DOI: https://doi.org/10.36910/6775-2524-0560-2025-60-12
УДК: 004.45
Гришанович Тетяна Олександрівна, к.ф.-м.н., доцент
https://orcid.org/0000-0002-3595-6964
Загура Юлія Вікторівна, здобувачка освіти
Волинський національний університет імені Лесі Українки, м. Луцьк, Україна

РЕАЛІЗАЦІЯ ГІБРИДНОГО АЛГОРИТМУ ГЕНЕРУВАННЯ ПАРОЛІВ У ВИГЛЯДІ ВЕБЗАСТОСУНКУ

Гришанович Т. О., Загура Ю. В. Реалізація гібридного алгоритму генерування паролів у вигляді вебзастосунку. У статті розглянуто проблему забезпечення надійності паролів як одного з ключових інструментів автентифікації та захисту доступу до цифрових ресурсів. Попри розвиток біометричних та багатофакторних технологій, саме паролі залишаються найбільш поширеним засобом безпеки завдяки простоті та універсальності. Водночас ненадійні комбінації є причиною більшості випадків зламу облікових записів, що зумовлює потребу у створенні ефективних механізмів їх генерації та управління. У роботі запропоновано вебзастосунок для автоматизованого формування складних паролів з можливістю налаштування параметрів довжини та складу символів, а також інтеграцією словникових вставок для підвищення зручності. Система реалізує гібридний алгоритм, який поєднує випадкову генерацію символів та словникові елементи, забезпечуючи баланс між стійкістю до злому і практичною запам'ятовуваністю. Додатковою перевагою є підтримка сучасних алгоритмів шифрування (SHA-256, MD5, Base64 тощо), що дозволяє адаптувати застосунок до різних сценаріїв використання. Веборієнтована архітектура, побудована на HTML5, CSS та JavaScript, гарантує автономність і конфіденційність роботи без необхідності серверної обробки. Розробка спрямована на підвищення рівня інформаційної безпеки та мінімізацію людського фактора у створенні паролів. Перспективи подальших досліджень полягають у використанні машинного навчання, інтеграції з корпоративними середовищами та розробці мобільних рішень.
Ключові слова: паролі, інформаційна безпека, вебзастосунок, генерація паролів, алгоритми шифрування, гібридний підхід, автентифікація

Hryshanovych T., Zahura Y. Implementation of a Hybrid Password Generation Algorithm as Web Application. The article addresses the problem of ensuring password reliability as a key tool for user authentication and access control in digital environments. Despite the development of biometric and multifactor authentication technologies, passwords remain the most common security mechanism due to their simplicity, universality, and cost-effectiveness. However, weak or reused passwords are the cause of the majority of account breaches, which highlights the need for efficient mechanisms of password generation and management. The paper presents a web application for automated generation of strong passwords with customizable parameters, including length, symbol composition, and optional dictionary-based word insertion to improve usability. The proposed system implements a hybrid algorithm that combines random symbol generation with dictionary elements, providing a balance between resistance to brute-force attacks and practical memorability. An additional feature is the integration of modern encryption methods (SHA-256, MD5, Base64, etc.), enabling adaptability to various security scenarios. The client-side architecture, based on HTML5, CSS, and JavaScript, ensures confidentiality and autonomy without requiring server-side processing. This development aims to enhance information security by reducing human-related risks in password creation. Future research perspectives include the use of machine learning for adaptive complexity adjustment, integration with corporate systems, and the development of mobile and cross-platform solutions.
Keywords: passwords, information security, web application, password generation, encryption algorithms, hybrid approach, authentication.

Постановка проблеми. Паролі відіграють роль одного з основних засобів автентифікації користувачів і контролю доступу до різноманітних інформаційних ресурсів. Саме вони забезпечують базовий рівень захисту від несанкціонованого доступу до персональних даних, електронної пошти, хмарних сервісів, банківських систем, корпоративних платформ тощо. Паролі не лише виконують технічну функцію, вони є ключовим елементом довіри між користувачем та системою. Суть значення паролів полягає у створенні межі між відкритим і захищеним цифровим простором. Безпарольний доступ навіть до звичайного облікового запису в соціальній мережі відкриває можливість для збору конфіденційної інформації, шахрайства або поширення шкідливого програмного забезпечення. У масштабах організацій неправильне управління паролями часто призводить до викрадення даних, фінансових втрат або порушення безперервності бізнес-процесів. Згідно з аналітичними звітами, понад 80 % випадків зламу облікових записів відбуваються через проблеми, пов'язані з паролями, їх ненадійність, повторне використання або відсутність належного захисту при зберіганні [1], через це ефективна політика управління паролями є не просто рекомендацією, а необхідною умовою функціонування захищеного цифрового середовища.

Варто зазначити, що попри розвиток сучасних технологій автентифікації (біометрія, токени, багатофакторна перевірка), паролі залишаються найпоширенішим способом захисту. Це пояснюється їхньою простотою реалізації та використання, універсальністю та відносною

дешевизною. Водночас зловмисники часто використовують соціальну інженерію, фішингові атаки або методи брутфорсу для компрометації облікових записів саме через слабкі або вкрадені паролі [2]. Значення паролів полягає у забезпеченні контролю доступу до ресурсів, захисті від атак та підтриманні конфіденційності даних. Їхня наявність і правильне використання — це фундамент, на якому будується загальна система інформаційної безпеки як для окремої особи, так і для цілої організації.

Метою даної роботи є розробка вебсистеми для генерування надійних паролів з різними параметрами складності та реалізація механізму їхнього шифрування для подальшого використання в інформаційних системах. Така програмна розробка підвищить рівень захисту даних шляхом створення ефективних та зручних у використанні засобів генерування паролів, оскільки автоматизоване створення паролів різної складності із можливістю подальшого шифрування дозволяє мінімізувати людський фактор та підвищити безпеку цифрових сервісів.

Доцільно, аби розроблюваний програмний продукт забезпечував наступні вимоги:

- генерування паролів різної довжини, користувач повинен мати можливість встановлювати довжину паролю в межах від 4 до 80 символів;
- налаштування складу символів, коли інтерфейс дозволяє обирати типи символів, які включатимуться до пароля: великі літери, малі літери, цифри, спеціальні символи, слова латиницею та кирилицею;
- автоматична оцінка складності пароля, під час якої згенерований пароль оцінюється за шкалою з урахуванням довжини, унікальності та типів символів;
- підтримка шифрування, користувач може застосовувати одне з кількох шифрувань (наприклад, SHA-256, Base64, ROT13, MD5, Reverse).

Аналіз останніх досліджень і публікацій.

Надійність паролю безпосередньо впливає на ефективність захисту облікових записів у цифровому середовищі. У звіті Cambridge Support [3] наведено п'ять ключових правил створення сильного паролю:

- мінімальна довжина має становити 12 символів, а для важливих облікових записів рекомендується використовувати ще довші комбінації;
- до складу надійного паролю повинні входити великі та малі літери, цифри, а також спеціальні символи, адже такий підхід значно ускладнює спроби злому шляхом перебору варіантів;
- не використовувати імена, дати народження, назви улюблених команд чи домашніх тварин;
- один і той самий пароль для кількох сервісів значно підвищує ризик втрати доступу в разі компрометації одного з облікових записів;
- навіть найнадійніший пароль із часом може бути вкраденим, тому рекомендується змінювати паролі періодично, особливо після підозрілої активності.

Інфографіка [3] демонструє, як швидко можуть бути зламані паролі залежно від їхньої структури та довжини. Наприклад, пароль, який складається лише з малих літер, довжиною до 8 символів, зламується миттєво. Навіть додавання однієї великої літери або цифри майже не впливає на результат. Лише при довжині 12 символів і наявності великої літери, цифри та символу час злому може сягати десятків тисяч років. Пароль із 14 символів, що містить усі типи символів, потребує приблизно 200 мільйонів років для злому, що фактично робить його майже недосяжним для сучасних засобів атаки перебором. Натомість прості паролі з 6–8 символів, навіть із додатковими елементами, зламуються за хвилини або години. Дотримання вищезгаданих вимог не є простою формальністю, це обов'язкова умова для забезпечення базової цифрової безпеки. Щоб створити справді безпечний пароль, який буде важко зламати, але зручно використовувати, розробляються різні методи та алгоритми. Серед них найбільш відомими є методи випадкової генерування, марківські моделі та підходи, що базуються на технологіях глибокого навчання.

Випадкове генерування паролів — це один із найпростіших і найпоширеніших способів. Пароль створюється шляхом випадкового вибору символів із певного набору (літери, цифри, спеціальні символи). Такий пароль має високу ентропію, тобто є складним для передбачення, однак часто його важко запам'ятати. У праці [4] наголошується, що користувачі мають тенденцію створювати передбачувані шаблони, тому автоматичне генерування допомагає уникнути людського фактора. Марківські моделі — це математичний підхід, який дозволяє генерувати паролі на основі ймовірності переходу між символами. Таким чином створюються комбінації, схожі на справжні слова, але водночас унікальні. Наприклад, модель OMECDN поєднує марківський процес і штучну

нейронну мережу, що дозволяє отримати більш реалістичні та захищені паролі [5]. Глибоке навчання — це сучасний напрям, що передбачає використання нейронних мереж для генерування нових паролів. Дослідження [6] показу', що моделі на базі автоенкодерів або генеративно-змагальних мереж здатні створювати паролі з високою складністю та унікальністю, що робить їх менш вразливими до атак. Сьогодні розробляється багато підходів до генерування паролів, кожен із яких має свої переваги: випадкове генерування проста, марківські моделі зручніші для запам'ятовування, а нейромережі найперспективніші для майбутнього. Вибір методу залежить від конкретних цілей та вимог до безпеки.

Станом на сьогодні існує значна кількість програм-генераторів паролів, одними із напоширеніших із яких є LastPass Password Generator [7], NordPass Password Generator [8], Dashlane Password Generator [9].

У таблиці 1 наведено порівняльний аналіз вищеназваних програмних продуктів.

Таблиця 1. Порівняльна таблиця генераторів паролів

Характеристика	LastPass	NordPass	Dashlane
Розробник	LastPass	Nor Security	Dashlane Inc.
Архітектура	Вебзастосунок	Вебзастосунок	Вебзастосунок
Мова реалізації	JavaScript	JavaScript	JavaScript
Максимальна довжина паролю	50 символів	60 символів	40 символів
Особливості	Паролі, що легко запам'ятовуються, інтеграція з менеджером, налаштування типів символів	Оцінка безпеки в реальному часі, зручний інтерфейс, зберігання в NordPass	Паролі, що легко запам'ятовуються, оцінка безпеки, зберігання в Dashlane
Переваги	Безкоштовність, зручність, локальне генерування	Гнучкість, простота, онлайн-оцінка безпеки	Простота, якісний UX, хороша репутація
Недоліки	Обмеження без підписки	Ряд функції тільки через менеджер	Обмежена автономність

На основі проведеного аналізу можна зробити висновок, що всі розглянуті генератори паролів мають зручний вебінтерфейс, підтримують гнучкі налаштування та забезпечують базову безпеку. Водночас кожен із них є частиною більшої екосистеми (менеджера паролів), що обмежує їхню автономність і змушує користувача встановлювати додаткові додатки або створювати обліковий запис.

У зв'язку з цим доцільною є розробка власної вебсистеми генерування паролів, яка буде поєднувати всі найкращі функції аналогів, але при цьому:

- не матиме прив'язки до сторонніх систем чи менеджерів паролів;
- дозволятиме створювати паролі довжиною до 80 символів;
- міститиме всі функції сучасних генераторів: вибір типів символів, налаштування складності, створення зручних для читання паролів;
- дозволятиме експортувати згенерований пароль у текстовий файл (.txt), щоб користувач міг зберегти його офлайн.

Викладення основного матеріалу і обґрунтування отриманих результатів.

На етапі проєктування програмного засобу для генерування паролів було розроблено діаграму варіантів використання, яка є одним із ключових інструментів специфікації вимог у UML-

нотації. Дана діаграма дозволяє наочно показати, які саме дії може виконувати користувач у системі, як ці дії пов'язані між собою, та яка роль користувача в процесі генерування паролю (рис. 1).

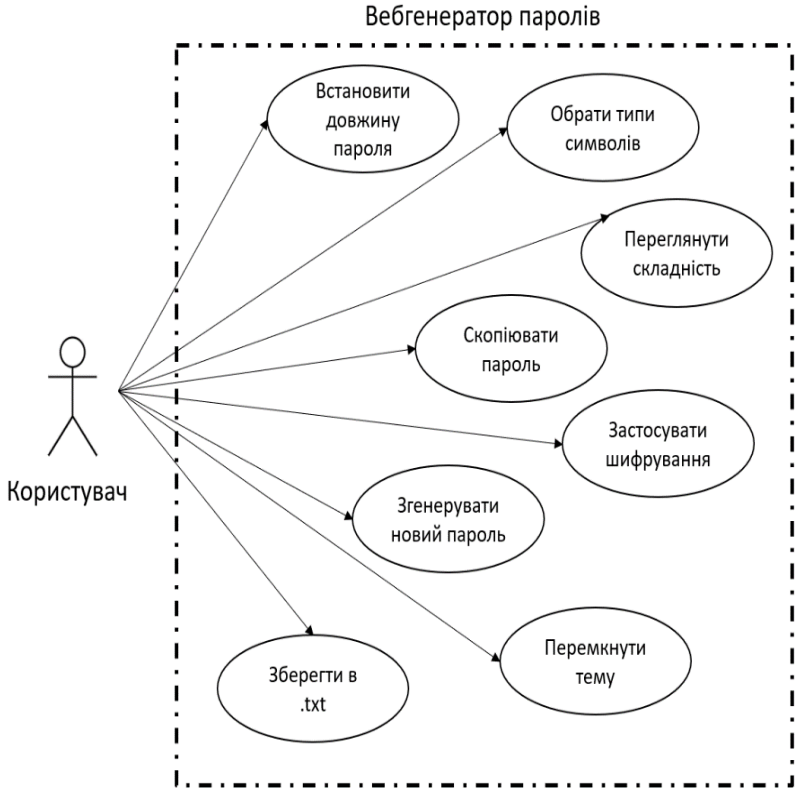

Рис. 1. Діаграма варіантів використання

Серед доступних дій, які може виконувати користувач:

- встановити довжину пароля;
- обрати типи символів;
- переглянути складність;
- скопіювати пароль;
- застосувати шифрування;
- згенерувати новий пароль;
- зберегти в .txt;
- перемкнути тему.

Ця діаграма відображає лише основні сценарії роботи одного користувача з інтерфейсом генератора паролів без деталізації внутрішніх процесів чи альтернативних потоків.

У процесі розробки програмного забезпечення, що пов'язане із захистом інформації, особливе місце посідає створення стійких до зламу паролів. Саме тому підхід до його генерування повинен бути гнучким, керованим користувачем і побудованим за зрозумілим, але ефективним алгоритмом. У розробленій вебсистемі реалізовано гібридний алгоритм генерування паролів, який поєднує класичні методи випадкового добору символів та словникову вставку, що підвищує зручність і варіативність створених комбінацій. На рисунку 2 подано діаграму активності, яка наочно демонструє логіку реалізованого алгоритму генерування паролів.

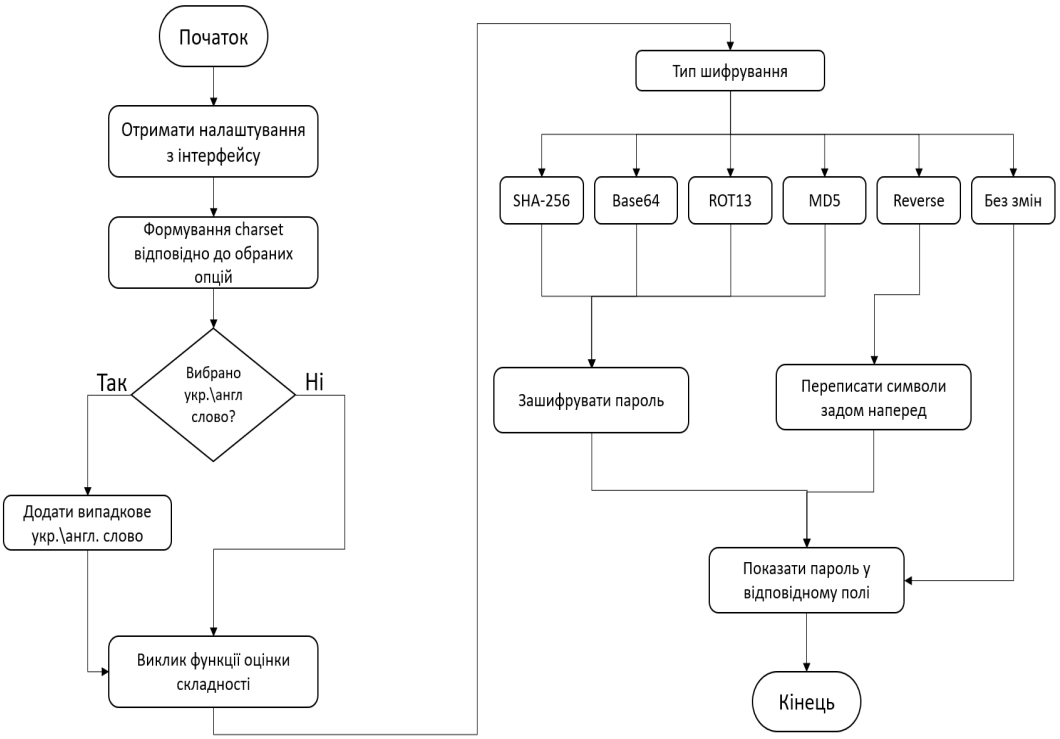

Рис. 2. Діаграма активності

Діаграма розділена на дві ключові частини: ліва частина відповідає за процес створення паролю відповідно до налаштувань користувача, а права — за застосування вибраного методу шифрування.

Процес починається з отримання налаштувань із графічного інтерфейсу. Користувач задає параметри, такі як довжина пароля, типи символів, які мають бути використані, та вибір словникової вставки. На основі цих параметрів формується набір символів (charset), з якого в подальшому буде складено випадкову комбінацію. Після цього система перевіряє, чи користувач увімкнув опцію додавання українського або англійського слова. Якщо так — до паролю додається випадкове слово зі словника. Якщо ж ця опція не активована, система переходить до наступного етапу.

Незалежно від цього, наступним кроком викликається функція оцінки складності згенерованого пароля. Вона аналізує створений пароль за кількома критеріями: довжиною, різноманітністю символів, кількістю унікальних символів тощо. Це дозволяє візуально інформувати користувача про рівень захищеності його пароля.

Далі відбувається перехід до гілки вибору шифрування. Користувач може обрати один із запропонованих алгоритмів: SHA-256, Base64, ROT13, MD5, Reverse або ж залишити пароль без змін. У випадках із SHA-256, Base64, ROT13 або MD5 система застосовує відповідну хеш-функцію для перетворення пароля. Якщо обрано Reverse, пароль просто переписується у зворотному порядку. Якщо ж обрано варіант «Без змін», шифрування не виконується.

Завершальним етапом є відображення згенерованого паролю у відповідному полі інтерфейсу. На цьому робота алгоритму завершується.

Описаний алгоритм поєднує випадкову генерацію з можливістю словникової вставки та додаткового шифрування, що дозволяє реалізувати різні сценарії створення паролів залежно від потреб користувача. Його гнучкість, модульність і простота реалізації роблять його ефективним для використання у вебсистемах з підвищеними вимогами до безпеки.

Програмна реалізація генератора паролів виконана з використанням стандартного HTML5 для побудови інтерфейсу, CSS — для візуального оформлення та JavaScript — для реалізації основної логіки. Уся система працює на клієнтській стороні, що забезпечує конфіденційність та автономність.

Користувач може керувати генерацією паролю за допомогою слайдера та чекбоксів. Значення параметрів зчитуються за допомогою DOM-елементів:

```
const lengthInput = document.getElementById("length");
const checkBoxes = document.querySelectorAll(".options input");
const encryptionSelect = document.getElementById("encryption");
```

Ці змінні отримують доступ до елементів інтерфейсу, які задають довжину пароля, типи символів і спосіб шифрування. Це дозволяє реагувати на будь-які зміни, зроблені користувачем.

Функція generatePassword() є основним елементом логіки вебзастосунку, що відповідає за формування паролю відповідно до вибраних користувачем параметрів. Її робота базується на гібридному підході, що поєднує словникову генерацію (за потреби) з випадковим добором символів, що забезпечує високу варіативність і складність пароля.

На початку функції відбувається зчитування стану всіх елементів інтерфейсу, які задають правила генерування:

```
const length = +lengthInput.value;
const useUpper = document.getElementById("uppercase").checked;
const useLower = document.getElementById("lowercase").checked;
const useNumbers = document.getElementById("numbers").checked;
const useSymbols = document.getElementById("symbols").checked;
const useUA = document.getElementById("ukrainianWords").checked;
const useEN = document.getElementById("englishWords").checked;
const encryption = encryptionSelect.value;
```

Таким чином, система «розуміє», які символи має використовувати і чи потрібно додавати словникові слова.

На основі обраних типів символів формується змінна charset, що містить допустимі символи для випадкового генерування:

```
let charset = "";
if (useUpper) charset +="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
if (useLower) charset +="abcdefghijklmnopqrstuvwxyz";
if (useNumbers) charset +="0123456789";
if (useSymbols) charset +="!@#$%^&*()-_=+[]{}<>.,&/\";
```

Цей набір буде використаний на наступному етапі для добору випадкових символів. Функція дає змогу користувачеві вставити на початок паролю випадкове слово з одного з попередньо підготовлених масивів:

```
let password = "";
if (useUA) password +=uaWords[Math.floor(Math.random()*uaWords.length)];
if (useEN) password +=enWords[Math.floor(Math.random()*enWords.length)];
```

Якщо обрано хоча б одну з мов, до паролю додається слово.

Основна частина паролю добирається випадковим чином із charset. Цикл триває доти, доки загальна довжина його не відповідатиме обраній:

```
for (let i = password.length; i < length; i++){
password += charset.charAt(Math.floor(Math.random()*charset.length));
}
```

Таким чином, гарантується точна довжина, незалежно від наявності словникових вставок.

Після генерування пароль передається до функції оцінки складності evaluateStrength(), а також залежно від вибору користувача, може бути зашифрований відповідним методом. Функція generatePassword() реалізує адаптивний, модульний алгоритм генерування, який:

- враховує всі обрані користувачем параметри;
- комбінує словниковий і символьний підходи;
- забезпечує контроль довжини;
- дозволяє оцінити складність та зашифрувати результат.

Після створення паролю він передається у функцію evaluateStrength(). Яка реалізує алгоритм кількісного аналізу згенерованого пароля, що дозволяє оцінити його стійкість до потенційного злому. Основна мета цієї функції, надати користувачу візуальний зворотний зв'язок про рівень надійності паролю.

Перш за все, зчитується довжина паролю: `const length = password.length;`

Потім за допомогою регулярних виразів визначається наявність чотирьох типів символів:

```
const hasUpper = /[A-ZА-ЯЇІЄҐ]/.test(password); //великі літери
const hasUpper = /[a-za-яїієґ]/.test(password); //малі літери
const hasDigits = /\d/.test(password); //цифри
const hasDigits = /[^A-Za-zA-Яф-яЇїІіЄєҐґ]/.test(password); //спецсимволи
```

Це дозволяє перевірити, наскільки різноманітним є пароль за своїм складом. Оцінюється кількість унікальних символів у паролі за допомогою спеціальної структури даних Set, яка автоматично виключає повтори:

```
const uniqueChars = new Set(password).size;
const uniqueness = uniqueChars / length;
```

Таким чином, якщо пароль містить багато однакових символів (наприклад, аааа1111), його унікальність буде низькою, що у свою чергу також знижує складність.

Далі формується числовий бал (від 0 до 10) залежно від виявлених властивостей:

```
score +=hasUpper ? 1 : 0;
score +=hasLower ? 1 : 0;
score +=hasDigits ? 1 : 0;
score +=hasSymbols ? 1 : 0;
if (length >= 12) score +=1;
if (length >= 20) score +=1;
if (uniqueness > 0.6) score +=1;
if ((hasUpper + hasLower + hasDigits + hasSymbols)) >=3 && length >=14) score +=1;
```

Також додається штраф за повторювані символи:

```
const repeatPenalty = /([a-zA-za-яА-Я\d])\1{2,}/.test(password) & -1 : 0;
score += repeatPenalty;
```

Це дозволяє уникати тривалих повторів. Результуючий бал обрізається в межах 0–10, після чого використовується для визначення текстового рівня складності:

```
const levels = ["Дуже низька", "Низька", "Слабка", "Нижче хорошої", "Хороша",
"Висока", "Дуже висока", "Міцна", "Екстремальна"];
```

Ці рівні прив'язані до відповідних класів CSS (weak, medium, strong, very-strong, extreme), які відображають колір індикатора складності.

Користувач може обрати один із декількох методів шифрування. Кожен із них реалізований окремим блоком. Наприклад, для SHA-256 використовується вбудований Web Crypto API:

```
async function encryptSHA256(message){
        const msgBuffer = new TextEncoder().encode(message);
        const hashBuffer = await crypto.subtle.digest("SHA-256", msgBuffer);
        return Array.from(new Unit8Array(hashBuffer))
                .map(b => b.toString(16).padStart(2, '0')).join('');
}
```

Цей метод повертає хеш паролю, який неможливо розшифрувати назад, що є корисним для зберігання у базах даних.

Генератор також дозволяє зберігати згенерований пароль у файл або копіювати його в буфер обміну:

```
function saveToFile(){
const blob = new Blob([resultField.value], {type: 'text/plain'});
const a = document.createElement("a");
a.href = URL.createObjectURL(blob);
a.download = "password.txt";
a.click();
        }
```

Дана функція створює тимчасовий об'єкт-посилання, який дозволяє користувачу завантажити пароль у вигляді текстового файлу без серверної обробки.

Для зручності користувача реалізовано перемикач між темною та світлою темами інтерфейсу. Стан теми зберігається у локальному сховищі:

```
if (localStorage.getItem("theme")==="light"){
        document.body.classList.add("light");
        themeSwitcher.checked = true;
}
```

Коли користувач змінює тему, обране значення зберігається, і при повторному вході застосунок відкривається з тим же оформленням.

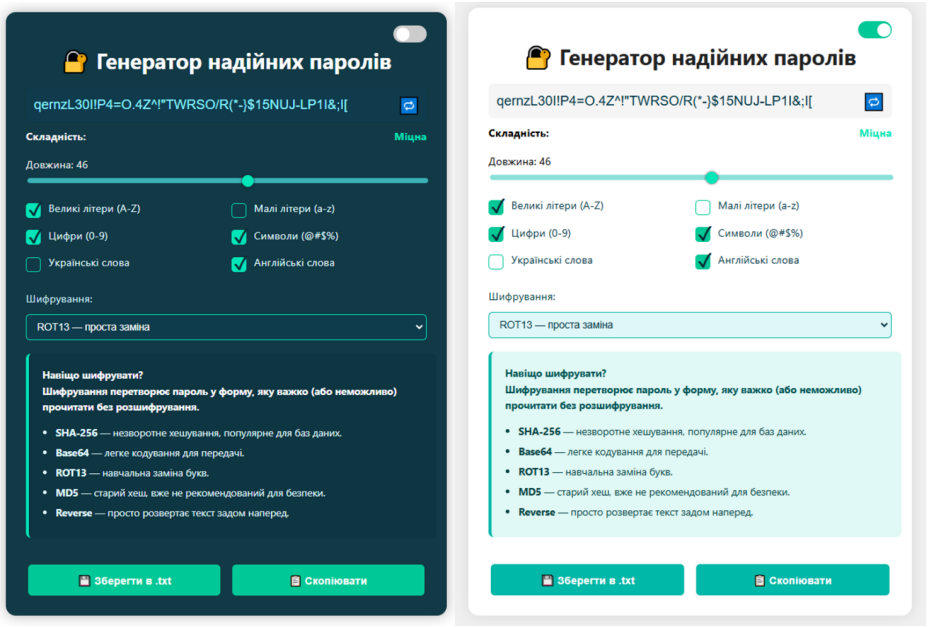

Рис. 3. Інтерфейс програмного засобу (темна та світла тема)

Реалізація генератора паролів базується на модульному підході, де кожен компонент виконує чітко визначену функцію. Інтерфейс є інтуїтивно зрозумілим, програма не вимагає інсталювання і працює у браузері. Програмна розробка надає можливість гнучко налаштовувати майбутній пароль і одразу бачити його складність.

Користувач може додатково обрати метод шифрування, який буде застосовано до згенерованого паролю перед виведенням. На рис. 4 показано, як виглядає розкривне меню з методами.

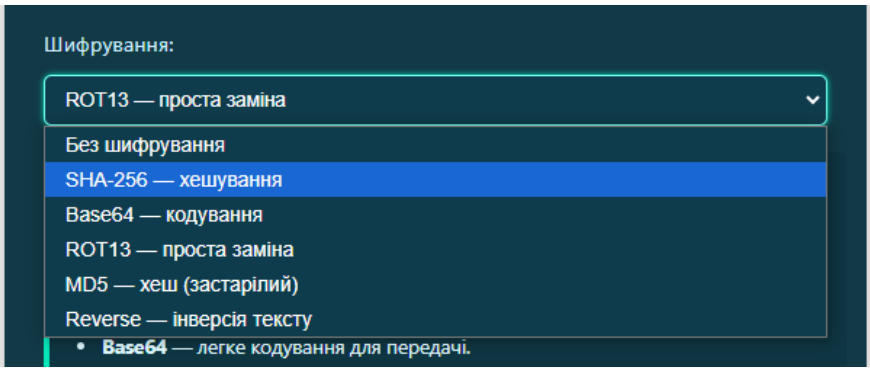

Рис. 4. Розкривне меню методів шифрування

Після створення паролю його можна зберегти у текстовий файл або скопіювати у буфер обміну.

Висновки та перспективи подальших досліджень.

У результаті проведеного дослідження створено вебсистему генерації надійних паролів, яка забезпечує можливість варіативного налаштування параметрів складності, включення словникових елементів та застосування сучасних алгоритмів шифрування. Запропонований гібридний підхід поєднує випадкову вибірку символів зі словниковими вставками, що дозволяє досягти оптимального балансу між стійкістю паролів до злому та зручністю їх практичного використання. Клієнтська реалізація, незалежна від серверної обробки, гарантує конфіденційність і автономність застосунку, що є важливим чинником в умовах зростання кіберзагроз.

Перспективи подальших досліджень полягають у розширенні спектра криптографічних методів, інтеграції системи з менеджерами паролів та корпоративними інформаційними середовищами, а також у використанні технологій машинного навчання для адаптивного формування рівня складності паролів відповідно до контексту їх використання. Доцільним є також розроблення мобільних застосунків та браузерних розширень з метою забезпечення кросплатформності та підвищення зручності. Окремим напрямом виступає вивчення питань підвищення юзабіліті та формування рекомендацій користувачам щодо ефективного управління згенерованими паролями.

Список літератури

1. Звіт про проміжне дослідження щодо інформованості цільових аудиторій про основні аспекти кібербезпеки. 2023 р. *Info Sapiens*. URL: https://www.sapiens.com.ua/publications/socpol-research/321/promizhne-doslidzhennya-shhodo-informovanosti-pro-osnovni-aspekty-kiberbezpeky-1.pdf (дата звернення: 19.03.2025).
2. Cybersecurity incidents caused by weak passwords worldwide. Statista. URL: https://www.statista.com/statistics/1271243/worldwide-security-incidents-caused-by-weak-passwords/ (дата звернення: 1.06.2025).
3. 5 essential rules for creating strong passwords. Cambridge Support. URL: https://cambridgesupport.com/2023/05/5-essential-rules-for-creating-strong-passwords/ (дата звернення: 19.07.2025).
4. Kelsey E. Random Password Generation. ODU Digital Commons. URL: https://digitalcommons.odu.edu/cgi/viewcontent.cgi?article=1029&context=covacci-undergraduateresearch. (дата звернення: 21.07.2025).
5. OMECDN: A Password-Generation Model Based on an Ordered Markov Enumerator and Critic Discriminant Network / J. Jiang et al. *Applied Sciences*. 2022. Vol. 12, no. 23. P. 12379. URL: https://doi.org/10.3390/app122312379 (дата звернення: 28.03.2025).
6. A new smart-cropping pipeline for prostate segmentation using deep learning networks / D. Zaridis et al. 2021. URL: https://doi.org/10.48550/arXiv.2107.02476 (дата звернення: 14.06.2025).
7. Why Should I Choose LastPass Password Manager - LastPass. *LastPass | Something went wrong*. URL: https://www.lastpass.com/why-lastpass?utm_source (дата звернення: 6.06.2025).
8. TOP20 Not-So-Secret Business Passwords. *NordPass*. URL: https://nordpass.com/poor-company-passwords/ (дата звернення: 29.05.2025).
9. Dashlane Review: Pros & Cons, Features, Ratings, Pricing and more. *TechRadar*. URL: https://www.techradar.com/reviews/dashlane?utm_source (дата звернення: 29.05.2025).

References

1. Zvit pro promizhne doslidzhennia shchodo informovanosti tsil′ovykh audytorii pro osnovni aspekty kiberbezpeky. 2023 r.. *Info Sapiens*. URL: https://www.sapiens.com.ua/publications/socpol-research/321/promizhne-doslidzhennya-shhodo-informovanosti-pro-osnovni-aspekty-kiberbezpeky-1.pdf (date of access: 19.03.2025).
2. Cybersecurity incidents caused by weak passwords worldwide. Statista. URL: https://www.statista.com/statistics/1271243/worldwide-security-incidents-caused-by-weak-passwords/ (date of access: 1.06.2025).
3. 5 essential rules for creating strong passwords. Cambridge Support. URL: https://cambridgesupport.com/2023/05/5-essential-rules-for-creating-strong-passwords/ (date of access: 19.07.2025).
4. Kelsey E. Random Password Generation. ODU Digital Commons. URL: https://digitalcommons.odu.edu/cgi/viewcontent.cgi?article=1029&context=covacci-undergraduateresearch. (date of access: 21.07.2025).
5. OMECDN: A Password-Generation Model Based on an Ordered Markov Enumerator and Critic Discriminant Network / J. Jiang et al. *Applied Sciences*. 2022. Vol. 12, no. 23. P. 12379. URL: https://doi.org/10.3390/app122312379 (date of access: 28.03.2025).
6. A new smart-cropping pipeline for prostate segmentation using deep learning networks / D. Zaridis et al. 2021. URL: https://doi.org/10.48550/arXiv.2107.02476 (date of access: 14.06.2025).
7. Why Should I Choose LastPass Password Manager - LastPass. *LastPass | Something went wrong*. URL: https://www.lastpass.com/why-lastpass?utm_source (date of access: 6.06.2025).
8. TOP20 Not-So-Secret Business Passwords. *NordPass*. URL: https://nordpass.com/poor-company-passwords/ (дата звернення: 29.05.2025).
9. Dashlane Review: Pros & Cons, Features, Ratings, Pricing and more. *TechRadar*. URL: https://www.techradar.com/reviews/dashlane?utm_source (date of access: 29.05.2025).