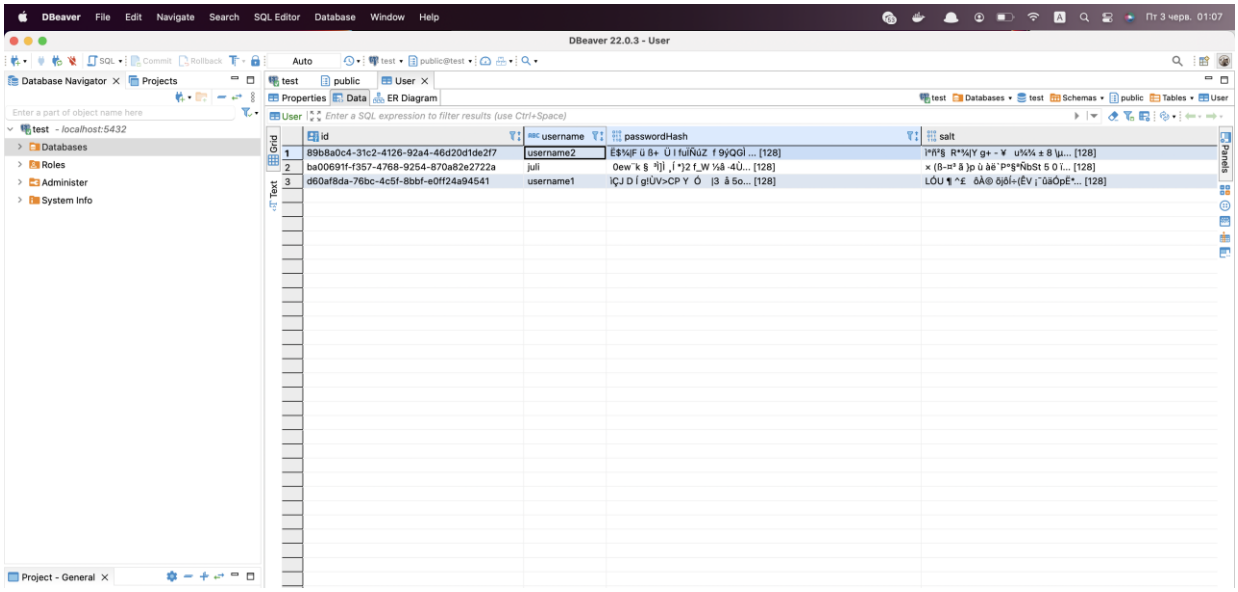

Рисунок 3.20 – Демонстрація бази даних додатку

Отже, паролі у вигляді відкритого тексту в базі даних не зберігаються. Замість цього зберігаються геші цих паролів, що підвищує рівень безпеки даних користувачів, оскільки навіть адміністратор бази даних не може переглянути паролі.

Висновки до розділу 3

У даному розділі було розроблено мобільний додаток, що наочно продемонстрував процес реєстрації нового користувача та подальшу його авторизацію в системі після ідентифікації (надання користувачем його юзернейму) та автентифікації (надання підтвердження, що він той, ким називається, тобто введення парольних даних).

Розроблено backend (серверну частину) та frontend (користувацький інтерфейс) додатку. Підключено базу даних та забезпечене безпечне зберігання паролів за допомогою криптографічної геш-функції PBKDF2.

ВИСНОВКИ

У дипломній роботі вирішена актуальна проблема безпечного зберігання паролів. У ході вирішення поставлених завдань були отримані наступні теоретичні та практичні результати:

1. Проаналізовано комплексний підхід до забезпечення безпеки даних користувачів, що проявляється передусім у забезпеченні безпеки бази даних та серверного обладнання.

2. Проаналізовано різні концепції зберігання паролів, обґрунтовано, що ефективною є концепція із застосуванням криптографічної геш-функції і солі.

3. Проаналізовано розповсюджені геш-функції, визначено їх особливості та відмінності. Визначено функцію гешування, що відповідає вимогам надійного зберігання паролів.

4. Сформульовано основні вимоги до механізму безпечного зберігання паролів: не зберігати паролі у вигляді звичайного тексту, не використовувати застарілі алгоритми гешування, забезпечувати використання солі (за необхідності й перцю) та надавати користувачам рекомендації зі створення надійних паролів.

5. Виконано практичну реалізацію механізму безпечного зберігання паролів через розробку мобільного застосунку, який забезпечує ідентифікацію, автентифікацію та авторизацію користувача, збереження його парольних даних у базі даних за допомогою криптографічної геш-функції та солі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wakefield, J. eBay faces investigations over massive data breach [Електронний ресурс] / Jane Wakefield. – 2014. –
Режим доступу: https://www.bbc.com/news/technology-27539799.

2. Database Security [Електронний ресурс]. –
Режим доступу: https://www.ibm.com/cloud/learn/database-security.

3. Porter, H. Nine sacked for breaching core ID card database [Електронний ресурс] / Henry Porter. – 2009. –
Режим доступу: https://www.theguardian.com/commentisfree/henryporter/2009/aug/10/id-card-database-breach.

4. Ricardo, C. Databases Illuminated [Электронный ресурс] / Catherine M. Ricardo, Susan Urban. – 2017. –
Режим доступу: http://samples.jbpub.com/9781284056945/DBICHAP8.pdf.

5. E-Government Act of 2002 [Електронний ресурс]: Public Law 107-347 Approved December 17, 2002. –
Режим доступу: https://www.govinfo.gov/content/pkg/STATUTE-116/pdf/STATUTE-116-Pg2899.pdf.

6. General Data Protection Regulation [Електронний ресурс]: Directive 95/46/EC of 27 April 2016. – Режим доступу: https://eur-lex.europa.eu/eli/reg/2016/679/oj.

7. HIPAA FOR DUMMIES [Електронний ресурс]. – Режим доступу: https://www.hipaaguide.net/hipaa-for-dummies/.

8. SARBANES-OXLEY ACT OF 2002 [Електронний ресурс]: Public Law 107–204, Approved July 30, 2002. –
Режим доступу: https://www.govinfo.gov/content/pkg/COMPS-1883/pdf/COMPS-1883.pdf.

9. GRAMM–LEACH–BLILEY ACT [Електронний ресурс]: Public Law 106–102, Approved November 12, 1999. –

Режим доступу: https://www.govinfo.gov/content/pkg/PLAW-106publ102/pdf/PLAW-106publ102.pdf.

10. Заплотинський Б. А. Основи інформаційної безпеки : конспект лекцій / Б. А. Заплотинський. – М.: КІІВіП НУ «ОЮА», кафедра інформаційно-аналітичної та інноваційної діяльності, 2017. – 128 с.

11. SQL Injection Prevention Cheat Sheet [Електронний ресурс]. – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html.

12. Guide to General Server Security [Електронний ресурс]: NIST Special Publication 800-123 of July 2008. – Режим доступу: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-123.pdf.

13. Identify IBM i host servers and associated programs [Електронний ресурс]. – Режим доступу: https://www.ibm.com/docs/en/i/7.4?topic=administration-identify-i-host-servers-associated-programs.

14. Design Principles [Електронний ресурс]. – Режим доступу: https://www.cs.clemson.edu/course/cpsc420/material/Design%20Principles/Design%20Principles.pdf.

15. Curtin, M. Developing Trust: Online Privacy and Security / Matt Curtin // Proceedings of the IEEE. – 2001. – Vol. 63, pages 1278– 1308.

16. Український тлумачний словник [Електронний ресурс]. – Режим доступу:https://ukrainian_explanatory.academic.ru/23170/%D0%B2%D1%96%D0%B4%D0%BC%D0%BE%D0%B2%D0%BE%D1%81%D1%82%D1%96%D0%B9%D0%BA%D1%96%D1%81%D1%82%D1%8C.

17. What Is Server Security - and Why Should You Care? [Електронний ресурс]. – Режим доступу: https://www.avast.com/c-b-what-is-server-security.

18. The 2019 State of Password and Authentication Security Behaviors Report [Електронний ресурс]. – Режим доступу: https://resources.yubico.com/53ZDUYE6/at/q3tmql-974v8g-73e8p5/YubicoPonemon_2019_State_of_Password_and_Authentication_Security_Behaviors_Report.pdf?format=pdf.

19. Yubico and Ponemon Institute Release the 2020 State of Password and Authentication Security Behaviors Report [Електронний ресурс]. – Режим доступу: https://www.yubico.com/press-releases/yubico-and-ponemon-institute-release-the-2020-state-of-password-and-authentication-security-behaviors-report/.

20. Modern password security for system designers [Електронний ресурс]. – Режим доступу: https://cloud.google.com/solutions/modern-password-security-for-system-designers.pdf.

21. The Password Meter [Електронний ресурс]. – Режим доступу: http://www.passwordmeter.com/.

22. Створення та використання надійних паролів [Електронний ресурс]. – Режим доступу: https://cutt.ly/eJSPz8s.

23. Create a strong password & a more secure account [Електронний ресурс]. – Режим доступу: https://support.google.com/accounts/answer/32040?hl=en#zippy.

24. Рекомендації ESET щодо створення надійного паролю [Електронний ресурс]. – Режим доступу: https://eset.ua/ua/news/view/561/recomendacii-Eset-po-parolu.

25. How can we help you? [Електронний ресурс]. – Режим доступу: https://support.google.com/accounts/?hl=en#topic&topic.

26. Practical Cryptography for Developers [Електронний ресурс]. – Режим доступу: https://cryptobook.nakov.com/.

27. Password Storage Cheat Sheet [Електронний ресурс]. – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#international-characters.

28. Passwords and hacking: the jargon of hashing, salting and SHA-2 explained [Електронний ресурс]. – Режим доступу: https://www.theguardian.com/technology/2016/dec/15/passwords-hacking-hashing-salting-sha-2#:~:text=Salting%20is%20simply%20the%20addition,same%20salt%20for%20every%20password.

29. Кисель В. А., Основы криптографии: учеб. Пособие / [В. А. Кисель, Н. В. Захарченко]. – Одесса, 1997. – 48 с.

30. Meaning of hash in English [Електронний ресурс]. – Режим доступу: https://dictionary.cambridge.org/dictionary/english/hash.

31. Diffie, W. New directions in cryptography. IEEE Transactions on Information Theory / W. Diffie, M. E. Hellman // IT-22(6) . – 1976. – 644–654.

32. Merkle, R. Secrecy, Authentication, and Public Key Systems. PhD thesis / R. C. Merkle // Stanford University, 1979.

33. Rabin, M. Digitalized signatures. In R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, editors, Foundations of Secure Computation / M. Rabin // Academic Press. – 1978. – 155–168.

34. Stepanenko, Y. Secure password storage with cryptographic hash function / Y. Stepanenko, V. Solodovnik. – 2021.

35. Корченко О. Г., Сіденко В. П., Дрейс Ю. О.. Прикладна криптологія: системи шифрування: підручник / О. Г. Корченко, В. П. Сіденко, Ю. О. Дрейс. – К.: ДУТ, 2014. – 448 с.

36. Antunes, L. One-Way Functions Using Algorithmic and Classical Information Theories [Electronic resource] / L. Antunes, A. Matos, A. Pinto, A. Souto // Theory Comput Syst. – 2012. – Access: https://www.researchgate.net/publication/230900687_One-Way_Functions_Using_Algorithmic_and_Classical_Information_Theories.

37. Naor, M. Universal One-Way Hash Functions and their Cryptographic Applications [Electronic resource] / M. Naor, M. Yung. – Access: https://www.wisdom.weizmann.ac.il/~naor/PAPERS/uowhf.pdf.

38. Goldreich, O. Foundations of Cryptography / O. Goldreich // Cambridge University Press, 2001.

39. Global Information Assurance Certification Paper [Електронний ресурс]. – Режим доступу: https://www.giac.org/paper/gsec/3294/study-hash-functions-cryptography/105433.

40. Contini, S. Universal A Critical Look at Cryptographic Hash Function Literature [Electronic resource] / S. Contini, R. Steinfeld, J. Pieprzyk. K. Matusiewicz. – Access: https://www.researchgate.net/publication/240768342_A_Critical_Look_at_Cryptographic_Hash_Function_Literature.

41. Hashing functions [Електронний ресурс]. – Режим доступу: https://www.ibm.com/docs/pt/psfa/7.2.1?topic=toolkit-hashing-functions.

42. Ntantogian, C. Evaluation of Password Hashing Schemes in Open Source Web Platforms [Electronic resource] / C. Ntantogian, S. Malliaros, C. Xenakis. – Access: https://www.cyberwatching.eu/sites/default/files/Passwords-sub-v26.pdf.

43. The MD5 Message-Digest Algorithm [Електронний ресурс]. – Режим доступу: https://www.ietf.org/rfc/rfc1321.txt.

44. Thomsen, S. Cryptographic Hash Functions [Electronic resource] / S. S. Thomsen. – Access: https://backend.orbit.dtu.dk/ws/portalfiles/portal/5025771/sst_thesis_v1.0.pdf.

45. Boer, B. Collisions for the Compression Function of MD5 [Electronic resource] / B. Den Boer, A. Bosselaers. – Access: https://www.esat.kuleuven.be/cosic/publications/article-143.pdf.

46. Announcing the first SHA1 collision [Електронний ресурс]. – Режим доступу: https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html.

47. Gradually sunsetting SHA-1 [Електронний ресурс]. – Режим доступу: https://security.googleblog.com/2014/09/gradually-sunsetting-sha-1.html.

48. National Institute of Standards and Technology / U.S. Department of Commerce. Federal Information Processing Standards Publication (FIPS PUB) 197. Advanced Encryption Standard (AES), 2001.

49. National Institute of Standards and Technology / U.S. Department of Commerce. Federal Information Processing Standards Publication (FIPS PUB) 180-2. Secure Hash Standard, 2002.

50. Speed Hashing [Електронний ресурс]. –
Режим доступу: https://blog.codinghorror.com/speed-hashing/.

51. Security releases issued [Електронний ресурс]. –
Режим доступу: https://www.djangoproject.com/weblog/2013/sep/15/security/.

52. Percival, C. Stronger Key Derivation via Sequential Memory-Hard Functions [Electronic resource] / C. Percival. – Access: https://www.tarsnap.com/scrypt/scrypt.pdf.

53. The scrypt Password-Based Key Derivation Function [Електронний ресурс]. –
Режим доступу: https://tools.ietf.org/html/rfc7914.

54. PHP password_hash [Електронний ресурс]. –
Режим доступу: http://php.net/manual/en/function.password-hash.php.

55. Hatzivasilis, G. Password Hashing Competition - Survey and Benchmark [Electronic resource] / G. Hatzivasilis, I. Papaefstathiou, C. Manifavas. – Access: https://eprint.iacr.org/2015/265.pdf.

56. Hatzivasilis, G. One-Way Functions Using Algorithmic and Classical Information Theories [Electronic resource] / G. Hatzivasilis. –
Access: https://www.researchgate.net/publication/317936505_Password-Hashing_Status

57. Netscape and sun announce javascript, the open, cross-platform object scripting language for enterprise networks and the Internet [Електронний ресурс]. –
Режим доступу: https://web.archive.org/web/20070916144913/https://wp.netscape.com/newsref/pr/newsrelease67.html.

58. ECMAScript 2021 Language Specification [Електронний ресурс]. –
Режим доступу: https://www.ecma-international.org/wp-content/uploads/ECMA-262_12th_edition_june_2021.pdf.

59. Hello world example [Електронний ресурс]. –
Режим доступу: https://expressjs.com/en/starter/hello-world.html.

60. Docker network create [Електронний ресурс]. –
Режим доступу: https://docs.docker.com/engine/reference/commandline/network_create/
.

61. Перекладаємо слово парсити [Електронний ресурс]. – Режим доступу: https://slovotvir.org.ua/words/parsyty.

62. Dotenv [Електронний ресурс]. –
Режим доступу: https://www.npmjs.com/package/dotenv.

63. Connection Strings [Електронний ресурс]. –
Режим доступу: https://www.postgresql.org/docs/current/libpq-connect.html#LIBPQ-CONNSTRING.

64. Sequelize [Електронний ресурс]. –
Режим доступу: https://sequelize.org/api/v6/class/src/sequelize.js~sequelize#instance-method-define.

65. Create a new app [Електронний ресурс]. –
Режим доступу: https://docs.expo.dev/get-started/create-a-new-app/.

ДОДАТОК А

Лістінг App.tsx

```
// зберігати состояніє будь яке (наприклад значення яке вводиться в інпут)
import { useState } from "react";
// компоненти
import { Button, Text, TextInput, View } from "react-native";
// пакет який використовуємо для того щоб робити запити http
import axios from "axios";
// градієнт
import { LinearGradient } from "expo-linear-gradient";

// куди будемо робити запити
const url = "http://localhost:3000";

// основка функція
export default function App() {
  // isLogin - значення, setIsLogin - функція для зміни значення
  const [isLogin, setIsLogin] = useState(false);
  const [error, setError] = useState("");
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");

  if (isLogin) {
    return (
      <View
        style={{
          flex: 1,
          backgroundColor: "#fff",
          alignItems: "center",
          justifyContent: "center",
        }}
      >
        <LinearGradient
          colors={["rgba(69, 131, 255, 0.8)", "rgba(255, 221, 48, 0.8)"]}
          style={{
            position: "absolute",
            left: 0,
            right: 0,
            top: 0,
            bottom: 0,
```

```
      }}
    />
    <Text style={{ fontSize: 25, marginBottom: 30 }}>Успішно!!!</Text>
    <Button
      title="Вийти"
      onPress={() => {
        setIsLogin(false);
        setError("");
        setUsername("");
        setPassword("");
      }}
    />
  </View>
 );
} else {
 return (
  <View
    style={{
      flex: 1,
      backgroundColor: "#fff",
      alignItems: "center",
      justifyContent: "center",
    }}
  >
    <LinearGradient
      // Background Linear Gradient
      colors={["rgba(69, 131, 255, 0.8)", "rgba(255, 221, 48, 0.8)"]}
      style={{
        position: "absolute",
        left: 0,
        right: 0,
        top: 0,
        bottom: 0,
      }}
    />
    <Text style={{ fontSize: 25, marginBottom: 30 }}>Додаток</Text>
    <TextInput
      style={{
        width: "90%",
        borderWidth: 2,
        borderColor: "grey",
        borderRadius: 5,
        height: 50,
        marginTop: 20,
        paddingHorizontal: 5,
```

```
    backgroundColor: "rgba(255, 255, 255, 0.8)",
  }}
  placeholder="Username"
  placeholderTextColor={"grey"}
  value={username}
  onChangeText={setUsername}
  autoCapitalize="none"
/>
<TextInput
  style={{
    width: "90%",
    borderWidth: 2,
    borderColor: "grey",
    borderRadius: 5,
    height: 50,
    marginTop: 20,
    marginBottom: 10,
    paddingHorizontal: 5,
    backgroundColor: "rgba(255, 255, 255, 0.8)",
  }}
  placeholder="Password"
  placeholderTextColor={"grey"}
  autoCapitalize="none"
  secureTextEntry={true}
  value={password}
  onChangeText={setPassword}
/>

<Text
  style={{
    color: "red",
    paddingHorizontal: 20,
    textAlign: "center",
    minHeight: 100,
  }}
>
  {error}
</Text>
<Button
  title="Увійти"
  onPress={() => {
    axios
      .post(url + "/login", {
        username,
        password,
```

```
            })
            .then((v) => {
              setIsLogin(true);
            })
            .catch(() => {
              setError(
                "Не правильне ім`я користувача або пароль, спробуйте ще раз"
              );
            });
        }}
      />
      <Button
        title="Зареєструватися"
        onPress={() => {
          axios
            .post(url + "/signup", {
              username,
              password,
            })
            .then((v) => {
              setIsLogin(true);
            })
            .catch(() => {
              setError("Помилка при реєстрації, спробуйте ще раз");
            });
        }}
      />
    </View>
  );
 }
}
```