Вінницький національний технічний університет

(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії

(повне найменування інституту)

Кафедра обчислювальної техніки

(повна назва кафедри)

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

ВЕБ-ДОДАТОК ДЛЯ ШИФРУВАННЯ ДАНИХ

Виконав: студент 5 курсу, групи 2КІ-23м

спеціальності 123 – «Комп'ютерна інженерія»

освітня програма – Комп'ютерна інженерія

Радловський Д.В.

(прізвище та ініціали)

Керівник: ст. викл. каф. ОТ

Колесник І.С.

(прізвище та ініціали)

Опонент: к.т.н., доц. кафедри ПЗ

Ракитянська Г.Б.

(прізвище та ініціали)

Допущено до захисту

Завідувач кафедри ОТ

д.т.н., проф. Азаров О. Д.

«___» _____2024 р

Вінниця ВНТУ 2024

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій та комп'ютерної інженерії

Кафедра Обчислювальної техніки

Галузь знань 12 - Інформаційні технології

Освітній рівень - магістр

Спеціальність 123 - «Комп'ютерна інженерія»

Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н. проф Азаров О.Д.

« 17 » вересня 2024 р.

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Радловський Денис Вікторович

1 Тема проекту «Веб-додаток для шифрування даних», керівник проекту Колесник І.С., к.т.н.,доц. кафедри ОТ, затверджені наказом вищого навчального закладу від «17» вересня 2024 р. №310

2 Строк подання студентом проекту «25» грудня 2024 р

3 Вихідні дані до проекту: передавання даних http, максимальна кількість токенів 50, температура 0.5, інтерфейс користувача.

4 Зміст текстової частини (перелік питань, які потрібно розробити): вступ, огляд і аналіз актуальності теми шифрування даних, аналіз розробки програм та методів шифрування даних, вибір технологій та фреймворків, розробка веб-сервісу, розробка рекомендацій з введення в експлуатацію, економічна частина.

5 Перелік графічного матеріалу: інтерфейс реєстрації, інтерфейс програми, лістинг коду програми.

6 Консультанти розділів роботи наведені в таблиці 1.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Колесник І.С., к.т.н.,доц. кафедри ОТ		
5	Небава М.І., к.е.н., професор каф. ЕПВМ		

7 Дата видачі завдання « 18 » вересня 2024 року

8 Календарний план виконання приведений в таблиці 2.

Таблиця 2 — Консультанти роботи

№ з/п	Назва етапів виконання бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі роботи	21.10.24	вик
2	Огляд інформаційних джерел	25.10-08.11.24	вик
3	Огляд і аналіз штучного інтелекту	09.11-20.1124	вик
4	Розробка веб-сервісу з штучним інтелектом	20.11-31.11.24	вик
5	Підготовка матеріалів та опис розробки	01.12-07.12.24	вик
6	Оформлення пояснювальної записки та ілюстративного матеріалу	08.12.24	вик
7	Оцінка комерційного потенціалу розробки		
8	Перевірка якості виконання магістерської кваліфікаційної роботи та усунення недоліків		

Студент _____ Радловський Д.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Колесник І.С.
(підпис) (прізвище та ініціали)

Консультант з економічної частини _____ Небава М.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004

Радловський Д.В. Веб-додаток для шифрування даних. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп’ютерна інженерія, Вінниця: ВНТУ, 2024 — 100 с.

На укр. мові. Бібліогр. 9 назв; рис.: 14; табл. 4; ліст. коду 7.

У магістерській кваліфікаційній роботі розглянуто теоретичний та практичний підхід до розробки веб-додатку для шифрування даних.

Проведено аналіз методів та інструментів створення веб-сервісів, досліджуються підходи до розробки веб-додатків з можливістю шифрування даних.

Результати дослідження показали, що розроблений веб-сервіс для шифрування даних на основі алгоритму AES забезпечує ефективне та безпечне виконання операцій шифрування та дешифрування. Система дозволяє користувачам легко захищати конфіденційну інформацію за допомогою сучасних криптографічних методів. Цей підхід може бути застосований у різних сферах, де необхідний захист даних, зокрема в фінансових установах, медичних організаціях та інших галузях, що працюють з чутливою інформацією.

Ключові слова: криптографія, шифрування, технології, веб-додатки.

ABSTRACT

Radlovskyi D.V. Web Application for Data Encryption. Master's Qualification Thesis in the specialty 123 — Computer Engineering, Vinnytsia: VNTU, 2024 — 100 pages. In Ukrainian. Bibliography: 9 titles; figures: 14; tables: 4; code listings: 7.

This thesis examines the theoretical and practical approaches to developing a web application for data encryption. The analysis of methods and tools for creating web services is conducted, and approaches to the development of web applications with data encryption capabilities are explored.

The results of the research show that the developed web service for data encryption, based on the AES algorithm, provides efficient and secure encryption and decryption operations. The system allows users to easily protect sensitive information using modern cryptographic methods. This approach can be applied in various fields where data protection is required, including financial institutions, healthcare organizations, and other sectors working with sensitive data.

Keywords: cryptography, encryption, technologies, web applications.

ЗМІСТ

ВСТУП

Актуальність теми створення веб-додатків для шифрування даних є надзвичайно високою в сучасному світі. З розвитком цифрових технологій обсяги даних, що передаються через Інтернет, зростають експоненційно. Це охоплює не лише особисте листування, але й проведення фінансових операцій, корпоративну діяльність, медичні записи та інші чутливі аспекти життя, що вимагають високого рівня конфіденційності та захисту. Водночас стрімке зростання кількості кіберзагроз і випадків порушення безпеки даних висуває на передній план потребу в ефективних, зручних та надійних інструментах для захисту інформації.

Одним із ключових методів забезпечення безпеки в цифровому середовищі є шифрування. Ця технологія дозволяє перетворювати відкритий текст у захищений формат, доступний лише за наявності відповідного ключа. Серед сучасних алгоритмів шифрування особливе місце займає AES (Advanced Encryption Standard), який визнаний міжнародним стандартом завдяки своїй стійкості до зламів, продуктивності та широким можливостям застосування. Інтеграція такого алгоритму у веб-додаток створює інструмент, здатний забезпечити конфіденційність і цілісність даних у різних сферах використання.

Об’єктом дослідження є інструменти для створення веб-додатків та впровадження криптографічних алгоритмів для захисту даних.

Предметом дослідження виступають процеси шифрування даних із використанням алгоритму AES у веб-середовищі та їх вплив на забезпечення безпеки інформації.

Метою роботи є розробка веб-додатку для шифрування даних із використанням AES та платформи Firebase для бекенду, що забезпечить надійний захист інформації в умовах сучасного цифрового середовища.

Для досягнення поставленої мети у роботі розв’язані такі задачі:

— проведено аналіз сучасних методів шифрування даних і веб-технологій;

— вибрано інструменти та середовища для розробки, включаючи алгоритм AES і платформу Firebase;

— спроектовано архітектуру веб-додатку з урахуванням вимог до надійності, продуктивності та зручності використання;

— реалізовано шифрування даних із використанням AES і зберігання

інформації через Firebase;

— проведено тестування створеного додатку на відповідність вимогам безпеки, продуктивності та масштабованості.

Наукова новизна дослідження полягає у створенні інтегрованого веб-рішення для шифрування даних, яке поєднує алгоритм AES із хмарною платформою Firebase, що забезпечує високу надійність і зручність у використанні для користувачів із різним рівнем технічної підготовки.

Практичне значення роботи полягає в тому, що створений веб-додаток може бути використаний як індивідуальними користувачами для захисту персональних даних, так і організаціями для забезпечення безпеки корпоративної інформації. Це сприяє підвищенню загального рівня довіри до цифрових технологій та створенню надійного середовища для взаємодії в Інтернеті.

Апробація результатів роботи здійснена у доповіді на Міжнародній науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)».

Таким чином, дана робота є актуальним внеском у забезпечення безпеки даних у цифровому середовищі, відповідаючи сучасним викликам та потребам інформаційного суспільства.

1 АНАЛІЗ СУЧАСНИХ ТЕНДЕНЦІЙ, АРХІТЕКТУР ТА МЕТОДІВ ШИФРУВАННЯ ДАНИХ

1.1 Сучасні тенденції та підходи до забезпечення безпеки даних

У сучасному цифровому світі забезпечення безпеки даних стало пріоритетом для компаній, організацій і звичайних користувачів. З розвитком інформаційних технологій обсяги даних, що передаються та зберігаються, зростають щоденно, що створює нові виклики в галузі кібербезпеки. Розвиток хмарних технологій, штучного інтелекту, блокчейну та багатьох інших інновацій змінює підходи до захисту інформації, ставлячи на перше місце конфіденційність, цілісність і доступність даних.

Однією з ключових тенденцій є активне впровадження хмарних сервісів. Компанії та користувачі дедалі частіше обирають хмарні платформи для зберігання інформації через їхню зручність, масштабованість і доступність. Але ці переваги супроводжуються ризиками, пов’язаними з безпекою: зловмисники можуть намагатися отримати доступ до даних через слабкі місця в конфігураціях або недосконалі механізми автентифікації. Для зменшення таких ризиків широко використовують шифрування даних як на рівні клієнта, так і сервера. Це забезпечує захист інформації навіть у випадку, якщо вона потрапляє до рук зловмисників.

Штучний інтелект також відіграє важливу роль у забезпеченні безпеки. Його алгоритми дозволяють виявляти загрози, які раніше залишалися непоміченими, та автоматизувати процеси захисту. Наприклад, технології машинного навчання аналізують трафік у реальному часі, ідентифікують підозрілу активність і можуть блокувати потенційні атаки ще до того, як вони завдадуть шкоди. Завдяки цьому штучний інтелект стає потужним інструментом у боротьбі з кіберзагрозами, підвищуючи ефективність існуючих методів захисту.

Не менш важливим підходом є інтеграція багатофакторної автентифікації, що стала стандартом для багатьох систем. Її використання забезпечує додатковий рівень захисту, оскільки для доступу до даних потрібно підтвердження через кілька незалежних каналів, наприклад, пароль і одноразовий код із мобільного додатка. Це значно ускладнює роботу зловмисників, навіть якщо один із факторів безпеки був скомпрометований.

Окремої уваги заслуговують блокчейн-технології, які все частіше застосовують для зберігання даних. Завдяки децентралізованій природі блокчейн гарантує, що записи в системі не можуть бути змінені або видалені без відома всіх учасників. Це створює високий рівень довіри до інформації, що зберігається, і робить блокчейн перспективним інструментом для побудови захищених систем у сферах фінансів, охорони здоров'я та державного управління.

Розробники сучасних систем приділяють багато уваги проєктуванню архітектури додатків, враховуючи потребу в безпеці з самого початку. Зокрема, використовуються такі принципи, як мінімізація доступу, яка обмежує права користувачів лише необхідним мінімумом, та модульність, що дозволяє швидко замінювати або вдосконалювати окремі компоненти без впливу на загальну безпеку системи.

Популярним підходом є також регулярний аудит безпеки та тестування систем на проникнення. Це дозволяє виявити вразливості на ранніх етапах і запобігти можливим атакам. Крім того, сучасні автоматизовані системи моніторингу, такі як SIEM, забезпечують аналіз і кореляцію даних у реальному часі, допомагаючи швидко реагувати на інциденти.

Значну роль у забезпеченні безпеки відіграє людський фактор. Навіть найдосконаліша технологія не може повністю захистити дані, якщо користувачі нехтують базовими правилами кібергігієни. Тому важливо проводити навчання для співробітників і користувачів, пояснюючи їм, як створювати надійні паролі, розпізнавати фішингові атаки та уникати небезпечних дій, які можуть поставити під загрозу безпеку системи.

Сучасні підходи до забезпечення безпеки даних передбачають не лише впровадження новітніх технологій, а й створення цілісної стратегії, яка враховує технічні, організаційні та освітні аспекти. Усе це сприяє побудові систем, здатних ефективно протистояти загрозам, зберігаючи довіру користувачів і забезпечуючи стабільність у роботі організацій.

Тема розробки веб-додатка для шифрування даних є надзвичайно актуальною в умовах сучасного цифрового світу, де щоденно генеруються величезні обсяги інформації. Зі зростанням популярності онлайн-сервісів і платформ для обміну даними, питання захисту інформації стає дедалі важливішим. Користувачі передають конфіденційні дані через мережу Інтернет — від особистих повідомлень і фотографій

до банківських транзакцій і комерційної документації. Усі ці дані можуть бути вразливими до несанкціонованого доступу, крадіжки або зміни.

Сучасний світ стикається з постійними кіберзагрозами, які мають тенденцію до ускладнення. Зловмисники використовують усе більш витончені методи атак, що ставить під сумнів безпеку навіть добре захищених систем. Результатом таких дій можуть бути значні фінансові втрати, компрометація репутації компаній та особисті наслідки для окремих користувачів. Тому розробка інструментів, які дозволяють ефективно захищати дані, є ключовою складовою сучасних інформаційних технологій.

Окремо варто зазначити, що зі збільшенням обсягів інформації, яку користувачі зберігають і передають через мережу, виникає потреба в універсальних і доступних рішеннях для захисту даних. Розробка веб-додатка для шифрування даних дозволяє забезпечити доступність цієї технології широкому колу користувачів, адже веб-додатки не потребують встановлення складного програмного забезпечення і можуть працювати на будь-якому пристрої з доступом до Інтернету.

Крім того, питання безпеки даних набуває особливого значення у зв’язку з глобалізацією економіки та активним переходом бізнесу в онлайн-середовище. Компанії дедалі частіше обирають цифрові рішення для ведення своєї діяльності, що, у свою чергу, підвищує залежність від технологій захисту даних. Забезпечення безпеки корпоративної інформації є не лише вимогою часу, а й обов’язковою умовою дотримання міжнародних стандартів і нормативних актів.

Важливим аспектом є також приватність користувачів. У світі, де приватна інформація часто стає предметом комерційного інтересу чи зловживань, люди все більше прагнуть до захисту своїх даних. Надійне шифрування дозволяє створити додатковий рівень довіри до онлайн-сервісів і допомагає користувачам почуватися впевнено, знаючи, що їхня інформація

1.2 Криптографія: визначення та значення

Криптографія — це фундаментальна наука, яка забезпечує безпеку інформації в сучасному цифровому світі, що стрімко розвивається. Вона відіграє ключову роль у підтримці довіри до технологій, які дедалі більше впливають на всі сфери життя, від особистих комунікацій до глобальних фінансових транзакцій. Завдяки криптографії,

користувачі можуть бути впевнені, що їхні дані залишаються захищеними, навіть у середовищі, яке стає все складнішим і потенційно небезпечним.

Захист даних у сучасному світі не є тривіальним завданням. Щодня передаються мільярди гігабайтів інформації, і кожна транзакція, повідомлення чи файл можуть бути мішенню для атак. Цифрові сервіси, якими ми користуємося, — від інтернет-банкінгу до соціальних мереж — вимагають впровадження найкращих рішень у галузі криптографії, щоб гарантувати конфіденційність і цілісність даних. Тут на перший план виходить не тільки технологічний аспект, але й довіра користувачів до систем. Саме криптографія створює основу для цієї довіри, оскільки вона забезпечує захист навіть у випадках, коли комунікаційні мережі чи сервіси є вразливими.

Історично криптографія завжди реагувала на виклики свого часу. Якщо у Стародавньому Єгипті чи Римі її використовували для захисту військових чи політичних повідомлень, то в сучасному світі її роль розширилася до глобального масштабу. Вона стала ключовим елементом у таких критично важливих сферах, як електронна комерція, онлайн-банкінг, охорона здоров'я й навіть кібербезпека національного рівня. Технології, які раніше застосовувалися вузьким колом фахівців, тепер впроваджуються у повсякденному житті. Це ще раз підкреслює необхідність постійного вдосконалення алгоритмів і методів шифрування для адаптації до нових викликів, таких як кіберзагрози чи розвиток квантових технологій.

Сучасний світ вимагає не просто розробки надійних криптографічних рішень, а й їх інтеграції в повсякденне життя. Успішні приклади цієї інтеграції — це протоколи SSL/TLS, що забезпечують безпечне з'єднання між користувачами та веб-сервісами. Але як саме вони впливають на користувача? Перш за все, ці технології створюють непомітний, але надійний "щит", який захищає паролі, платіжні дані чи особисту інформацію під час передавання через інтернет. Коли користувач бачить "замок" у рядку адреси браузера, це не просто символ — це результат багаторічної еволюції криптографії, яка спрямована на створення безпечного цифрового середовища.

Ще одним важливим прикладом є месенджери з наскрізним шифруванням, такі як Signal або WhatsApp. Вони стали стандартом у сфері конфіденційності особистих комунікацій. У цих випадках криптографія забезпечує анонімність і захист користувачів навіть у тих країнах, де існують суворі обмеження свободи слова. Такі інструменти не лише гарантують технічний захист, але й формують важливу етичну

основу для сучасного інформаційного суспільства, де право на приватність стає одним із основних прав людини.

Крім захисту окремих користувачів, криптографія є фундаментальною технологією для корпоративного сектору. Безпечне передавання даних між підприємствами, захист інтелектуальної власності, забезпечення надійності транзакцій — усе це було б неможливим без використання сучасних алгоритмів шифрування. Наприклад, у фінансовій сфері саме криптографія дозволяє захищати мільярди доларів від крадіжок і шахрайства, забезпечуючи стабільність економіки в глобальному масштабі.

Однак технологічний прогрес не стоїть на місці. Криптографія змушена постійно адаптуватися до нових викликів, таких як потенційний вплив квантових комп'ютерів. У сучасних реаліях виникає потреба в розробці стійких до квантових обчислень алгоритмів, які б гарантували безпеку навіть за умов, коли традиційні методи стануть вразливими.

Таким чином, криптографія є більше ніж просто технічним інструментом. Це цілий набір принципів і технологій, що гарантують стабільність, безпеку й довіру до цифрового світу. Її значення важко переоцінити, особливо в умовах глобальної цифровізації та постійного зростання кіберзагроз. Без криптографії сучасне суспільство було б значно вразливішим, що підтверджує необхідність постійного розвитку цієї галузі.

1.3 Огляд сучасного стану шифрування даних

Огляд сучасного стану шифрування даних дозволяє оцінити ключові тенденції, технології та новітні підходи в галузі криптографії, що розвиваються на фоні зростаючих обсягів оброблюваної інформації та підвищених вимог до захисту даних. Зокрема, однією з основних цілей сучасних систем шифрування є забезпечення високого рівня безпеки інформації, яка передається через відкриті канали зв’язку, такі як Інтернет. Враховуючи постійну еволюцію обчислювальних потужностей і новітніх атакуючих технологій, захист даних стає надзвичайно складною задачею, що вимагає розробки ефективних криптографічних алгоритмів.

Одним з основних напрямків розвитку сучасних шифрів є використання складних математичних принципів, що забезпечують стійкість алгоритмів до потенційних атак.

Сучасні методи шифрування включають асиметричні алгоритми, хешування, блочні шифри, а також використання ключових шифрів різної довжини. Вони формують надійні механізми захисту даних, значно ускладнюючи можливість зловмисників отримати доступ до зашифрованої інформації. Однак незважаючи на досягнення у створенні криптографічних алгоритмів, нові загрози вимагають від науковців і інженерів постійного оновлення та вдосконалення існуючих рішень.

Одним із найбільших викликів для криптографії на сьогодні є швидкий розвиток обчислювальних технологій. Завдяки зростаючим можливостям обчислювальних потужностей, зловмисники можуть використовувати новітнє апаратне забезпечення, щоб здійснювати атакувальні операції, що раніше були неможливими. Використання спеціалізованих процесорів і доступ до потужних обчислювальних кластерів створюють нові загрози для традиційних криптографічних систем. Саме тому виникає потреба в розробці нових алгоритмів шифрування, які б були здатні витримувати такі атаки. Це включає використання більш складних математичних операцій, таких як багатоступеневі функції хешування або квантову криптографію, що використовує принципи квантової механіки для підвищення стійкості до атак.

Зокрема, квантова криптографія є одним із найбільш перспективних і інноваційних напрямків у галузі захисту даних. Вона здатна забезпечити неймовірно високий рівень безпеки, завдяки використанню квантових принципів, таких як квантова заплутаність, яка дозволяє значно ускладнити спроби перехоплення або змінення даних під час їх передачі. В основі квантової криптографії лежить ідея, що жоден зловмисник не зможе перехопити квантову інформацію без її зміни, що дозволяє забезпечити високий рівень конфіденційності.

Одним із найбільш актуальних аспектів сучасного шифрування є забезпечення безпеки в розподілених системах та хмарних обчисленнях. Ці технології дозволяють зберігати і обробляти величезні обсяги даних, що вимагає застосування нових підходів до шифрування. Оскільки хмарні сервери можуть знаходитися в різних частинах світу, виникає потреба в ефективному захисті даних не тільки під час їх передачі, а й при їх зберіганні. Одним із таких підходів є шифрування на рівні файлів, яке передбачає зашифровку окремих файлів або документів перед їх зберіганням на серверах або в хмарних сховищах. Це дозволяє забезпечити конфіденційність інформації навіть у випадку несанкціонованого доступу до самих серверів. Важливою перевагою такого

підходу є те, що навіть якщо зловмисник отримає доступ до файлів, він не зможе прочитати їх вміст без наявності відповідного ключа для дешифрування.

Ще одним важливим аспектом шифрування в розподілених системах є захист баз даних. Оскільки в базах даних зберігається величезна кількість конфіденційної інформації, таких як особисті дані користувачів, фінансові дані та інша чутлива інформація, її захист має надзвичайно важливе значення. Для цього використовуються різні методи шифрування, які можуть стосуватися як цілих баз даних, так і окремих полів або рядків. Такий підхід дозволяє налаштувати рівень захисту для різних типів даних і забезпечити більш точну настройку конфіденційності.

Шифрування мережевих комунікацій є ще одним важливим аспектом у захисті інформації. Враховуючи, що багато важливих даних передаються через відкриті мережі, такі як Інтернет, необхідно забезпечити їх захист від перехоплення або зміни. Для цього використовуються різноманітні протоколи шифрування, зокрема SSL/TLS, які забезпечують захист даних під час їх передачі по мережі. Протоколи SSL і TLS дозволяють створювати зашифровані канали зв'язку між клієнтами та серверами, що є особливо важливим для забезпечення безпеки онлайн-транзакцій, інтернет-банкінгу та інших чутливих операцій.

Особливе значення має шифрування в контексті мобільних пристроїв і Інтернету речей (IoT). Оскільки ці пристрої мають обмежені обчислювальні ресурси та потребують економії енергії, важливо розробляти такі алгоритми шифрування, які б забезпечували високий рівень безпеки при мінімальних витратах на енергозабезпечення та обчислювальні ресурси. Оптимізація криптографічних алгоритмів для таких пристроїв дозволяє забезпечити достатній рівень захисту без значного впливу на продуктивність пристроїв.

Таким чином, сучасні тенденції в шифруванні даних включають розвиток складних і сильних криптографічних алгоритмів, таких як квантова криптографія, а також адаптацію до нових технологій, таких як хмарні обчислення та мобільні пристрої. Водночас, необхідність забезпечення високого рівня безпеки в умовах постійно зростаючих обчислювальних потужностей і нових загроз вимагає від криптографів постійного вдосконалення алгоритмів і пошуку нових рішень. Безпека даних, що передаються та зберігаються в цифровому середовищі, продовжує бути однією з найважливіших проблем сучасної інформаційної безпеки.

Загалом, сучасні підходи до шифрування даних демонструють постійний розвиток і адаптацію до нових вимог цифрового світу. Від квантових технологій до оптимізації криптографії для мобільних пристроїв і розподілених систем — усі ці зусилля спрямовані на забезпечення високого рівня безпеки, що є основою для безпечного обміну та зберігання інформації в умовах зростаючих кіберзагроз.

1.4 Основні методи шифрування інформації

Основні методи шифрування інформації є ключовими складовими сучасної криптографії і використовуються для забезпечення безпеки даних у цифровому середовищі. Шифрування є основним засобом захисту даних від несанкціонованого доступу, забезпечуючи конфіденційність, цілісність, а також автентичність інформації. Сучасні методи шифрування застосовуються в різних сферах: від комунікацій і мережевої безпеки до фінансових систем і зберігання персональних даних. У цьому розділі ми розглянемо різноманітні алгоритми та протоколи шифрування, їх принципи роботи, а також основні характеристики кожного методу.

Симетричне шифрування — це метод шифрування, при якому для шифрування і розшифрування повідомлень використовується один і той самий ключ. Важливим аспектом є безпека розподілу ключа між відправником і отримувачем. Саме для того, щоб уникнути перехоплення ключа, застосовуються різні протоколи обміну ключами. У випадку симетричного шифрування ключ, що використовується, повинен залишатися конфіденційним, оскільки його компрометація ставить під загрозу всю систему шифрування.

Відомі алгоритми симетричного шифрування:

— Advanced Encryption Standard (AES)

AES є одним з найбільш широко використовуваних і безпечних алгоритмів симетричного шифрування. Він був розроблений для заміни застарілого стандарту DES. AES підтримує три різні довжини ключа — 128, 192 та 256 біт, що дозволяє налаштувати рівень безпеки в залежності від потреб. Завдяки своїй швидкості, ефективності та високому рівню безпеки, AES застосовується в фінансових установах, урядових структурах і для захисту даних у хмарних середовищах.

Процес шифрування за допомогою AES включає кілька основних етапів, серед яких: субституція, перестановка, додавання раундів і операції XOR. Завдяки тому, що

AES працює з блоками даних розміром 128 біт, його ефективність дозволяє забезпечити безпеку навіть при великих обсягах даних.

— Data Encryption Standard (DES)

DES — один із найстаріших алгоритмів симетричного шифрування, який був стандартом у криптографії до початку 2000-х років. DES використовує 56-бітний ключ і блочне шифрування даних, розміром 64 біти. Хоча DES був популярним, його безпека з часом виявилася недостатньою, оскільки можливість застосування брутфорс-атак (перевірка всіх можливих ключів) швидко збільшилася з розвитком комп'ютерних технологій. В результаті, DES був замінений на більш потужні алгоритми, зокрема на AES.

— Triple DES (3DES)

3DES був розроблений як вдосконалена версія DES для підвищення рівня безпеки. Алгоритм здійснює три проходи шифрування для кожного блоку даних за допомогою трьох різних ключів. Хоча 3DES є більш безпечним за DES, він значно повільніший і менш ефективний порівняно з AES. Сьогодні 3DES вважається застарілим і більше не рекомендується для використання, хоча все ще застосовується в деяких фінансових і банківських системах.

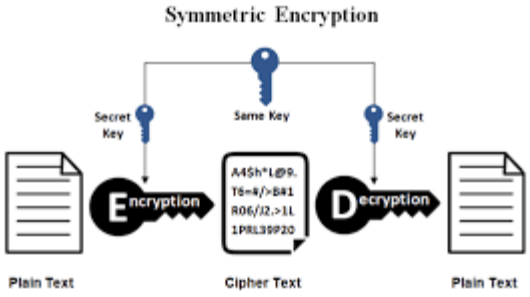

Рисунок 1.1 – Алгоритм Симетричного шифрування

Проблеми та обмеження симетричного шифрування

Основною проблемою симетричного шифрування є необхідність безпечного обміну ключами між учасниками обміну даними. Якщо ключ потрапить у руки зловмисника, вся система стає уразливою. Саме тому в поєднанні з симетричним шифруванням часто використовуються інші методи, наприклад, асиметричне шифрування, для безпечного обміну ключами.

Асиметричне шифрування — це метод, при якому для шифрування і розшифрування повідомлень використовуються два різні ключі: публічний і приватний.

Публічний ключ використовується для шифрування даних, а приватний — для їх розшифрування. Оскільки приватний ключ ніколи не передається по каналу зв'язку, цей метод дозволяє уникнути проблем з безпечним обміном ключами, характерних для симетричного шифрування.

Відомі алгоритми асиметричного шифрування:

— RSA (Rivest-Shamir-Adleman)

RSA є одним із найпоширеніших і відомих алгоритмів асиметричного шифрування. Алгоритм RSA заснований на математичній проблемі факторизації великих чисел і використовує пару ключів: публічний і приватний. Генерація ключів відбувається за допомогою двох великих простих чисел, добуток яких є частиною публічного ключа. Приватний ключ необхідний для розшифрування повідомлення. RSA дозволяє здійснювати шифрування і підписування даних, що є основою для багатьох протоколів безпеки, таких як SSL/TLS.

— Elliptic Curve Cryptography (ECC)

Криптографія на основі еліптичних кривих є сучасним методом шифрування, який використовує властивості еліптичних кривих для забезпечення безпеки. Оскільки для досягнення того ж рівня безпеки, що й у RSA, ECC використовує коротші ключі, це забезпечує значно більшу ефективність при збереженні високого рівня захисту. ECC є основою таких алгоритмів, як ECDSA (Elliptic Curve Digital Signature Algorithm) для підписів і ECDH (Elliptic Curve Diffie-Hellman) для обміну ключами.

— ElGamal

Алгоритм ElGamal є ще одним методом асиметричного шифрування, заснованим на математичній задачі обчислення дискретного логарифма. ElGamal застосовується для шифрування та підписів, забезпечуючи високий рівень безпеки, проте його алгоритм є менш ефективним у порівнянні з RSA і ECC.

Рисунок 1.2 – Алгоритм асиметричного шифрування

Переваги і недоліки асиметричного шифрування

Основною перевагою асиметричного шифрування є те, що публічний ключ може бути відкрито переданий будь-яким учасникам обміну даними, що значно спрощує процедуру розподілу ключів. Проте цей метод є повільнішим за симетричне шифрування через складність математичних операцій, що потребує значних обчислювальних ресурсів.

Хеш-функції — це математичні алгоритми, які перетворюють вхідні дані будь-якої довжини на хеш-код фіксованої довжини. Хеш-функції широко використовуються для забезпечення цілісності даних, а також у процесах аутентифікації. Важливими характеристиками хеш-функцій є:

Колізії: навіть незначні зміни вхідних даних повинні призводити до значних змін у хеш-коді, що дозволяє виявляти будь-які зміни в даних.

Односторонність: неможливо відновити вхідні дані з хеш-коду.

Швидкість: хеш-функції повинні бути швидкими у обчисленні.

Популярні хеш-функції:

— MD5 — один з найстаріших і найпоширеніших алгоритмів хешування, що створює хеш довжиною 128 біт. Попри свою популярність, MD5 не є безпечним, оскільки була виявлена вразливість до колізій, що робить його непридатним для сучасних криптографічних застосувань.

— SHA-1 був розроблений для усунення вразливостей MD5. Він створює хеш довжиною 160 біт і довгий час використовувався в криптографії та цифрових підписах. Однак з розвитком методів криптоаналізу були виявлені вразливості до колізій і зламів, що призвело до рекомендацій щодо його відмови.

— SHA-256 і SHA-512 є частинами родини алгоритмів SHA-2, які значно безпечніші за своїх попередників. SHA-256 створює хеш довжиною 256 біт, а SHA-512 — 512 біт. Вони активно використовуються для перевірки цілісності даних у блокчейнах, сертифікатах SSL/TLS та інших системах.

Еліптичні криві (ECC) є основою для шифрування, яке забезпечує високий рівень безпеки при використанні коротших ключів. Алгоритми на основі еліптичних кривих дають змогу створювати системи шифрування, які ефективні з точки зору швидкості і ресурсів, але при цьому забезпечують рівень безпеки, порівнянний з іншими методами, які використовують значно більші ключі.

Відомі алгоритми на основі еліптичних кривих:

— ECDSA (Elliptic Curve Digital Signature Algorithm)

ECDSA є одним з основних алгоритмів для цифрових підписів, заснованих на еліптичних кривих. Він забезпечує високий рівень безпеки при використанні коротших ключів, що робить його особливо популярним для мобільних пристроїв і обмежених ресурсів.

— ECDH (Elliptic Curve Diffie-Hellman)

ECDH є алгоритмом для безпечного обміну ключами, що дозволяє двом сторонам створити спільний секретний ключ без необхідності передавати його по каналу зв'язку. Цей метод широко застосовується у VPN, HTTPS і інших протоколах для безпечного обміну даними.

— ECIES (Elliptic Curve Integrated Encryption Scheme)

ECIES — це схема шифрування, яка об'єднує шифрування та обмін ключами на основі еліптичних кривих для забезпечення конфіденційності даних. Вона є однією з найбільш ефективних методів для забезпечення безпеки в мобільних і IoT пристроях, де ресурси обмежені.

Кожен з методів шифрування має свої особливості, переваги та обмеження. Вибір конкретного методу залежить від вимог до безпеки, ефективності та ресурсів, доступних у системі. Сучасні криптографічні системи комбінують різні підходи для досягнення максимальної безпеки, використовуючи як симетричне, так і асиметричне шифрування, а також хеш-функції для забезпечення цілісності даних. Технології криптографії продовжують розвиватися, і нові методи шифрування забезпечують ще більш високий рівень захисту в цифровому середовищі.

1.5 Сучасні фреймворки та технології для розробки веб-додатків

Розробка веб-додатків сьогодні є однією з найбільш динамічно розвиваючихся галузей в інформаційних технологіях. Оскільки вимоги до функціональності, продуктивності та масштабованості веб-додатків зростають, розробники дедалі частіше звертаються до сучасних фреймворків та технологій, які спрощують процес розробки і підтримки складних систем. Вибір правильного фреймворку для проекту має велике значення, оскільки він визначає ефективність розробки, адаптивність до змін та подальшу підтримку системи.

Сучасні фреймворки, серед яких найбільш відомими є React, Angular, Vue.js, надають розробникам не тільки готову основу для побудови додатків, але й цілу екосистему для створення складних, багатофункціональних рішень, що відповідають високим вимогам користувачів. Вибір конкретного фреймворку залежить від багатьох факторів, таких як складність проекту, вимоги до масштабованості, інтеграції з іншими системами, а також рівень команди розробників.

Одним із важливих аспектів, які визначають ефективність фреймворків, є підтримка компонентної архітектури. У сучасному розробці веб-додатків, особливо з використанням таких технологій, як React, компоненти дозволяють розбивати додаток на окремі, незалежні частини, що можуть бути повторно використані в інших частинах проекту. Це не тільки спрощує підтримку та масштабування додатка, але й дозволяє оптимізувати розробку за рахунок повторного використання компонентів.

React — це бібліотека для створення інтерфейсів користувача, яка стала однією з найбільш популярних серед розробників завдяки своїй ефективності та гнучкості. В основі React лежить концепція віртуального DOM, який дозволяє мінімізувати кількість змін, що вносяться в реальний DOM, що суттєво пришвидшує оновлення сторінок. Завдяки цьому веб-додатки, побудовані на React, працюють дуже швидко навіть за умови великих обсягів даних. Крім того, React дозволяє створювати динамічні компоненти, що автоматично адаптуються до змін у структурі даних, що є критично важливим у сучасних додатках, де дані можуть постійно змінюватися.

Однією з ключових переваг React є наявність потужної екосистеми додаткових бібліотек, таких як Redux, які допомагають організувати управління станом додатка. Це дозволяє розробникам легко створювати складні додатки з великою кількістю змінюваних даних, не турбуючись про складні проблеми з синхронізацією стану між різними компонентами. Крім того, React надає можливості для серверного рендерингу, що покращує продуктивність і SEO-показники веб-додатків.

Окрім React, існують інші популярні фреймворки, які також мають свої унікальні переваги та специфікації, що дозволяють ефективно розробляти різні типи веб-додатків.

Angular, розроблений компанією Google, є одним з найбільш потужних фреймворків для створення складних корпоративних додатків. Він відрізняється строгою архітектурною організацією, яка забезпечує структурованість і передбачуваність коду навіть в умовах великих команд розробників. Однією з основних

особливостей Angular є двосторонній зв'язок даних, що дозволяє автоматично синхронізувати змінені дані в інтерфейсі користувача з даними на сервері без необхідності додаткових маніпуляцій з боку розробника. Це суттєво полегшує процес розробки додатків, що потребують високої інтерактивності та складної логіки.

Ще однією важливою перевагою Angular є використання TypeScript, що є надмножиною JavaScript і дозволяє забезпечити типізацію даних. Це критично важливо для великих проектів, оскільки статичне типізування дозволяє знизити кількість помилок у коді, що часто виникають при роботі з JavaScript. Angular також має потужну систему модулів, яка дозволяє ефективно організувати код, зменшити його складність і забезпечити масштабованість додатка.

Vue.js поєднує найкраще з обох попередніх фреймворків, надаючи простоту використання, схожу на React, і при цьому можливість створення більш структурованих додатків, властиву Angular. Однією з головних переваг Vue є низький поріг входження для новачків у веб-розробці, що дозволяє швидко освоїти основи і розпочати створення проектів. У той же час Vue забезпечує достатню гнучкість для великих додатків завдяки своїй реактивній моделі управління даними. Vue не вимагає суворого дотримання певної архітектурної схеми, що дозволяє розробникам обирати необхідні функції та налаштування без зайвих обмежень.

Із розвитком серверних технологій набувають популярності Node.js та Express.js, які дозволяють виконувати JavaScript код на сервері. Це створює можливість розробки повноцінних веб-додатків за допомогою однієї мови програмування на обох сторонах (клієнтській та серверній), що значно спрощує процес розробки і підвищує ефективність. Express.js, будучи мінімалістичним, але потужним фреймворком для Node.js, надає все необхідне для створення веб-серверів та API. Він дозволяє з легкістю організовувати маршрутизацію запитів, обробку HTTP-запитів та взаємодію з базами даних.

Сучасні веб-додатки також активно використовують GraphQL — альтернативу традиційним REST API. GraphQL дозволяє створювати більш гнучкі та ефективні запити до серверів, зокрема, дає можливість клієнтам запитувати лише ті дані, які їм дійсно необхідні, що значно зменшує навантаження на сервер і мережу. Це особливо важливо для великих додатків із складною структурою даних, де стандартний REST API може стати неефективним.

Відзначимо також зростаючу популярність серверлес-архітектур, де розробники можуть зосередитися на написанні функцій і не турбуватися про управління інфраструктурою. Вони дозволяють запускати функції в хмарі, сплачуючи лише за час їх виконання, що робить серверлес-підхід економічно вигідним і зручним для масштабування додатків.

Таким чином, сучасні фреймворки та технології для розробки веб-додатків значно покращують продуктивність, масштабованість та адаптивність додатків. Завдяки інтеграції з новими інструментами для управління станом, API та серверними технологіями, процес розробки стає все більш ефективним. Це дозволяє створювати веб-додатки, що відповідають високим вимогам користувачів, забезпечують стабільну роботу і готові до інтеграції з іншими системами. Вибір фреймворку і технологій для розробки веб-додатків має вирішальне значення для успіху проекту і залежить від цілей, які ставить перед собою команда розробників, а також від специфіки самого проекту.

2 ОБГРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ВЕБ-ДОДАТКУ

2.1 Аналіз потреб і вимог користувачів

Аналіз потреб і вимог користувачів у галузі шифрування інформації є одним із найважливіших етапів при розробці криптографічних рішень, оскільки саме від розуміння цих потреб залежить безпека та ефективність майбутньої системи. Шифрування забезпечує конфіденційність, цілісність даних, а також захист від несанкціонованого доступу. Тому правильне визначення потреб користувачів має велике значення, адже на основі цього вибираються криптографічні методи, які здатні забезпечити належний рівень захисту.

У сучасному цифровому середовищі, де кіберзагрози постійно зростають, створення надійних і ефективних систем шифрування є пріоритетним завданням для розробників. Водночас системи мають бути інтуїтивно зрозумілими та зручними для користувачів, адже забезпечення безпеки не повинно обмежувати зручність використання. Зрозуміло, що користувачі повинні мати не тільки доступ до потужних алгоритмів шифрування, але й отримувати прості та зрозумілі інтерфейси для реалізації цих процесів.

Одним із перших і найважливіших аспектів є виявлення специфічних сценаріїв використання шифрування. Це означає, що кожен тип користувача або організація має свої унікальні вимоги до шифрування, в залежності від сфери діяльності. Наприклад, банки повинні застосовувати найсучасніші методи шифрування, адже вони працюють з дуже чутливими фінансовими даними. Для них основним пріоритетом є забезпечення захисту фінансових транзакцій, що потребує використання надійних та перевірених алгоритмів, здатних витримувати складні кіберзагрози. З іншого боку, в медичній сфері шифрування повинно захищати особисті медичні записи пацієнтів, що вимагає ще більших стандартів безпеки, зокрема щодо захисту інформації від несанкціонованого доступу. Для цього необхідно вибирати алгоритми, що відповідають вимогам галузевих стандартів, таких як HIPAA у США або GDPR в Європі, які регламентують захист персональних даних.

Крім того, у сучасних умовах слід враховувати постійно змінювані виклики у галузі інформаційної безпеки. Наприклад, зростаюча загроза квантових комп'ютерів вимагає постійного вдосконалення алгоритмів шифрування, щоб гарантувати їхню

ефективність навіть у майбутньому. Тому розробка нових криптографічних методів, які здатні протистояти таким новим загрозам, стає необхідністю для забезпечення довгострокового захисту інформації. Важливою є і здатність системи шифрування швидко адаптуватися до змін, щоб у разі появи нових загроз не виникало серйозних вразливих місць.

Ще одним критичним аспектом є забезпечення належного балансу між безпекою та зручністю використання системи. Користувачі не тільки вимагають високої надійності при передачі чутливих даних, але й очікують, що система буде легкою у використанні. Оскільки для багатьох користувачів шифрування може бути складним і незнайомим процесом, інтерфейс повинен бути інтуїтивно зрозумілим і простим для взаємодії. Водночас, система повинна не лише відповідати високим стандартам безпеки, але й забезпечувати ефективність у виконанні операцій шифрування. Наприклад, криптографічні рішення повинні працювати швидко, навіть якщо передаються великі обсяги даних, і не створювати зайвих затримок, що може призвести до зниження ефективності робочих процесів.

Особливо важливим є врахування регулювання та дотримання стандартів безпеки. Відповідність сучасним юридичним вимогам, таким як GDPR в Європейському Союзі або HIPAA в США, є необхідною для організацій, що працюють з персональними даними. Криптографічні рішення, зокрема ті, що застосовуються у медичних і фінансових сферах, повинні відповідати вимогам щодо захисту персональних даних і гарантувати їхню конфіденційність. Це вимагає від розробників не лише застосування надійних шифрувальних алгоритмів, але й уважності до змін в законодавстві, а також інтеграції відповідних механізмів безпеки в систему.

У процесі розробки веб-додатку для шифрування інформації необхідно врахувати кілька важливих вимог, які дозволять забезпечити ефективну та надійну роботу системи. Однією з основних вимог є правильний вибір хостингу для веб-додатку. Для забезпечення стабільної роботи системи важливо обрати надійного хостинг-провайдера, який гарантує високу доступність та пропускну здатність. Завдяки цьому користувачі зможуть здійснювати доступ до системи з будь-якої точки світу без ризику затримок або відмов у роботі.

Ще однією важливою вимогою є авторизація користувачів. Для захисту чутливих даних система повинна мати надійний механізм авторизації. Це може включати

створення унікальних облікових записів для кожного користувача, а також використання сучасних методів захисту, таких як двофакторна аутентифікація. Це дозволить значно підвищити рівень безпеки та знизити ризики, пов'язані з несанкціонованим доступом до системи.

Важливим аспектом є також збереження історії зашифрованих повідомлень. Для того, щоб забезпечити прозорість операцій, кожне зашифроване повідомлення має зберігатися в базі даних з відповідними мітками часу та інформацією про користувача, який здійснив шифрування. Така функція дозволить здійснювати аудит і контролювати дії користувачів системи.

Врахування всіх цих аспектів дозволить створити надійну і зручну систему шифрування, що буде відповідати сучасним вимогам безпеки та юридичним стандартам. Система повинна бути не тільки захищеною, але й масштабованою, здатною адаптуватися до змінюваних умов і нових загроз у цифровому середовищі.

2.2 Вибір середовища розробки

При створенні веб-додатку було використано мови HTML, CSS та JavaScript. Ці технології забезпечили як функціональність, так і привабливий зовнішній вигляд додатку, а також дозволили створити інтуїтивно зрозумілий інтерфейс для взаємодії з користувачами. Кожна з цих мов виконує унікальні завдання, що робить їх поєднання оптимальним для веб-розробки. У цьому розділі ми розглянемо ключові аспекти та переваги застосування кожної мови, а також їх вплив на створення повноцінного веб-додатку.

HTML (HyperText Markup Language) є основою веб-розробки, адже саме ця мова визначає структуру та організацію контенту веб-сторінок. Завдяки стандарту HTML, який підтримується всіма сучасними браузерами, розробники можуть бути впевнені, що їхній веб-додаток відображатиметься коректно на різних пристроях і в різних середовищах. Простота використання HTML є його вагомою перевагою, оскільки синтаксис цієї мови є зрозумілим і легким для вивчення навіть для початківців. Теги HTML дають змогу чітко визначити роль кожного елемента на сторінці, сприяючи створенню чіткої та логічної структури. Семантична природа HTML, яка передбачає використання спеціалізованих тегів, таких як <header>, <section>, <footer>, значно покращує не лише доступність для людей із порушеннями зору, але й оптимізацію для

пошукових систем. Завдяки цим можливостям, HTML не лише формує каркас веб-додатків, а й забезпечує їхню відповідність сучасним стандартам розробки.

CSS (Cascading Style Sheets) забезпечує стиль та естетику веб-сторінок, що є надзвичайно важливим для залучення уваги користувачів. Завдяки цій мові можливо розділити структурну та візуальну складові сторінки, що полегшує її підтримку та оновлення. CSS пропонує широкі можливості для налаштування зовнішнього вигляду, дозволяючи змінювати кольори, шрифти, розміри, розташування елементів та багато іншого. Одним із ключових аспектів сучасної веб-розробки є адаптивний дизайн, що забезпечується за допомогою CSS. Використовуючи медіа-запити, розробники можуть створювати сторінки, які автоматично підлаштовуються під розміри екранів різних пристроїв, що робить веб-додатки зручними для користувачів як на мобільних телефонах, так і на великих моніторах. Додатково, CSS дозволяє створювати анімації та переходи, які додають динамічності сторінкам, роблячи їх візуально привабливішими. У поєднанні з підтримкою більшістю браузерів, CSS стає універсальним інструментом для забезпечення якісного дизайну.

JavaScript є однією з основних мов програмування, яка надає веб-додаткам високу динамічність і інтерактивність. З самого початку веб-розробки JavaScript отримав популярність завдяки своїй здатності працювати безпосередньо в браузері, що дозволяє створювати веб-додатки, здатні миттєво реагувати на дії користувачів без перезавантаження сторінки.

Однією з головних причин, чому JavaScript вибрано для веб-додатків, є його можливість маніпулювати Document Object Model (DOM), який є репрезентацією HTML-документу в пам'яті браузера. DOM дає можливість змінювати HTML-елементи в реальному часі, що дозволяє створювати інтерфейси, які безпосередньо реагують на користувацькі взаємодії, такі як натискання кнопок, введення даних у форми чи прокручування сторінки.

Завдяки JavaScript веб-додатки можуть бути значно динамічнішими. Наприклад, JavaScript дозволяє створювати інтерактивні елементи на сторінці: від простих анімацій і зміни стилів до складних інтерактивних віджетів, як-от календарі, фільтри для пошуку та динамічні меню. Усі ці функції реалізуються без необхідності перезавантажувати сторінку, що дозволяє підтримувати високий рівень зручності для користувача.

Ще одним важливим аспектом є асинхронне програмування в JavaScript, яке дає змогу виконувати операції у фоновому режимі без блокування основного потоку додатка. Наприклад, з допомогою технології AJAX або більш нових методів, таких як fetch API, можна здійснювати запити до сервера та отримувати або надсилати дані без перезавантаження сторінки. Це дозволяє створювати більш чуйні додатки, що значно підвищує зручність і швидкість роботи з веб-додатком, оскільки користувач не помічає затримок, коли обробляються запити.

Обробка подій також є однією з ключових можливостей JavaScript, що робить веб-додатки інтерактивними. JavaScript дозволяє визначати обробники подій для різних взаємодій користувача, таких як натискання кнопок, введення даних у поля форм, переміщення курсору по екрану та інші дії. Це дозволяє створювати інтерактивні елементи, які змінюються в залежності від взаємодії з користувачем.

Для створення більш складних додатків, JavaScript підтримує концепцію модульності, що дозволяє організувати код у незалежні блоки, або модулі, що виконують конкретні завдання. Це значно полегшує підтримку і масштабування додатків, адже кожен модуль можна розвивати та тестувати окремо. Крім того, за допомогою таких популярних фреймворків, як React, Vue.js, або Angular, можна створювати ще більш складні і масштабовані додатки, де код поділяється на компоненти, що дозволяє розробникам ефективніше працювати над великими проектами.

Нарешті, важливо зазначити, що JavaScript активно підтримується всіма сучасними веб-браузерами, що забезпечує сумісність між різними платформами та пристроями. Більше того, більшість веб-браузерів мають вбудовані інструменти для відладки коду JavaScript, що значно полегшує процес розробки.

Таким чином, JavaScript є не лише основною мовою програмування для інтерактивних веб-додатків, але й незамінним інструментом для забезпечення швидкої, динамічної і зручної взаємодії з користувачем. Його багатофункціональність, можливості для інтеграції з іншими сервісами та потужна підтримка спільноти роблять JavaScript найкращим вибором для створення сучасних веб-додатків.

Застосування цих мов у поєднанні дозволяє створювати повноцінні веб-додатки, які відповідають сучасним стандартам і очікуванням користувачів. HTML визначає структуру сторінки, CSS додає стилізацію та дизайн, а JavaScript забезпечує

інтерактивність та динамічність. Т акий підхід гарантує створення продуктів, які є не лише функціональними, але й зручними та привабливими. Завдяки цьому веб-додатки стають ефективними інструментами для досягнення різноманітних бізнес-цілей та покращення користувацького досвіду.

2.4 Спосіб реалізації бекенду

Для реалізації бекенду веб-додатків існує кілька варіантів, кожен з яких має свої особливості та переваги залежно від вимог проекту. Один з найпоширеніших способів — це використання традиційних серверних технологій, таких як Node.js, Django або Ruby on Rails. Ці підходи дозволяють розробляти сервери та API з нуля, даючи розробнику повний контроль над усіма аспектами роботи додатку. Node.js, зокрема, користується популярністю завдяки своїй асинхронній природі, що дозволяє обробляти велику кількість одночасних запитів без значних затримок. Такі фреймворки як Express.js допомагають швидко налаштувати сервер і створювати API, а також дозволяють використовувати JavaScript на обох сторонах — як на клієнтській, так і на серверній.

Однак цей підхід вимагає більше часу та зусиль для налаштування і підтримки. Розробникам необхідно буде враховувати безліч нюансів, таких як керування сесіями, аутентифікація користувачів, безпека даних, масштабування тощо. Крім того, традиційне використання таких технологій може вимагати наявності власної інфраструктури або серверів, що може стати додатковим викликом для стартапів або невеликих команд.

Іншим варіантом є використання так званих серверлес-платформ, таких як AWS Lambda, Google Cloud Functions або Firebase. Цей підхід значно спрощує управління бекендом, оскільки дозволяє розробникам не турбуватися про інфраструктуру і зосередитись на написанні функцій. Використання серверлес-технологій дозволяє знижувати витрати на хостинг і зменшувати час на масштабування, оскільки ці платформи автоматично обробляють зміну навантаження. Вони також можуть забезпечити високу доступність і швидке реагування на запити користувачів без необхідності постійного моніторингу серверів.

Однак серверлес також має свої обмеження, зокрема у вигляді потенційних складнощів з підтримкою складних процесів або великих обсягів даних, що вимагають

постійного зберігання. Для малих проектів або стартапів серверлес може бути чудовим рішенням, але для великих і складних проектів з великою кількістю взаємодій це може виявитися не таким ідеальним.

Ще одним варіантом є використання традиційних баз даних і серверів з автоматизованим управлінням, таких як Amazon RDS, Google Cloud SQL або Heroku. Ці платформи пропонують бази даних як сервіс (DBaaS), що дозволяє зосередитись на розробці додатку, не турбуючись про конфігурацію і обслуговування серверів. Вони також підтримують масштабування, забезпечують резервне копіювання і високу доступність, що робить їх привабливими для більших проектів.

Вибір технології для бекенду значною мірою залежить від вимог проекту і команди. Для невеликих проектів, де важливо швидко реалізувати рішення і мінімізувати витрати, серверлес-платформи можуть стати ідеальним вибором. Для більших проектів, які потребують більш гнучкого і масштабованого рішення, традиційні сервери та бази даних з автоматизованим управлінням можуть бути кращими варіантами.

У нашому проекті ми обрали Firebase, оскільки ця платформа забезпечує все необхідне для реалізації високоякісного бекенду без потреби в обслуговуванні власних серверів і баз даних. Firebase пропонує цілий набір інструментів, таких як аутентифікація користувачів, реальна база даних для зберігання даних, а також підтримка хмарного хостингу. Це дозволяє нам зосередитись на розробці функціоналу, а не на технічних аспектах інфраструктури. Крім того, Firebase легко масштабувати, що робить його ідеальним рішенням для стартапів, які мають невеликий бюджет, але хочуть створити надійне і швидке рішення.

Таким чином, ми обрали Firebase не лише через його простоту у використанні, але й завдяки наявності великої кількості готових інструментів для роботи з даними та аутентифікацією, що дозволяє зменшити час розробки і знизити витрати на обслуговування.

2.5 Вибір бібліотеки для оформлення дизайну сайту

Існує багато CSS-бібліотек, які широко використовуються розробниками для створення веб-сайтів і забезпечення зручного та адаптивного дизайну. Однак серед них є кілька особливо популярних і ефективних. Розглянемо кілька з них та коротко

проаналізуємо їхні особливості:

Bootstrap — одна з найпоширеніших і найбільш затребуваних CSS-бібліотек у світі. Вона надає великий набір готових компонентів і стилів, що дозволяє швидко створювати професійний веб-дизайн. Ключова перевага Bootstrap полягає в тому, що ця бібліотека має адаптивний дизайн, що дозволяє веб-сайтам автоматично адаптуватися під різні пристрої й екрани. Крім того, спільнота розробників і багата документація значно спрощують процес освоєння бібліотеки, що робить її ідеальним вибором як для новачків, так і для досвідчених спеціалістів.

Foundation — ще одна популярна CSS-бібліотека, яка також має великий набір компонентів та стилів. Вона забезпечує більшу гнучкість у налаштуваннях і пропонує широкі можливості для кастомізації під індивідуальні потреби. Завдяки своєму адаптивному дизайну та підтримці різних браузерів, Foundation є хорошим вибором для розробників, які потребують більше контролю над дизайном та функціональністю веб-сайту.

Bulma — це легка і проста в освоєнні бібліотека, яка ідеально підходить для тих, хто хоче швидко розпочати проект. Вона надає простий у використанні набір компонентів і стилів, при цьому залишаючи достатньо простору для творчості. Bulma ідеально підходить для створення невеликих веб-сайтів і лендінгів, де важливі швидкість та зручність розробки.

Semantic UI — відрізняється від інших бібліотек завдяки своєму інтуїтивно зрозумілому підходу до класифікації та іменування класів. Бібліотека надає безліч компонентів, які можна легко інтегрувати у проекти, а також має вбудовану підтримку адаптивного дизайну. Це робить Semantic UI хорошим вибором для розробників, які хочуть забезпечити зручність у використанні та організації коду.

Хоча всі ці бібліотеки мають свої переваги, Bootstrap є однією з найкращих для створення веб-сайтів завдяки кільком ключовим факторам:

— готовий набір компонентів: Bootstrap надає великий набір готових компонентів, таких як кнопки, форми, каруселі, модальні вікна та багато іншого. Всі ці компоненти мають сучасний та чистий дизайн, що дозволяє значно скоротити час на розробку та забезпечити єдність стилю на всьому сайті. Це означає, що розробники можуть зосередитись на основній логіці сайту, а не на розробці кожного елемента з нуля;

— адаптивний дизайн: З вбудованою підтримкою адаптивного дизайну Bootstrap

дозволяє створювати сайти, які добре виглядатимуть на різних пристроях — від мобільних телефонів до великих моніторів. Компоненти автоматично підлаштовуються під розмір екрана, що забезпечує зручний користувацький досвід;

— гнучкість і налаштування: Bootstrap пропонує широкі можливості для налаштування компонентів. За допомогою простих класів можна змінювати кольори, розміри, шрифти, відступи та інші параметри. Це дозволяє розробникам легко адаптувати бібліотеку під конкретні вимоги проекту або бренду;

— кросс-браузерна сумісність: Bootstrap забезпечує безперебійну роботу на основних браузерах, таких як Chrome, Firefox, Safari, Edge, що мінімізує ризики виникнення проблем із сумісністю. Це дозволяє створити універсальний дизайн, який працює стабільно на різних платформах;

— спільнота та документація: Одна з головних переваг Bootstrap — це велика спільнота розробників, які активно вносять покращення, підтримують один одного і розв’язують проблеми. Крім того, бібліотека має дуже детальну документацію, яка надає чіткі приклади використання всіх компонентів і дозволяє швидко освоїти навіть новачкам;

— швидкість розробки: Використання готових рішень для створення інтерфейсу значно скорочує час розробки. Не потрібно створювати стилі з нуля або шукати інші CSS-бібліотеки. Bootstrap надає весь необхідний інструментарій для розробки адаптивного та красивого дизайну, що дозволяє зосередитись на логіці додатку.

В цілому, Bootstrap залишається одним з найкращих варіантів для розробки сучасних веб-сайтів завдяки своїм широким можливостям, адаптивності, кросс-браузерній сумісності та зручності використання. Ця бібліотека дозволяє створювати високоякісні веб-дизайни, які виглядають однаково добре на різних пристроях і зменшують час, необхідний для розробки.

3 ТЕОРІЯ ТА ВИБІР МЕТОДУ ШИФРУВАННЯ

3.1 Теорія синхронного і асинхронного методів шифрування

Синхронні методи шифрування, також відомі як симетричні, є одними з найстаріших і найбільш широко використовуваних способів захисту даних. Принцип їх роботи полягає у використанні одного і того ж секретного ключа як для шифрування, так і для розшифрування інформації. Це означає, що як відправник, так і отримувач мають однаковий ключ, що дозволяє їм взаємодіяти з даними без сторонніх втручань. Проте для того, щоб ця система працювала належним чином, необхідно забезпечити абсолютну безпеку самого ключа, оскільки доступ до нього дозволяє будь-якому зловмиснику розшифрувати дані.

Процес шифрування в рамках синхронного методу включає в себе застосування математичних алгоритмів, таких як AES (Advanced Encryption Standard) або DES (Data Encryption Standard), які змінюють оригінальні дані згідно з певними алгоритмічними операціями, що виконуються за допомогою секретного ключа. Ці алгоритми можуть бути досить складними, однак вся процедура шифрування проходить без необхідності у зовнішніх перевірках або використанні додаткових елементів, як у випадку з асиметричним шифруванням. Це робить синхронні методи досить швидкими і ефективними, що дозволяє їх використовувати в системах, де необхідна висока швидкість обробки великих обсягів даних.

Однак важливо пам'ятати, що основний недолік симетричних методів полягає в необхідності безпечного обміну ключем. Оскільки цей ключ має бути ідентичним у обох учасників процесу, передача його через незахищені канали може привести до витоку інформації. Тому в багатьох випадках для забезпечення додаткової безпеки застосовуються різноманітні протоколи для аутентифікації та перевірки ключів, а також використовуються механізми періодичної зміни ключа, щоб зменшити ризик його компрометації.

Крім того, синхронні методи шифрування відрізняються від асиметричних тим, що вимагають значно менших обчислювальних ресурсів. Це означає, що процес шифрування даних займає менше часу, а система може працювати з великими обсягами інформації без суттєвих затримок. Однак на практиці, якщо ключ зловмисником буде перехоплений, вся система шифрування може бути скомпрометована. Тому для захисту

даних зазвичай застосовуються додаткові рівні безпеки, такі як хешування або використання кількох ключів для різних частин системи.

Ще одним аспектом, на який варто звернути увагу, є масштабованість синхронного шифрування. У разі необхідності обробки великої кількості однотипних повідомлень, можна створити ефективніші рішення, які дозволяють працювати з великими масивами даних, підтримуючи високу швидкість обробки. Це робить синхронне шифрування ідеальним варіантом для таких застосувань, як фінансові транзакції, захист персональних даних або обмін конфіденційною інформацією між організаціями.

Таким чином, хоча синхронне шифрування має низку переваг, таких як швидкість і ефективність, воно також потребує серйозних заходів безпеки, зокрема на етапах передачі і зберігання секретного ключа. Тому його найчастіше використовують разом з іншими методами захисту або в тих випадках, коли необхідна швидка обробка даних, а ризик компрометації ключа є мінімальним.

Асиметричне або асинхронне шифрування є одним із основних методів захисту даних, який відрізняється від синхронного тим, що використовує два різних ключа — публічний і приватний. У цьому методі один з ключів, публічний, може бути відкрито розповсюджений серед усіх учасників, а інший, приватний, зберігається в таємниці. Публічний ключ використовується для шифрування інформації, тоді як приватний ключ служить для її розшифрування. Оскільки для шифрування і розшифрування використовуються різні ключі, це надає більш високий рівень безпеки, оскільки навіть якщо публічний ключ доступний всім, без приватного ключа отримати доступ до шифрованої інформації неможливо.

Основним принципом роботи асиметричного шифрування є забезпечення безпеки комунікацій через відкритий канал передачі даних, де кожен учасник має власну пару ключів. Тому, навіть якщо дані передаються через незахищену мережу, вони залишаються недоступними для третіх осіб. Усі ключі генеруються таким чином, що один з них не може бути обчислений або відновлений з іншого. Це робить асиметричне шифрування дуже надійним методом захисту інформації, оскільки немає необхідності передавати або зберігати секретні ключі на кожному з етапів передачі.

Один з основних елементів асиметричного шифрування — це алгоритм RSA (Rivest–Shamir–Adleman), який є найпоширенішим і найбільш використовуваним

методом. RSA ґрунтується на математичних обчисленнях, зокрема на складності факторизації великих простих чисел, що забезпечує високий рівень безпеки. Інші алгоритми, такі як ElGamal або ECC (Elliptic Curve Cryptography), також використовуються для забезпечення криптографічної безпеки в різних сценаріях.

Однією з головних переваг асиметричного шифрування є відсутність необхідності у безпечному каналі для передачі ключа. Оскільки публічний ключ може бути переданий відкрито, він не несе загрози компрометації даних. Це робить асиметричне шифрування ідеальним для систем, де є потреба в безпечному обміні даними через відкриті канали зв'язку, наприклад, у веб-комунікаціях або електронних платіжних системах. Водночас, сам процес шифрування та розшифрування є складнішим і займає більше часу, ніж у синхронному шифруванні, через більшу обчислювальну складність.

Однак головною проблемою асиметричного шифрування є його відносно низька швидкість порівняно з синхронним методом. Оскільки алгоритми, що використовуються для генерації і обробки великих ключів, є обчислювально інтенсивними, це призводить до значних затримок у процесі шифрування та розшифрування великих обсягів даних. Для вирішення цієї проблеми часто використовують гібридні методи шифрування, де асиметричне шифрування застосовується для безпечної передачі секретного ключа, а синхронне шифрування — для обміну даними.

Одним із найбільш важливих застосувань асиметричного шифрування є використання цифрових підписів. У цьому випадку приватний ключ використовується для підпису документа або повідомлення, тоді як публічний ключ дозволяє будь-кому перевірити, чи було повідомлення підписано саме тим, хто володіє відповідним приватним ключем. Це дозволяє гарантувати цілісність і автентичність даних у процесі їх передачі, що особливо важливо в юридичних та фінансових операціях.

Асиметричне шифрування також широко використовується в SSL/TLS протоколах для захисту веб-з'єднань. Під час установлення з'єднання між браузером і сервером використовуються публічні і приватні ключі для забезпечення безпечного обміну даними та шифрування інформації, що передається. Зокрема, під час встановлення з'єднання за допомогою HTTPS використовуються механізми асиметричного шифрування для захисту початкової передачі ключів, після чого

переходять до синхронного шифрування для обміну великими обсягами даних.

3.2 Обрунтування вибору методу шифрування AES

Вибір методу шифрування є важливим етапом у розробці системи безпеки для веб-додатків і програмного забезпечення загалом. Шифрування є основним інструментом для захисту даних від несанкціонованого доступу, тому його ефективність і безпека мають критичне значення. Серед різноманітних методів шифрування, таких як симетричні та асиметричні алгоритми, для конкретного проекту обрано AES (Advanced Encryption Standard). Цей вибір обґрунтовано рядом технічних та практичних причин, які зробили AES одним з найбільш оптимальних варіантів для сучасних застосувань. Переваги використання AES:

— швидкість та ефективність: AES є швидким та ефективним алгоритмом шифрування, що особливо важливо для забезпечення високої продуктивності в умовах обмежених ресурсів. Завдяки своїй простоті та оптимізації для апаратного забезпечення, AES забезпечує високу швидкість роботи навіть з великими обсягами даних;

— безпека: AES використовує ключі різної довжини — 128, 192 або 256 біт, що дозволяє вибрати оптимальний рівень безпеки залежно від вимог проекту. Завдяки сучасним методам криптоаналізу та високій стійкості до атак, AES є одним з найбільш надійних алгоритмів на сьогоднішній день;

— адаптованість до різних платформ: AES підтримується майже всіма сучасними платформами та бібліотеками, що робить його універсальним для використання в різноманітних програмних і апаратних системах;

— широка підтримка: AES є стандартом, прийнятим урядами та великими організаціями для захисту даних, що підтверджує його надійність і довіру до нього в глобальному масштабі;

— підтримка апаратного шифрування: AES підтримує апаратне прискорення, що дозволяє значно підвищити продуктивність шифрування і розшифрування на спеціалізованому обладнанні, знижуючи навантаження на процесор.

У нашому випадку обрання AES для шифрування даних обумовлене наступними факторами:

— вимоги до безпеки: Для забезпечення надійного захисту даних у нашій системі

використовуються ключі великої довжини (256 біт), що дає високий рівень стійкості до атаки методом підбору або криптоаналізу;

— швидкість виконання: Оскільки наш додаток потребує обробки великих обсягів даних в реальному часі, ефективність AES, яка забезпечує швидке шифрування, є критично важливою;

— підтримка стандартів: Вибір AES дозволяє нам слідувати міжнародним стандартам безпеки, що є важливим при роботі з чутливими даними, такими як персональні або фінансові дані.

Для більш детального розуміння вибору AES як симетричного методу шифрування, доцільно порівняти синхронні та асинхронні методи шифрування. Нижче представлено таблицю (табл. 3.1), яка допомагає виявити ключові відмінності між ними.

Таблиця 3.1 – Порівняння синхронних та асинхронних методів шифрування

Характеристика	Синхронне шифрування (AES)	Асиметричне шифрування (RSA, ECC)
Тип ключа	Один спільний ключ для шифрування і розшифрування	Пара ключів: публічний та приватний
Швидкість	Висока, особливо при шифруванні великих обсягів даних	Низька, оскільки обчислення складніші
Безпека	Висока, але залежить від захисту ключа	Висока, особливо при передачі через незахищені канали
Використання в реальному часі	Ідеально підходить для високопродуктивних систем	Зазвичай не використовується для реального часу
Захист ключів	Потрібно забезпечити безпеку ключа в процесі передачі та зберігання	Публічний ключ можна передавати відкрито, приватний ключ зберігається секретно
Підтримка апаратного шифрування	Підтримує апаратне шифрування для прискорення	Зазвичай не підтримується апаратним шифруванням
Складність реалізації	Простота в реалізації і використанні	Складніша реалізація, вимагає більше ресурсів
Основні сфери використання	Шифрування великих обсягів даних, VPN, шифрування файлів	Захист переданої інформації через відкриті канали, електронні підписи

AES був обраний для даного проекту з огляду на його високу ефективність, швидкість шифрування, високий рівень безпеки та універсальність у використанні. У порівнянні з асиметричними методами шифрування, такими як RSA, AES має значні переваги в контексті швидкості обробки великих обсягів даних, що критично важливо

для сучасних веб-додатків і систем. Водночас, використання AES у поєднанні з іншими методами, такими як асиметричне шифрування для обміну ключами, може забезпечити найкраще поєднання швидкості та безпеки.

3.3 Принцип роботи методу AES

Алгоритм AES (Advanced Encryption Standard) є симетричним блоковим шифром, який функціонує за принципом поділу даних на блоки фіксованого розміру (128 біт) і обробки їх через серію математичних операцій. Кожна операція виконується з використанням криптографічного ключа, розмір якого може бути 128, 192 або 256 бітів. Кількість раундів, через які проходять дані, залежить від розміру ключа: 10 раундів для ключа 128 біт, 12 для 192 і 14 для 256 бітів.

Основна ідея AES полягає у багаторазовому застосуванні серії перетворень, спрямованих на створення шифротексту, що максимально відрізняється від вихідного тексту навіть за мінімальних змін у вхідних даних.

Дані, які підлягають шифруванню, представлені у вигляді блоку розміром 4×4 байтів, що називається State. Наприклад, якщо вхідний текст — це 128 бітів (16 байтів), то він заповнює матрицю

Ключ також перетворюється у схожу матрицю, яку використовують у кожному раунді.

Перед початком основних раундів дані у матриці StateState модифікуються через операцію XOR із початковим ключем. Це описується формулою:

$$State' = State \oplus Key \tag{3.1}$$

де $\oplus$ — побітова операція XOR. Цей етап забезпечує базове маскування даних і робить їх готовими до подальших перетворень.

Основні операції в раундах

1. Підстановка байтів (SubBytes) На цьому етапі кожен байт матриці StateState замінюється на нове значення за допомогою таблиці підстановок (S-box). S-box побудована на основі обернених елементів у полі Галуа GF(2^8), що забезпечує нелінійність перетворення. Якщо, наприклад, байт si,js_{i,j} у шестнадцятковому вигляді дорівнює 3C, то його нове значення в S-box може бути 62.

2. Зсув рядків (ShiftRows):Рядки матриці зсуваються вліво на різну кількість позицій. Цей зсув забезпечує перемішування байтів у межах блоку, посилюючи

дифузію даних. Формально:

$$si,j′=si,(j+i)mod4 \quad (3.2)$$

де ii — номер рядка, jj — номер стовпця.

3. Змішування стовпців (MixColumns) Стовпці матриці обробляються як поліноми, які множаться на незмінний поліном C(x)C(x) у полі Галуа:

$$C(x)=03x^3+01x^2+01x+02 \quad (3.3)$$

Ця операція забезпечує розсіювання впливу кожного байта на весь блок. Нове значення байтів у стовпці розраховується за допомогою матричного множення.

Операція виконується в полі GF(2^8), що передбачає додавання і множення з модулем $x^8+x^4+x^3+x+1$.

4. Додавання раундового ключа (AddRoundKey) Ця операція аналогічна до початкового XOR, але використовує ключ, специфічний для поточного раунду.

У фінальному раунді операція MixColumns пропускається, щоб зберегти остаточну структуру шифротексту. В результаті дані мають вигляд, кардинально відмінний від початкового тексту.

Загальний принцип шифрування у AES можна виразити так:

$$State_{final}=Round(State_{initial}\oplus Key_0,Key_1,\ldots,Key_N) \quad (3.4)$$

де N — кількість раундів. Усі проміжні обчислення повторюються за встановленою схемою, що гарантує високий рівень захисту.

Математичні переваги AES

AES забезпечує високий рівень криптографічної стійкості завдяки поєднанню нелінійних і лінійних перетворень, що ускладнюють аналіз за допомогою сучасних криптоаналітичних методів. Алгоритм зберігає ефективність навіть для великих обсягів даних і складних ключів, залишаючись стандартом безпеки у світі шифрування.

3.4 Реалізація шифрування даних у веб-додатках

Реалізація шифрування даних у веб-додатку

У сучасних веб-додатках забезпечення безпеки даних користувачів є критично важливим завданням. Для цього активно застосовуються криптографічні методи, такі як AES, які можна реалізувати різними способами. Ми зупинили свій вибір на використанні бібліотеки CryptoJS, але перед цим варто розглянути можливі альтернативи та пояснити наш підхід.

Одним із популярних підходів є використання стандартного Web Crypto API, який

вбудований у більшість сучасних браузерів. Цей метод забезпечує високий рівень захисту завдяки зберіганню ключів у безпечній пам’яті браузера. Він підходить для додатків, де потрібна максимальна відповідність сучасним вимогам безпеки. Проте реалізація за допомогою цього API потребує значних зусиль і часу через складний синтаксис і асинхронну природу роботи.

Ще одним варіантом є серверна реалізація криптографії, наприклад, з використанням модулів, доступних у середовищі Node.js. Такий підхід дозволяє переносити всі обчислення на сервер, що гарантує захист ключів і даних від потенційних загроз на клієнтському боці. Однак він додає навантаження на сервер і ускладнює архітектуру додатка, особливо якщо потрібно забезпечити шифрування безпосередньо в браузері.

Серед альтернативних підходів також можна виділити інтеграцію з хмарними сервісами, які надають готові інструменти для роботи з даними. Це рішення має свої переваги, оскільки дозволяє розробникам делегувати частину завдань з безпеки на сторонні сервіси. Однак воно не завжди відповідає потребам проекту через обмеження в конфігурації або залежність від стороннього провайдера.

Вибір бібліотеки CryptoJS для нашого проекту ґрунтується на прагматичності й ефективності цього інструменту. Вона дозволяє реалізувати алгоритм AES на клієнтському боці без зайвих труднощів. CryptoJS має простий і зрозумілий інтерфейс, що дає змогу розробникам швидко впровадити шифрування в додаток, не витрачаючи зайвий час на складну налаштування. Бібліотека також добре документована й активно підтримується спільнотою, що є важливим фактором для забезпечення надійності коду.

Окрім цього, CryptoJS оптимально підходить для веб-додатків, які виконують обробку даних у реальному часі, оскільки вона працює безпосередньо в браузері. Це дозволяє користувачам отримувати результати шифрування та дешифрування без затримок, пов’язаних із передачею даних на сервер. Водночас, на відміну від самостійної реалізації алгоритму, використання перевіреної бібліотеки мінімізує ризики появи вразливостей через помилки в коді.

Таким чином, вибір бібліотеки CryptoJS для реалізації шифрування у нашому веб-додатку забезпечив поєднання простоти, функціональності та надійності. У нашому проекті цей інструмент дозволив досягти високого рівня безпеки, залишаючись при цьому легким у використанні та підтримці.

4 РОЗРОБКА І ТЕСТУВАННЯ ВЕБ-ДОДАТКУ

4.1 Проектування основної сторінки веб-додатку

У лістингу 3.1 представлено основний код, що відповідає за головну сторінку веб-додатка. Цей фрагмент забезпечує розміщення ключових елементів інтерфейсу, таких як поля для введення інформації, яку необхідно зашифрувати, та ключа для шифрування. Сам код зберігається у файлі **index.html** і буде повністю представлений у додатках.

Лістинг коду 3.1

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SecureEncrypt</title>
  <link rel="stylesheet" href="styles.css">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
</head>
<body>
  <div class="container" id="mainContainer">
    <h1 class="app-title">SecureEncrypt</h1>
    <div class="form-group">
      <label for="inputText">Text to Secure:</label>
      <input type="text" placeholder="Enter text here" id="inputText" class="form-control">
    </div>
    <div class="form-group">
      <label for="encryptionKey">Encryption Key:</label>
      <input type="password" placeholder="Enter your key" id="encryptionKey" class="form-control" autocomplete="off">
    </div>
    <div class="action-buttons">
      <button onclick="handleEncrypt()" class="btn btn-success">Encrypt Text</button>
      <button onclick="handleDecrypt()" class="btn btn-warning">Decrypt Text</button>
```

```
    </div>
    <div class="result-section">
      <label for="outputResult">Output:</label>
      <pre id="outputResult" class="border rounded p-2"></pre>
    </div>
    <button id="googleSignIn" class="btn btn-outline-primary mt-3">
      <img class="mr-2" height="20px" src="./assets/google.svg" alt="" />Sign in with Google
    </button>
  </div>
</body>

</html>
```

Цей код немає за собою певного складного функціоналу, оскільки основна робота веб-додатку відбувається завдяки коду, що знаходиться у файлах з розширенням .js, але для коректної роботи інтернет-системи цей код є основою, від якої і будується майбутній проект.

Далі для правильного розуміння роботи програми розберемо елементи коду по черзі:

<!DOCTYPE html>: Це декларація типу документа і показує браузеру, що він повинен очікувати коду HTML5.

<html lang="en">: Відкривається тег <html>, який вказує, що це початок HTML-документа. Атрибут lang="en" вказує, що мова документа - англійська.

<head>: Внутрішній контейнер для заголовків та інших метаданих документа, які не відображаються на сторінці. Тут ми знаходимо:

<meta charset="UTF-8">: Встановлює кодування символів документа на UTF-8, що дозволяє коректно відображати символи з різних мов.

<meta name="viewport" content="width=device-width, initial-scale=1.0">: Встановлює параметри відображення на різних пристроях, зокрема ширину відповідно до розміру пристрою та початковий масштаб.

<title>Encipher</title>: Встановлює заголовок веб-сторінки, який буде відображатись у вкладці браузера або у результатах пошуку.

Підключення зовнішніх CSS-файлів за допомогою тегу <link>. В даному випадку, ми підключаємо файл "styles.css", який містить стилі для сторінки, а також "bootstrap.min.css", який є основним файлом стилів Bootstrap бібліотеки.

<body>: Внутрішній контейнер для вмісту веб-сторінки. Тут ми знаходимо:

: Створює контейнер з класом "container" і ідентифікатором "main". Цей контейнер буде містити основний вміст сторінки.

<h1 class="site-title">Encipher</h1>: Заголовок першого рівня з класом "site-title", який містить текст "Encipher". Він представляє назву сайту або заголовок сторінки.

: Контейнер для елементів вводу. В даному випадку, він містить два поля вводу типу "text" з атрибутами placeholder і id.

: Контейнер для кнопок. Містить дві кнопки з класом "btn btn-primary" і текстом "Encrypt" і "Decrypt". Виконують відповідні функції encrypt() і decrypt() при натисканні.

: Контейнер для створення рядків і колонок у Bootstrap сітці. У даному випадку, він містить одну колонку шириною "col-md-6 offset-md-3", що означає, що вона займає половину ширини контейнера з відступом відліво на половину ширини.

<pre id="output"></pre>: Елемент <pre> використовується для відображення тексту зі збереженням форматування. Має ідентифікатор "output" для подальшого використання в JavaScript.

Це загальний розбір HTML-коду. Кожен елемент і атрибут має свою роль і взаємодію з CSS-стилями та JavaScript-кодом для створення функціональної інтерактивної веб-сторінки.

Для реалізації всіх функції, що повинні бути на головній сторінці нам потрібно, під’єднати JavaScript-код у відповідному місці, а саме всередині тегу <script> у кінці файлу. Сам JavaScript-код можна писати і напряму, але для зручності написання було вибрано інший варіант, а саме створити новий файл у якому записати весь код і після цього підключити його до index.html. Тут, окрім файлів такого типу, можна також під’єднати інші бібліотеки, що будуть допомагати роботі веб-додатку. Те, як виглядає під’єднання цих файлів можна побачити у лістингу 3.2.

Лістинг коду 4.2 – під’єднання скриптів до головної сторінки

```
<button id="saveDataButton" class="btn btn-primary fixed-bottom">Save Encrypted Text</button>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/crypto-js/4.0.0/crypto-js.min.js"></script>
  <script src="https://www.gstatic.com/firebasejs/8.2.8/firebase-app.js"></script>
```

```
<script src="https://www.gstatic.com/firebasejs/8.2.8/firebase-auth.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.2.8/firebase-database.js"></script>
<script src="your-firebase-config.js"></script>
<script src="script.js"></script>
<script>
  // Після завантаження сторінки
  window.addEventListener('load', function () {
    // Змінити тип поля вводу на "password"
    var passwordInput = document.getElementById('key');
    passwordInput.setAttribute('type', 'password');
    firebase.auth().onAuthStateChanged(function (user) {
      var signInButton = document.getElementById('signInButton');
      if (user) {
        signInButton.style.display = "none"; // Приховуємо кнопку "Sign In"
      } else {
        signInButton.style.display = "block"; // Відображаємо кнопку "Sign In"
      }
    });
  });
</script>
```

Цей код не потребує детального пояснення, оскільки у ньому в основному просто під’єднуються бібліотеки, що були описані у попередніх розділах дипломної роботи, але є декілька рядків, що додають функціонал саме для головної сторінки сайту. Більш детального пояснення потребує тільки один скрипт, який виконується безпосередньо всередині, файлу і не був переданий до наступних js-кодів. Цей скрипт знаходиться у самому кінці html-файлу для того, щоб він виконувався після повного завантаження сторінки і має наступний функціонал:

Встановлення типу поля вводу для пароля: В рядку var passwordInput = document.getElementById('key'); ми отримуємо посилання на поле вводу з ідентифікатором "key", яке є полем для пароля. Потім в рядку passwordInput.setAttribute('type', 'password'); ми змінюємо тип поля вводу на "password", щоб забезпечити відображення введених символів у вигляді зірочок або крапок.

Керування відображенням кнопки "Sign With Google": За допомогою firebase.auth().onAuthStateChanged ми встановлюємо обробник події, який спрацьовує при зміні стану авторизації користувача. Внутрішня функція приймає параметр user,

який представляє поточного користувача.

Якщо user існує (користувач авторизований), то кнопка "Sign With Google" приховується за допомогою signInButton.style.display = "none";.

Якщо user не існує (користувач неавторизований), то кнопка "Sign With Google" відображається за допомогою signInButton.style.display = "block";.

Цей скрипт забезпечує зміну типу поля вводу для пароля і керує відображенням кнопки "Sign With Google" залежно від стану авторизації користувача.

4.2 Програмування основного функціоналу головної сторінки

Головний функціонал головної сторінки знаходиться у файлі script.js, там знаходяться функції що відповідають за створення деяких нових кнопок для авторизованих користувачі, а також за функціонал кнопок, авторизацію, запис інформації у базу даних. Весь лістинг коду, що знаходиться у файлі script.js. Можна побачити у додатках, але основну увагу потрібно надати функціям, що відповідають за шифрування і дешифрування даних, їх буде наведено у Лістингу 4.3

Лістинг коду 4.3 – реалізація функції шифрування і дешифрування даних

```
function encrypt() {
  const text = document.getElementById("text").value;
  const key = document.getElementById("key").value;
  encryptedText = CryptoJS.AES.encrypt(text, key).toString();



  const modal = document.createElement("div");
  modal.className = "modal fade";
  modal.innerHTML = `
    <div class="modal-dialog">
      <div class="modal-content">
        <div class="modal-header">
          <h5 class="modal-title">Encrypted Text</h5>
          <button type="button" class="close" data-dismiss="modal">&times;</button>
        </div>
        <div class="modal-body">
          <textarea class="form-control" rows="5" readonly>${encryptedText}</textarea>
```

```
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-primary" data-dismiss="modal">Close</button>
      </div>
    </div>
  </div>
 `;

 // Show the modal dialog.
 document.body.appendChild(modal);
 $(modal).modal("show");
}




const saveDataButton = document.getElementById("saveDataButton");
saveDataButton.addEventListener("click", function () {
 saveData(encryptedText);
});

function saveData(encryptedText) {
 // Отримуємо ідентифікатор поточного користувача
 const userId = firebase.auth().currentUser?.uid;

 // Перевіряємо, чи користувач увійшов в систему
 if (userId) {
  const userDataRef = database.ref('users/' + userId + '/encryptedTexts');

  // Генеруємо новий унікальний ключ для запису зашифрованого тексту
  const newEncryptedTextRef = userDataRef.push();

  // Встановлюємо значення зашифрованого тексту та дати для нового запису
  newEncryptedTextRef.set({
   encryptedText: encryptedText,
   timestamp: firebase.database.ServerValue.TIMESTAMP
  })
```

```
      .then(() => {
        console.log('Дані успішно збережено');
      })
      .catch((error) => {
        console.log('Помилка при збереженні даних:', error);
      });
  } else {
    console.log('Ідентифікатор користувача не знайдено');
  }
}
```

```
function decrypt() {
  const encryptedText = document.getElementById("text").value;
  const key = document.getElementById("key").value;

  try {
    const decryptedBytes = CryptoJS.AES.decrypt(encryptedText, key);
    const decryptedText = decryptedBytes.toString(CryptoJS.enc.Utf8);

    // Create a modal dialog for displaying the decrypted text.
    const modal = document.createElement("div");
    modal.className = "modal fade";
    modal.innerHTML = `
      <div class="modal-dialog">
        <div class="modal-content">
          <div class="modal-header">
            <h5 class="modal-title">Decrypted Text</h5>
            <button type="button" class="close" data-dismiss="modal">&times;</button>
          </div>
          <div class="modal-body">
            <textarea class="form-control" rows="5" readonly>${decryptedText}</textarea>
          </div>
          <div class="modal-footer">
            <button type="button" class="btn btn-primary" data-dismiss="modal">Close</button>
```

```
        </div>
      </div>
    </div>
  `;

  // Show the modal dialog.
  document.body.appendChild(modal);
  $(modal).modal("show");
 } catch (error) {
  showDecryptionErrorModal();
 }
}
```

Цей код містить три функції: encrypt(), saveData(), та decrypt(). Основний функціонал коду пов'язаний з шифруванням та розшифруванням тексту за допомогою бібліотеки CryptoJS, а також збереженням та відображенням даних.

Функція encrypt() отримує значення тексту та ключа з відповідних полів вводу, шифрує текст за допомогою CryptoJS, створює модальне вікно з зашифрованим текстом та відображає його на сторінці.

Функція saveData(encryptedText) відповідає за збереження зашифрованого тексту в базі даних Firebase Realtime Database. Вона отримує зашифрований текст та ідентифікатор поточного користувача, створює новий запис з зашифрованим текстом та датою, і додає його до вузла даних користувача. У разі успішного збереження виводить повідомлення про успішне збереження, в іншому випадку виводить повідомлення про помилку.

Функція decrypt() отримує значення зашифрованого тексту та ключа з відповідних полів вводу, намагається розшифрувати текст за допомогою CryptoJS. Якщо розшифрування успішне, створює модальне вікно з розшифрованим текстом та відображає його на сторінці. У випадку помилки при розшифруванні викликає функцію showDecryptionErrorModal(), яка відповідає за відображення модального вікна з повідомленням про помилку розшифрування (ця функція не надана в наданому коді).

Інша частина коду, що вказана у лістингу 3.4 відповідає уже за допоміжні функції сайту, такі як перехід до історії, реєстрація, шифрування текстового

файлу. В основному функції, які будуть знаходитись тут з’являються тільки після реєстрації користувача.

Лістинг 4.4

```
document.getElementById("signInButton").addEventListener("click", function () {
  firebase.auth().signInWithPopup(provider)
    .then((res) => {
      user = res.user;
      userId = user.uid;
      localStorage.setItem('userId', userId);
      signInButton.style.display = "none"; // Приховуємо кнопку "Sign In"
      createUserNameButton(user.displayName);
      createFileEnchancerButton();
          createSignOutButton();
    })
    .catch((err) => {
      console.log("Error Occurred");
      console.log(err.code);
      signInButton.style.display = "block"; // Відображаємо кнопку "Sign In"
    });
});
window.addEventListener('load', function() {
  // Check if there is a stored user ID
  const storedUserId = localStorage.getItem('userId');
  if (storedUserId) {
    userId = storedUserId;
    signInButton.style.display = "none";
    if (user && user.displayName) {
      createUserNameButton(user.displayName);
      createFileEnchancerButton();
      createSignOutButton();
    } else {
      firebase.auth().onAuthStateChanged(function(user) {
        if (user && user.displayName) {
          createUserNameButton(user.displayName);
          createFileEnchancerButton();
          createSignOutButton();
```

```
        } else {
          createUserNameButton("User");
        }
       });
    }
  } else {
    signInButton.style.display = "block";
  }
});
function createUserNameButton(displayName) {
  var userNameButton = document.createElement("button");
  userNameButton.classList.add("user-button");
  if (displayName) {
    userNameButton.innerHTML = displayName;
    userNameButton.addEventListener("click", function () {
      window.location.href = "savedData.html";
    });
  } else {
    userNameButton.innerHTML = "User";
  }
  document.getElementById("main").appendChild(userNameButton);
  document.getElementById("signInButton").style.display = "none";
}
function createFileEnchancerButton(){
  var fileEnchancerButton = document.createElement("button");
  fileEnchancerButton.classList.add("fileEnchancer-button");
  fileEnchancerButton.innerHTML = "File Enchancer";
  fileEnchancerButton.addEventListener("click", function () {
    window.location.href = "FileEnchancer.html";
  });
  document.getElementById("main").appendChild(fileEnchancerButton);
}
function createSignOutButton() {
  var signOutButton = document.createElement("button");
  signOutButton.classList.add("signout-button");
  signOutButton.innerHTML = "Sign Out";
  signOutButton.addEventListener("click", signOut);
```

```
  document.getElementById("main").appendChild(signOutButton);
}
function signOut() {
  firebase.auth().signOut().then(function() {
    // Sign-out successful.
    localStorage.removeItem('userId');
    window.location.reload();
  }).catch(function(error) {
    // An error happened.
    console.log(error);
  });
}
window.addEventListener('load', function() { ... });
```

Ця функція виконується при завантаженні сторінки. Вона перевіряє, чи є збережений ідентифікатор користувача у локальному сховищі. Якщо такий ідентифікатор існує, то встановлюється змінна userId, приховується кнопка "Sign In" і створюються відповідні кнопки для користувача.

function createUserNameButton(displayName) { ... }

Ця функція створює кнопку з ім'ям користувача. Якщо передано значення displayName, то кнопка отримує це ім'я, а також додає обробник події для переходу до сторінки "savedData.html". Якщо значення displayName відсутнє, кнопка отримує текст "User". Кнопка додається до елементу з ідентифікатором "main".

function createFileEnchancerButton() { ... }

Ця функція створює кнопку "File Enchancer". Додається обробник події, який переходить до сторінки "FileEnchancer.html". Кнопка додається до елементу з ідентифікатором "main".

function createSignOutButton() { ... }

Ця функція створює кнопку "Sign Out" для виходу з облікового запису. При кліку на кнопку виконується вихід з облікового запису, очищення збереженого ідентифікатора користувача і перезавантаження сторінки.

Цей набір функцій виконує дії, пов'язані з автентифікацією користувача, створенням відповідних кнопок і керуванням станом аутентифікації на сторінці. Вони спрощують роботу з Firebase та дозволяють здійснювати взаємодію з обліковим

записом користувача.

Після виконання основного коду програми уже можна побачити як виглядає сайт у початковому вигляді. Вигляд сайту можна побачити на рисунку 4.1

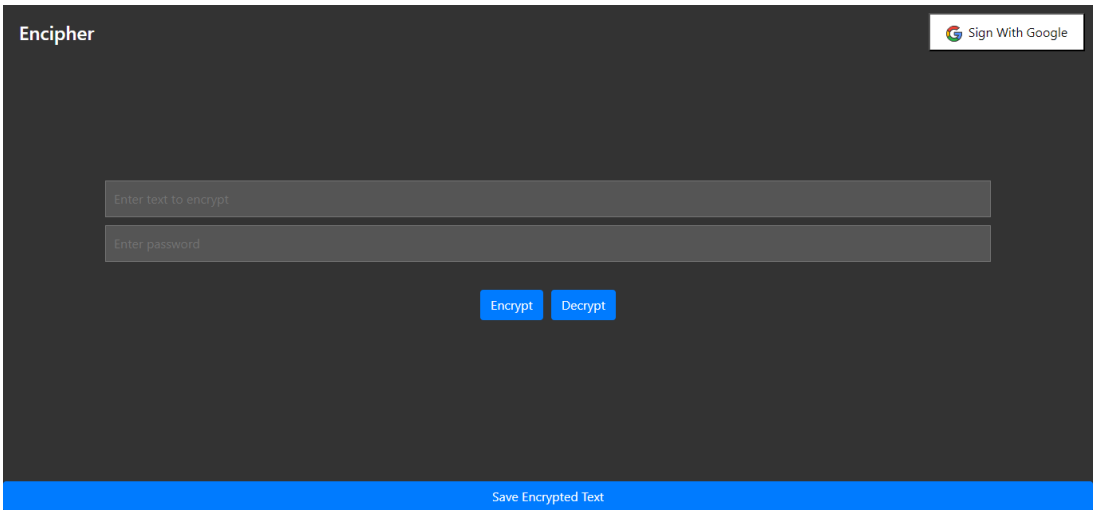

Рисунок 4.1 – Головна сторінка сайту

Для створення подальшого функціоналу сайту, що був заданий у завданні потрібно створити ще дві сторінки, пояснення їх коду можна буде побачити у наступних розділах.

4.3 Створення сторінки для перевірки історії раніше зашифрованих даних

Для зручності користуванням сайту було створено додаткову сторінку, на яку можна перейти тільки, якщо користувач заре’єстрований. Сторінка повинна містити декілька елементів, а саме кнопка для показу історії і її видалення з спеціального екрану, а також кнопка для повернення на головну сторінку з можливістю шифрувати дані і записувати їх в історію.

На головній сторінці ми уже надали функціонал кнопці яка висвітлює ім’я зареєстрованого користувача, тому тепер при натисканні ми будемо переміщатись на сторінку з історією, після успішної реєстрації. Основний код цієї сторінк наведено у лістингу 4.5.

Лістинг коду 4.5 – Основний код сторінки з історією зашифрованих даних користувача

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Saved Data</title>
  <link rel="stylesheet" href="styles.css">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
</head>
<body>
  <div class="container">
    <div class="header">
      <h1 class="site-title">Saved Data</h1>
      <button id="goToEncryptionButton" class="go-to-encryption-button">Go To Encryption</button>
    </div>
    <div class="row">
      <div class="col-md-6 offset-md-3">
        <div id="dataContainer"></div>
        <div class="button-container">
          <button id="historyButton" class="btn btn-primary custom-button">History</button>
          <button id="deleteHistoryButton" class="btn btn-danger custom-button">Delete History</button>
        </div>
      </div>
    </div>
  </div>
```

У даному коді ми маємо HTML-сторінку для "Saved Data". Структура коду включає наступні елементи:

У розділі <head>:

Встановлення кодування символів: <meta charset="UTF-8">.

Налаштування viewport: <meta name="viewport" content="width=device-width, initial-scale=1.0">.

Заголовок сторінки: <title>Saved Data</title>.

Підключення зовнішніх стилів styles.css: <link rel="stylesheet" href="styles.css">.

Підключення стилів з бібліотеки Bootstrap: <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">.

У розділі <body>:

Головний контейнер з класом container.

Заголовок сторінки з класом site-title і текстом "Saved Data".

Кнопка з класом go-to-encryption-button і текстом "Go To Encryption".

Розділ для розміщення вмісту у вигляді сітки Bootstrap.

Вкладений для центрування вмісту на середині сторінки.

Контейнер з id dataContainer, який буде містити дані.

Контейнер з класом button-container для розміщення кнопок.

Кнопка з id historyButton, класом btn btn-primary custom-button і текстом "History".

Кнопка з id deleteHistoryButton, класом btn btn-danger custom-button і текстом "Delete History".

Це загальна структура коду для "Saved Data" сторінки у якій не вказані посилання на файли бібліотек і файлів, що підключаються у тегові <script>. Загалом ця сторінка має три елементи, що надають їй функціоналу, тому ми повинні більш детально розглянути саме їх. Три елементи – це три обробника подій для кнопок, що були створені в тегові <body>. Лістинг коду цих обробників можна побачити у лістингу 4.6.

Лістинг коду 4.6 – Обробник подій для задання функціоналу сторінки

```
document.getElementById('goToEncryptionButton').addEventListener('click', function() {
    window.location.href = 'index.html';
  });
  document.getElementById('historyButton').addEventListener('click', function() {
    const userId = firebase.auth().currentUser?.uid;
    if (userId) {
      const userDataRef = database.ref('users/' + userId + '/encryptedTexts');
      userDataRef.once('value')
        .then((snapshot) => {
          const encryptedTexts = snapshot.val();
          var dataContainer = document.getElementById('dataContainer');
          dataContainer.innerHTML = '';
          if (encryptedTexts) {
            for (const key in encryptedTexts) {
              const encryptedText = encryptedTexts[key].encryptedText;
```

```
            const timestamp = encryptedTexts[key].timestamp;
            const date = new Date(timestamp).toLocaleString();

            var dataElement = document.createElement('p');
            dataElement.textContent = encryptedText + ' (додано ' + date + ')';
            dataContainer.appendChild(dataElement);
          }
          console.log('Зчитані дані:', encryptedTexts);
        } else {
          console.log('Дані не знайдено');
        }
      })
      .catch((error) => {
        console.log('Помилка при зчитуванні даних:', error);
      });
  } else {
    console.log('Ідентифікатор користувача не знайдено');
  }
});
document.getElementById('deleteHistoryButton').addEventListener('click', function() {
  var dataContainer = document.getElementById('dataContainer');
  dataContainer.innerHTML = '';
});
```

Перший обробник події відповідає кнопці goToEncryptionButton. При натисканні на цю кнопку виконується функція, яка змінює поточне місцезнаходження сторінки на index.html.

Другий обробник події пов'язаний з кнопкою historyButton. При натисканні на цю кнопку виконується функція, яка перевіряє ідентифікатор поточного користувача в Firebase Authentication. Якщо ідентифікатор користувача доступний, відбувається зчитування даних з бази даних. Якщо дані присутні, вони виводяться на екран разом з датою додавання. Якщо дані відсутні, виводиться повідомлення про це.

Третій обробник події пов'язаний з кнопкою deleteHistoryButton. При натисканні на цю кнопку виконується функція, яка очищує контейнер dataContainer, видаляючи всі дані з нього.

Таким чином, код містить обробники подій для трьох кнопок, кожен з яких

виконує певні дії у відповідності до натискання користувачем.

Сам вигляд цієї сторінки після внесення налаштувань у css код можна побачити на рисунку 4.2

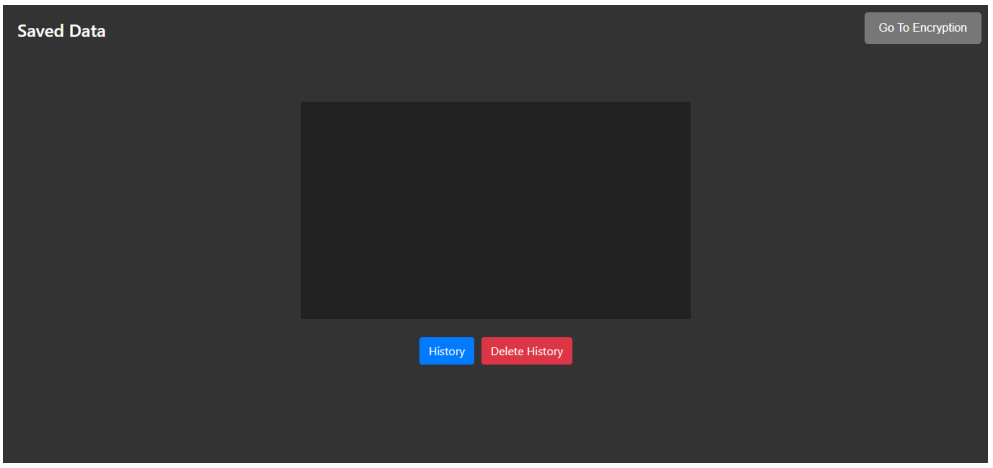

Рисунок 4.2 – Сторінка для перевірки збережених даних

Створення цієї сторінки було обов'язковим для забезпечення зручності користувачам, у ній ми використали інформацію, яку завантажували до бази даних на головній сторінці. А також надали користувачу можливість у будь-який момент переходити назад до головної сторінки і мати можливість

4.4 Створення сторінки для шифрування текстових файлів

Останньою функціональною сторінкою, було вирішено створити вікно у якому є можливість завантажувати текстовий файл до інтернет системи і після шифрування завантажувати уже зашифрований файл на ПК. Перехід на цю сторінку також можна здійснити тільки у випадку коли користувач уже авторизований у системі веб-додатку. Сама сторінка має назву File enchancer і на сторінці повинно знаходитись 4 об'єкта, поле для вводу ключа і три кнопки з наступним функціоналом: завантажити текстовий файл у систему, зашифрувати текст, що знаходиться в файлі і завантажити уже зашифрований файл. Реалізацію цих елементів на сторінці можна побачити у Лістингу 3.7

Лістинг коду 4.7 – Створення елементів сторінки File Enchancer

```
<div class="input-container">
    <input type="file" id="fileInput" accept=".txt">
    <input type="text" id="encryptionKey" placeholder="Enter key">
```

```
    <button id="encryptButton">Encrypt</button>
  </div>
  <button id="downloadEncryptedButton" disabled>Download encrypted file</button>
  <button id="returnButton" class="go-to-encryption-button">Go to Encryption</button>
```

Після створення цих елементів на сторінці нам потрібно надати їм функціонал у тегові <script>, а також підключити відповідні бібліотеки. Функціонал кнопкам буде надано відповідним кодом, що можна побачити у лістингу 3.8.

Лістинг коду 4.8 – Надання функціоналу елементам сторінки

```
document.getElementById('encryptButton').addEventListener('click', function() {
    var fileInput = document.getElementById('fileInput');
    var encryptionKey = document.getElementById('encryptionKey').value;
    var file = fileInput.files[0];
    if (file && encryptionKey) {
      var fileReader = new FileReader();
      fileReader.onload = function(event) {
        var fileContent = event.target.result;
        var encryptedContent = CryptoJS.AES.encrypt(fileContent, encryptionKey).toString();
        var downloadLink = document.createElement('a');
        downloadLink.href = 'data:text/plain;charset=utf-8,' +
encodeURIComponent(encryptedContent);
        downloadLink.download = file.name + '.encrypted.txt';
        downloadLink.style.display = 'none';
        document.body.appendChild(downloadLink);
          document.getElementById('downloadEncryptedButton').addEventListener('click', function()
{
          downloadLink.click();
        });

        document.getElementById('downloadEncryptedButton').disabled = false;
      };
      fileReader.readAsText(file);
    }
  });
  document.getElementById('returnButton').addEventListener('click', function() {
    window.location.href = 'index.html';
  });
```

У цьому коді ми маємо наступні елементи:

- fileInput: елемент вводу файлу. Використовується для отримання обраного файлу користувачем.
- encryptionKey: елемент вводу текстового поля. Використовується для отримання введеного користувачем ключа шифрування.
- file: змінна, яка містить обраний файл користувачем.

Після отримання цих елементів виконується перевірка на наявність файлу та ключа шифрування. Якщо обидва присутні, створюється об'єкт FileReader для читання вмісту файлу.

- FileReader має обробник події onload, який виконується після успішного завантаження вмісту файлу. У цьому обробнику події відбувається наступне:
- Отримується вміст файлу з результату завантаження за допомогою event.target.result та зберігається у змінній fileContent.
- Вміст файлу шифрується за допомогою бібліотеки CryptoJS, використовуючи ключ шифрування. Зашифрований вміст зберігається у змінній encryptedContent.
- Створюється елемент <a> для створення посилання на зашифрований файл. Встановлюються відповідні атрибути, такі як href, download та style. Елемент не відображається на сторінці.
- Додається обробник події для кнопки завантаження зашифрованого файлу. При натисканні кнопки викликається функція downloadLink.click(), що запускає завантаження файлу.
- Активується кнопка завантаження, встановлюючи disabled в значення false. Тепер користувач може завантажити зашифрований файл.

Цей код виконує обробку події натискання на кнопку "Encrypt" та виконує шифрування обраного файлу за допомогою введеного ключа шифрування.

Останній фрагмент коду надає функціонал кнопці 'returnButton', для того, щоб була можливість у користувача повернутись назад до головної сторінки сайту.

Саму сторінку можна побачити на рисунку 4.3.

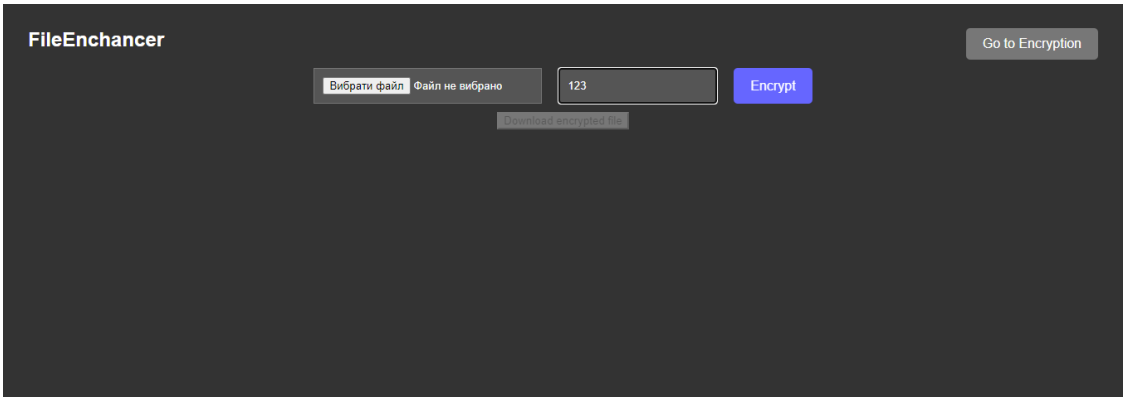

Рисунок 4.3 – Робота сторінки File Enchancer

Після того як користувач вибрав текстовий файл для шифрування, вписав код і натиснув кнопку зашифрувати, кнопка завантажити зашифрований файл почне підсвічуватись і користувач отримає можливість завантажити його.

4.5 Налаштування Firebase

Оскільки ми уже створили весь необхідний функціонал інтернет системи, для його коректної роботи потрібно підключити Firebase. Для того, щоб правильно налаштувати і підключити необхідні функції цієї платформи, потрібно завантажити платформу для виконання мережевих застосунків, а саме Node.js.. Ця платформа надає нам можливість користуватись пакетним менеджером npm (Node Package Manager). npm є стандартним пакетним менеджером для Node.js та використовується для установки, оновлення та керування залежностями JavaScript-проектів.

За допомогою npm можна швидко встановлювати залежності для своїх проектів, включаючи пакети, які розробляються спільнотою, а також власні пакети. Ви також можете використовувати npm для управління версіями пакетів, виконання сценаріїв побудови проекту та багато іншого.

Для того, щоб розпочати налаштування і ініціалізацію Firebase нам потрібно зареєструватись у цьому сервісі, оскільки це сервіс від фірми Google нам потрібно використати свою електронну адресу для реєстрації.

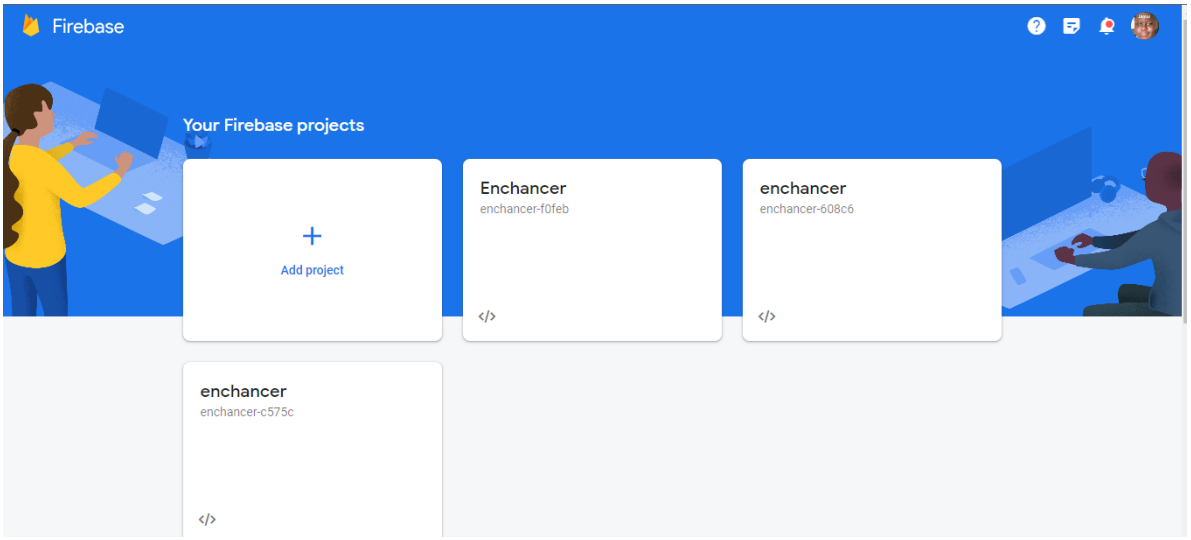

Рисунок 4.4 – Зовнішній вигляд сайту

Після успішної реєстрації користувачу сервісом для продовження роботи потрібно створити проект. Для цього просто натискаємо кнопку додати проект, надаємо йому назву і приймаємо умови користування сервісом. Після чого проект буде створено. На рисунку 4.5 можна побачити приклад успішного створення проекту.

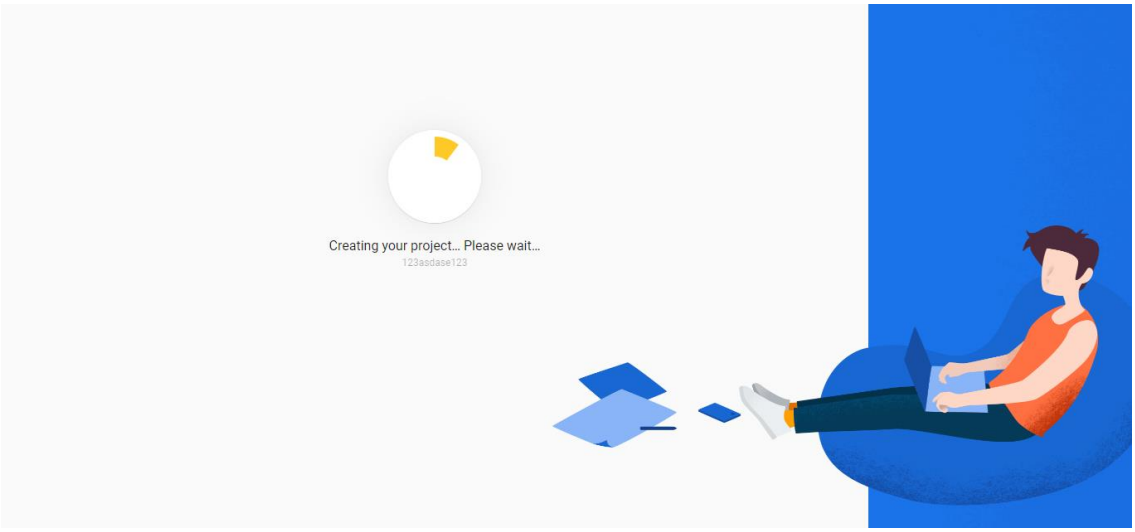

Рисунок 4.5 – Успішне створення проекту

Firebase надає можливість створювати проекти на різних платформах, а також має можливість робити свої проекти мультиплатформеними, але при виконанні дипломної роботи на цікавить тільки кнопка web. Після натискання на неї потрібно слідувати інструкціям, що з’являються. І відразу під час налаштування проекту можна розпочати його хостити.

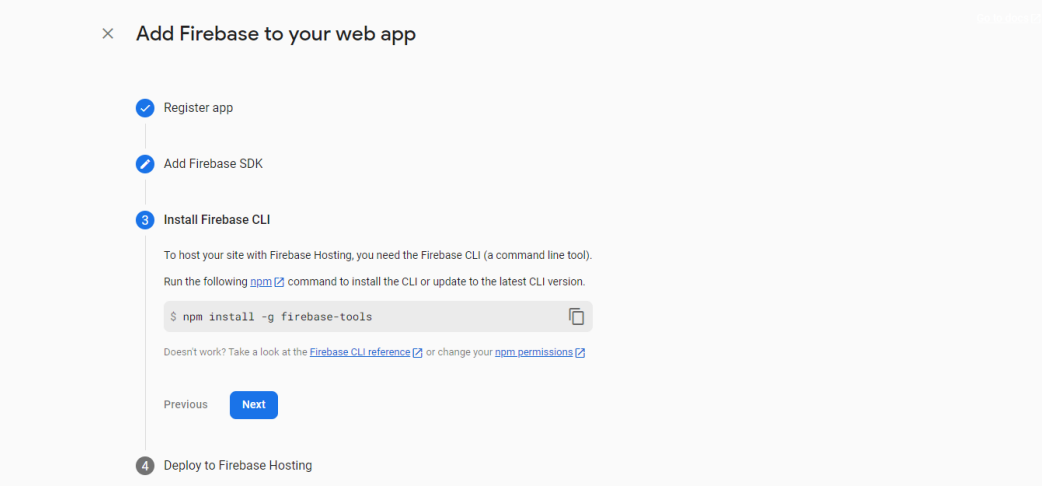

Рисунок 4.6 – Реєстрація проекту

І уже після цього нам і потрібно почати використовувати npm, а саме в проекті відкрити консоль і вписати команди, що нам надавали під час реєстрації, а саме команду npm install -g firebase-tools. Вона завантажить до нашого проекту необхідні для роботи з firebase інструменти, вони дозволять нам розпочати хост нашого веб-додатка. За допомогою інших команд. Саме підключення до хмарного хостинга після реєстрації буде виглядати наступним чином

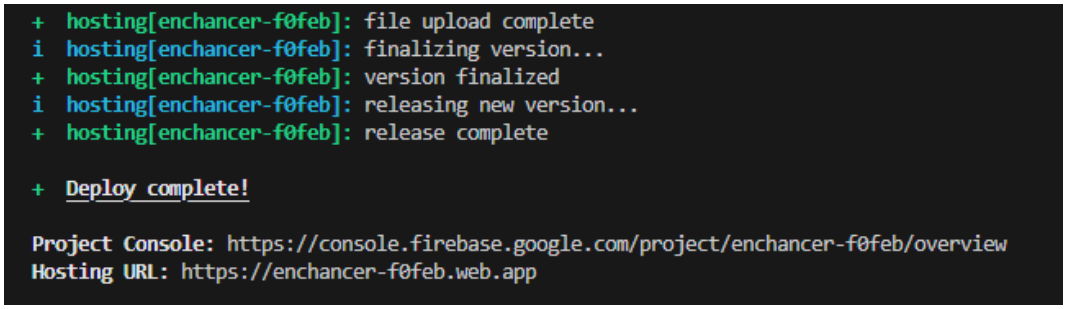

Рисунок 3.7 – Підключення до хмарного сервісу

Після цього ми отримуємо посилання на наш сайт, але воно не буде працювати, оскільки ми не налаштували правильно конфігурацію firebase, для того щоб її налаштувати нам при реєстрації додатку було видано ApiKey і інші дані для конфігурації. Ці дані ми записали до файлу your-firebase-config.js код цього файлу вказано у лістингу 4.9

Лістинг коду 4.9 – налаштування конфігурації Firebase

```
var firebaseConfig = {
  apiKey: "##########################",
  authDomain: "enchancer-f0feb.firebaseapp.com",
```

```
  databaseURL:"https://enchancer-f0feb-default-rtdb.europe west1.firebasedatabase.app",
  projectId: "enchancer-f0feb",
  storageBucket: "enchancer-f0feb.appspot.com",
  messagingSenderId: "###################",
  appId: "###################################"
 };
 firebase.initializeApp(firebaseConfig);
```

Після налаштування конфігурації ми уже можемо повноцінно користуватися сервісом. Налаштування функції для сайту дуже легко провести для цього переходимо на сайті Firebase до свого створеного додатку і перевіряємо які самі функції можна підключити. На рисунку 4.8 можна побачити Для того, щоб їх підключити просто натискаємо на варіант, що нам потрібен і налаштовуємо його по інструкції, що показується при підключені будь-якої функції.

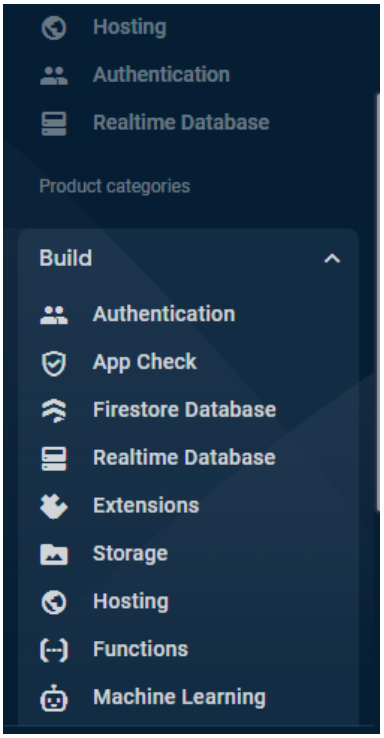

Рисунок 4.8 – Функції Firebase

Після проведення всіх вище вказаних налаштувань і підключення необхідних функцій, ми уже в коді самої програми можемо налаштовувати їх як забажаємо, окрім цього у самому сервісі можемо перевіряти чи правильно все працює.

4.6 Тестування розробленої інтернет системи

Після розробки інформаційної інтернет-системи, важливо перевірити чи все правильно працює. Для цього ми перевіримо всі етапи розробки ,які ми проходили. Для початку перевіримо правильність налаштування хостингу і перейдемо по посиланню що знаходиться у додатку Firebase.

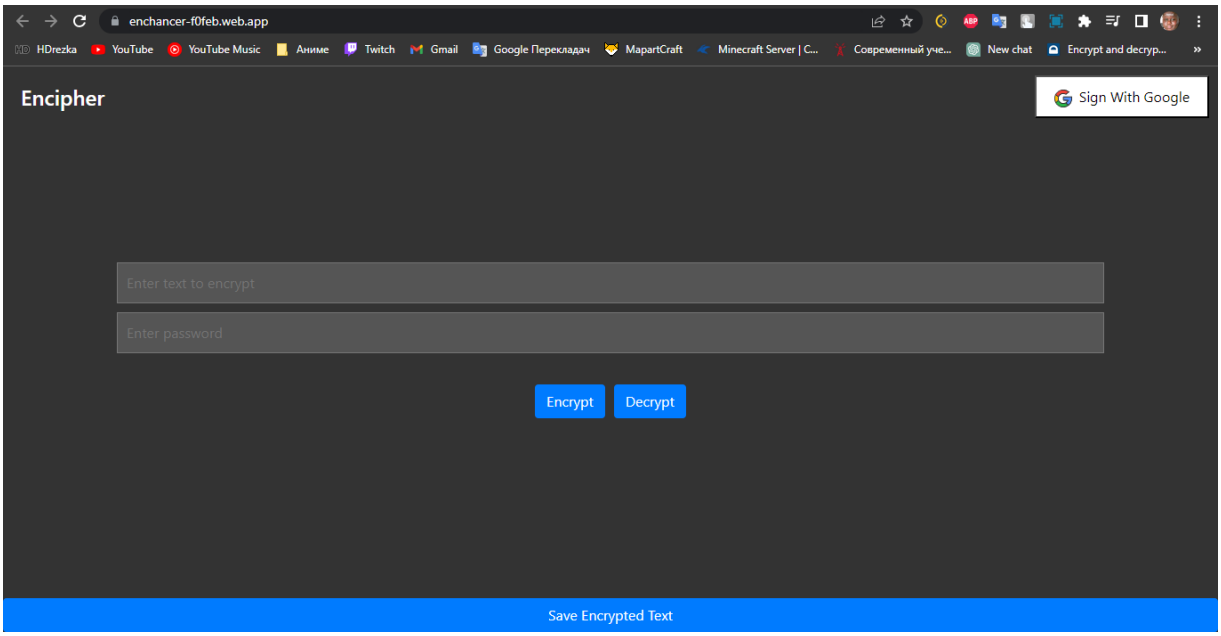

Рисунок 4.9 – Запуск сайту

Після запуску веб-додатку нам потрібно визначити чи коректно працює авторизація і шифрування даних, для цього натискаємо кнопку реєстрації і вводимо потрібний текст разом з ключем.

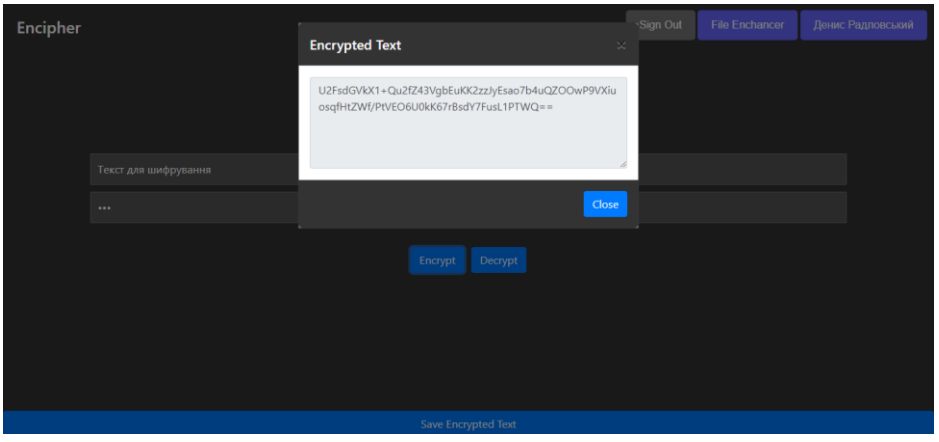

Рисунок 4.10 – Приклад коректної роботи функцій головної сторінки

Оскільки шифрування працює коректно, потрібно також перевірити роботу бази даних і історії, для цього закриваємо модальне вікно і натискаємо кнопку зберегти дані. Після натиску на кнопку зберігання даних перейдемо на сторінку історії і виведемо її на екран.

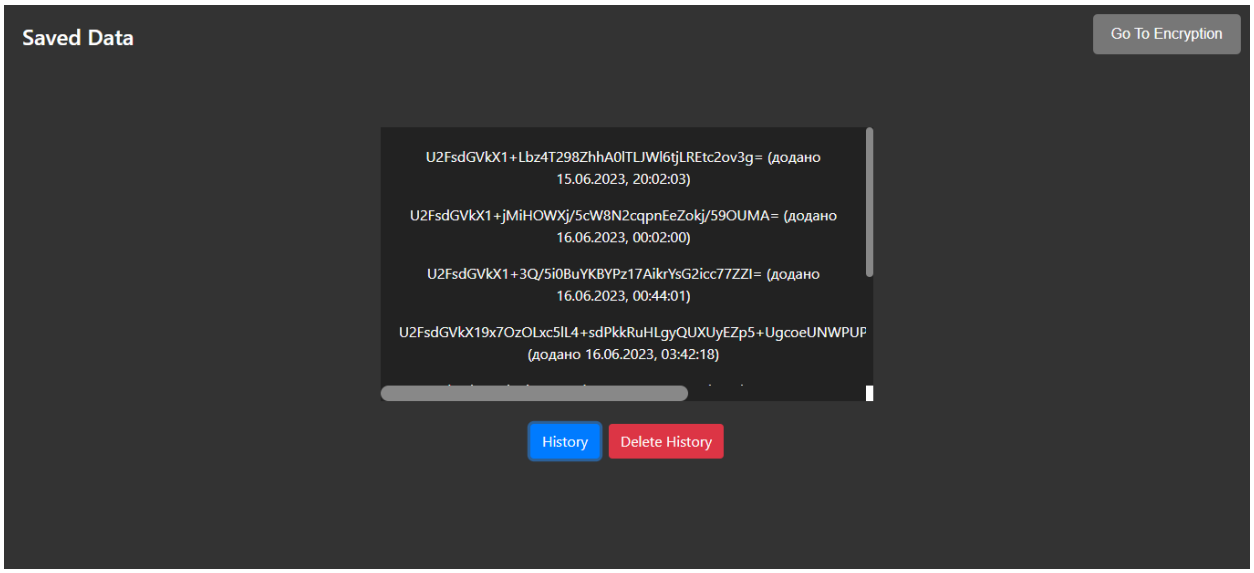

Рисунок 4.11 – Приклад коректної роботи історії

Всі функції працюють коректно і в кінці потрібно перевірити роботу сторінки для шифрування текстових файлів. Для цього ми створили спеціальний текстовий файл з заготовленим текстом і провели його через шифрування.

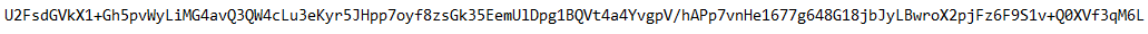

Рисунок 4.12 – Робота функції для шифрування файлів

Після перевірки всіх функцій веб-додатку ми визначили, що весь код працює коректно, а також сервіс Firebase працюэ правильно.

4.7 Розробка інструкції користувача

Відкрийте посилання на веб-додаток, а саме https://enchancer-f0feb.web.app/

Для реєстрації у інтернет системі натисніть кнопку зареєструватись у правому верхньому куті.

У поле для вводу тексту до шифрування введіть потрібний вам текст, після чого введіть ключ для шифрування, натисніть кнопку шифрувати.

Скопіюйте текст, що з’явився в модальному вікні і збережіть в безпечному місці, або натисніть кнопку зберегти, що знаходиться у низу головної сторінки, тоді зашифрований текст збережеться у базі даних.

Щоб знайти раніше зашифрований текст, що зберігся у базі даних натисніть на

кнопку на якій знаходиться ім’я користувача і у вікні що відкрилось натисніть на кнопку історія, якщо вам потрібно очистити історію з екрану, натисніть кнопку видалити історію.

Для переходу на попередню сторінку у правому верхньому куті знаходиться кнопка під назвою перейти до шифрування.

Для того, щоб зашифрувати текстовий файл, натисніть кнопку шифрувальник файлів, після чого завантажте файл в систему і введіть ключ для шифрування, натискаєте спочатку кнопку шифрувати і тільки після цього можна буде завантажити зашифрований файл.

Якщо вам потрібно вийти з вашого облікового запису перейдіть на головну сторінку і натисніть вийти з акаунту.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку веб-сервісу з можливістю шифрування даних. Особливістю розробки є підвищення рівня захищеності веб-сервісу за рахунок шифрування даних, так як в наш час почастішали випадки кіберзлочинності і питання шифрування даних набирає актуальності з збільшенням обчислювальних можливостей пристроїв.

Аналогом може бути Cryptii – 47000 грн.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 4.1.

Таблиця 4.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Крите-рій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність продукту в реальних умовах
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку

Продовження табл. 5.1

Ринкові переваги					
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так із комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження табл. 5.1

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 5.2

Таблиця 5.2 – Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	3	4	4
Наявність аналогів на ринку	3	3	3
Цінова політика	4	4	4
Технічні та споживчі властивості виробу	4	3	3
Експлуатаційні витрати	4	4	3
Ринок збуту	4	4	4
Конкурентоспроможність	3	4	3
Фахівці з технічної і комерційної реалізації	4	3	4
Фінансування	4	4	3
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	3	4	4
Супровідна документація	3	3	4
Сума	42	43	42
Середньоарифметична сума балів	(42+43+42) / 3 = 42,33		

За даними таблиці 5.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 5.3.

Таблиця 5.3 - Рівні комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається підвищенням рівня захищеності веб-сервісу за рахунок шифрування даних, так як в наш час почастішали випадки кіберзлочинності і питання шифрування даних набирає актуальності з збільшенням обчислювальних можливостей пристроїв.

5.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

5.2.1 Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{M}{T_р} \cdot t, \quad (5.1)$$

де М – місячний посадовий оклад конкретного розробника (дослідника), грн.;

$T_р$ – число робочих днів за місяць, 22 днів;

t – число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 4.4.

Таблиця 4.4 – Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	40000	1818,18	42	76363,636
Програміст	36000	1636,36	42	68727,273

Всього	145090,91

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

5.2.2 Додаткова заробітна плата розробників, які брати участь в розробці обладнання/програмного продукту.

Додаткову заробітну плату прийнято розраховувати як 14,5 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 14{,}5\ \% / 100\ \% \qquad (5.2)$$

$З_д = (145090{,}91 \cdot 14{,}5\ \% / 100\ \%) = 21038{,}18$ (грн.)

5.2.3 Нарахування на заробітну плату розробників.

Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \qquad (5.3)$$

$Н_з = (145090{,}91 + 21038{,}18) \cdot 22\ \% / 100\ \% = 36548{,}40$ (грн.)

5.2.4. Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

5.2.5 Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді розраховується за формулою:

$$A = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12} \text{ [грн.].} \qquad (5.4)$$

де Ц – балансова вартість обладнання, грн.;

Т – термін корисного використання обладнання згідно податкового законодавства, років

$t_{вик}$ – термін використання під час розробки, місяців

Розрахуємо, для прикладу, амортизаційні витрати на комп'ютер балансова вартість якого становить 25000 грн., термін його корисного використання згідно податкового законодавства – 2 роки, а термін його фактичного використання – 1,91 міс.

$$A_{обл} = \frac{25000}{2} \times \frac{1{,}91}{12} = 1988{,}636 \; грн.$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 4.5. Так як вартість ліцензійної ОС та спеціалізованих ліцензійних нематеріальних ресурсів менше 20000 грн, то даний нематеріальний актив не амортизується, а його вартість включається у вартість розробки повністю, $B_{нем.ак.}$ = 11000 грн.

Таблиця 5.5 – Амортизаційні відрахування на матеріальні та нематеріальні ресурси для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортиза-ційні відрахуван-ня, грн.
Комп'ютер та комп'ютерна периферія	25000	2	1,91	1988,636
Офісне обладнання (меблі)	33000	4	1,91	1312,500
Приміщення	1800000	20	1,91	14318,182
Всього				17619,32

5.2.6 Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому

тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \quad (5.5)$$

де В – вартість 1 кВт-години електроенергії для 1 класу підприємства з ПДВ в 2024 році для Вінницької області за даними Енера-Вінниця, В = 10,42 грн./кВт;

П – встановлена потужність обладнання, кВт. П = 0,3 кВт;

Ф – фактична кількість годин роботи обладнання, годин.

$К_п$ – коефіцієнт використання потужності, $К_п$ = 0,9.

$В_е = 0,9 \cdot 0,3 \cdot 8 \cdot 42 \cdot 10,42 = 945,3024$ (грн.)

5.2.7 Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.6)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

$I_в$ = 145090,91 * 70% / 100% = 101563,6 (грн.)

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за

статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$H_{нзв} = (З_о + З_р) \cdot \frac{H_{нзв}}{100\%}, \quad (5.7)$$

де $H_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$H_{нзв}$ = 145090,91 * 150 % / 100 % = 217636 (грн.)

5.2.9 Витрати на проведення науково-дослідної роботи.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$B_{заг}$ = 145090,91+21038,18+36548,40+17619,32+11000+945,30+101563,6+
+217636 = 551442,11 грн.

5.2.11 Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{B_{заг}}{\eta} \text{ (грн)}, \quad (5.8)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

ЗВ = 551442,11 / 0,5 = 1102884 грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

а) вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

б) зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

в) кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб’єктів (зацікавлених осіб) цього попиту;

г) визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов’язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

- абсолютного економічного ефекту (чистого дисконтованого доходу);
- внутрішньої економічної дохідності (внутрішньої норми дохідності);
- терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації

потенційним інвестором можна об’єднати, враховуючи визначені ситуації з відповідними умовами.

5.3.1 Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.9)$$

де $\pm\Delta Ц_o$ – зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_o = Ц_б \pm \Delta Ц_o$;

$Цб$ – вартість програмного продукту у році до впровадження результатів розробки;

ΔN – збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт $\lambda = 0{,}8333$.

ρ – коефіцієнт, який враховує рентабельність продукту;

ϑ – ставка податку на прибуток, у 2024 році ϑ =18%.

Припустимо, що при прогнозованій ціні 25000 грн. за одиницю, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на *500* грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року – на 1500 шт., протягом другого року – на 1300 шт., протягом третього року на 1000 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

*$\Delta П_1$ = (0*500 + (*25000 + 500 *)*1500)* 0,8333* 0,23) * (1 - 0,18) = 5893749,764 грн.*

$\Delta П_2$ = (0*500 + (25000 + 500)*(1500+1300)* 0,8333* 0,23) * (1 - 0,18) = 11221699,551 грн.

$\Delta П_3$ = (0*500 + (25000 + 500)*(1500+1300+1000)* 0,8333* 0,23) * (1 - 0,18) = 15229449,391 грн.

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 32344898,71 грн.

5.3.2 Розрахунок ефективності вкладених інвестицій та періоду їх окупності.

Розраховуємо приведену вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т – період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0,05...0,15$;

t – період часу (в роках).

Збільшення прибутку ми отримаємо, починаючи з першого року:

ПП = (5893749,764/(1+0,1)^1)+(11221699,551/(1+0,1)^2)+(15229449,391/
/(1+0,1)^3) = 5357954,33 + 9274131,86 + 11442110,74 = 26074196,93 грн.

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \qquad (5.11)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}=2...5$, але може бути і більшим;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$PV = 2 * 1102884 = 2205768{,}45$ *грн.*

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід *(NPV, Net Present Value)* для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV, \tag{5.12}$$

$E_{абс} = 26074196{,}93 - 2205768{,}45 = 23868428{,}48$ грн.

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розра-хувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності *(IRR, Internal Rate of Return)* вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.13}$$

$T_{ж}$ – життєвий цикл наукової розробки, роки.

$E_{в} = \sqrt[3]{(1 + 23868428,48/2205768,45} - 1 = 1,278$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{5.14}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d = (0,09…0,14);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{\min} = 0,14 + 0,05 = 0,19.$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_{в}}, \tag{5.15}$$

$T_{ок}$ = 1 / 1,278 = 0,78 р.

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,78 роки, то фінансування даної наукової розробки є доцільним.

Висновки до розділу: економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 1102884 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог

і є висококонкурентоспроможним. Період окупності складе близько 0,78 роки.

ВИСНОВКИ

У сучасному світі, який все більше залежить від інформаційних технологій, безпека даних стає критично важливим питанням. Величезна кількість інформації, яка щоденно передається та зберігається в мережі, створює передумови для появи ризиків несанкціонованого доступу, крадіжки або модифікації даних. У цьому контексті шифрування даних виступає як надійний інструмент для забезпечення конфіденційності, цілісності та автентичності інформації.

Шифрування є основою для захисту персональних даних, банківських транзакцій, комунікацій в месенджерах, медичної інформації та багатьох інших сфер. Саме воно дозволяє перетворювати зрозумілі для людини дані у зашифровані символи, зрозумілі лише тим, хто володіє ключем для їх розшифрування. Це забезпечує недоступність інформації для сторонніх осіб навіть у випадку перехоплення її під час передачі через незахищені канали зв’язку.

Особливої актуальності шифрування набуває у зв’язку з розвитком кіберзлочинності, коли несанкціонований доступ до даних може мати катастрофічні наслідки. Воно захищає як від фінансових втрат, так і від загрози репутації організацій та приватних осіб. Усе це робить шифрування важливим компонентом цифрової епохи.

У цьому проекті було детально розглянуто основні методи шифрування, а також обрано та реалізовано алгоритм AES, який належить до симетричних методів. Симетричні методи шифрування, зокрема AES, забезпечують високу швидкість та ефективність обробки даних, що робить їх ідеальними для задач, де потрібна швидка та надійна робота. У той же час AES визнаний одним із найбільш захищених алгоритмів завдяки своїй стійкості до атак і відповідності сучасним стандартам безпеки.

Реалізований веб-додаток використовує бібліотеку CryptoJS для інтеграції AES, що забезпечує зручність у роботі з алгоритмом і дозволяє зберегти високу продуктивність навіть при великих обсягах даних. Було розроблено інтуїтивно зрозумілий інтерфейс для вводу тексту і ключа, що дозволяє користувачеві без зайвих складнощів шифрувати та розшифровувати інформацію.

Важливо підкреслити, що шифрування даних – це не лише спосіб захисту, а й

технологія, яка сприяє розвитку довіри до цифрових продуктів. Люди частіше обирають сервіси, які демонструють турботу про безпеку даних, а організації отримують конкурентні переваги, інтегруючи сучасні криптографічні рішення.

Цей проект став відображенням сучасних тенденцій у сфері інформаційної безпеки, показавши, як шифрування може бути реалізоване в реальному програмному продукті. Отримані знання про принципи роботи алгоритмів, їх переваги і недоліки, а також про шляхи інтеграції криптографії у веб-додатки є важливим кроком у моєму професійному розвитку.

Таким чином, шифрування даних не лише є важливим аспектом захисту інформації, але й інструментом, що сприяє стабільності цифрового середовища. Виконаний дипломний проект підтвердив актуальність і ефективність сучасних криптографічних підходів, а також продемонстрував можливості їх практичного застосування в реальних умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1.Криптографія[Електронний ресурс] Режим доступу [https://esu.com.ua/article-1576]

2. Behrouz A. Forouzan, Debdeep Mukhopadhyay Cryptography and Network Security. - Mc Graw Hill Education Private Limited, 2015. – 702 p.

3. Henk C.A. van Tilborg, Sushil Jajodia Encyclopedia of Cryptography and Security. - Springer Science & Business Media, 2014. - 1416 p.

4. Aumasson J.-P. Serious Cryptography. A Practical Introduction to Modern Encryption. –San Francisco, No Starch Press, 2017. – 312 p.

5. Шифрування AES [Електронний ресурс] CryptoJs documentation Режим доступу [https://code.google.com/archive/p/crypto-js/]

6. Розробка дизайну [Електронний ресурс] bottstrap documentation Режим доступу [https://getbootstrap.com/docs/5.3/getting-started/javascript/

7. Шифрування AES[Електронний ресурс] Режим доступу [https://www.geeksforgeeks.org/advanced-encryption-standard-aes/]

8. Алгоритм Rijndael[Електронний ресурс] Режим доступу [https://www.techtarget.com/searchsecurity/definition/Rijndael]

9. Пустовіт О., Устименко В., Про застосування алгебраїчної комбінаторики до проблем кодування та криптографії // Математичне моделювання в економіці, № 1-2. – Київ. – 2017. – С. 31-46.

10. HTML documentation [Електронний ресурс] Режим доступу [https://developer.mozilla.org/en-US/docs/Web/HTML]

11. JavaScript documentation [Електронний ресурс] Режим доступу [https://developer.mozilla.org/en-US/docs/Web/JavaScript].

ДОДАТОК А

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій та комп'ютерної інженерії
Кафедра Обчислювальної техніки
Галузь знань 12 - Інформаційні технології
Освітній рівень - магістр
Спеціальність 123 - «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н. проф Азаров О.Д.

« 17 » вересня 2024 р.

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Радловський Денис Вікторович

1 Тема проекту «Веб-додаток для шифрування даних», керівник проекту Колесник І.С., к.т.н.,доц. кафедри ОТ, затверджені наказом вищого навчального закладу від «17» вересня 2024 р. №310

2 Строк подання студентом проекту «25» грудня 2024 р

3 Вихідні дані до проекту: передавання даних http, максимальна кількість токенів 50, температура 0.5, інтерфейс користувача.

4 Зміст текстової частини (перелік питань, які потрібно розробити): вступ, огляд і аналіз актуальності теми шифрування даних, аналіз розробки програм та методів шифрування даних, вибір технологій та фреймворків, розробка веб-сервісу, розробка рекомендацій з введення в експлуатацію, економічна частина.

5 Перелік графічного матеріалу: інтерфейс реєстрації, інтерфейс програми, лістинг коду програми.

6 Консультанти розділів роботи наведені в таблиці 1.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Колесник І.С._ к.т.н.,доц. кафедри ОТ		
5	Небава М.І., к.е.н., професор каф. ЕПВМ		

7 Дата видачі завдання « 18 » вересня 2024 року

8 Календарний план виконання приведений в таблиці 2.

Таблиця 2 — Консультанти роботи

№ з/п	Назва етапів виконання бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі роботи		
2	Огляд інформаційних джерел		
3	Огляд і аналіз штучного інтелекту		
4	Розробка веб-сервісу з штучним інтелектом		
5	Підготовка матеріалів та опис розробки		
6	Оформлення пояснювальної записки та ілюстративного матеріалу		
7	Оцінка комерційного потенціалу розробки		
8	Перевірка якості виконання магістерської кваліфікаційної роботи та усунення недоліків		

Студент _____ Радловський Д.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Колесник І.С.
(підпис) (прізвище та ініціали)

Консультант з економічної частини _____Небава М.І.
(підпис) (прізвище та ініціали)

ДОДАТОК Б

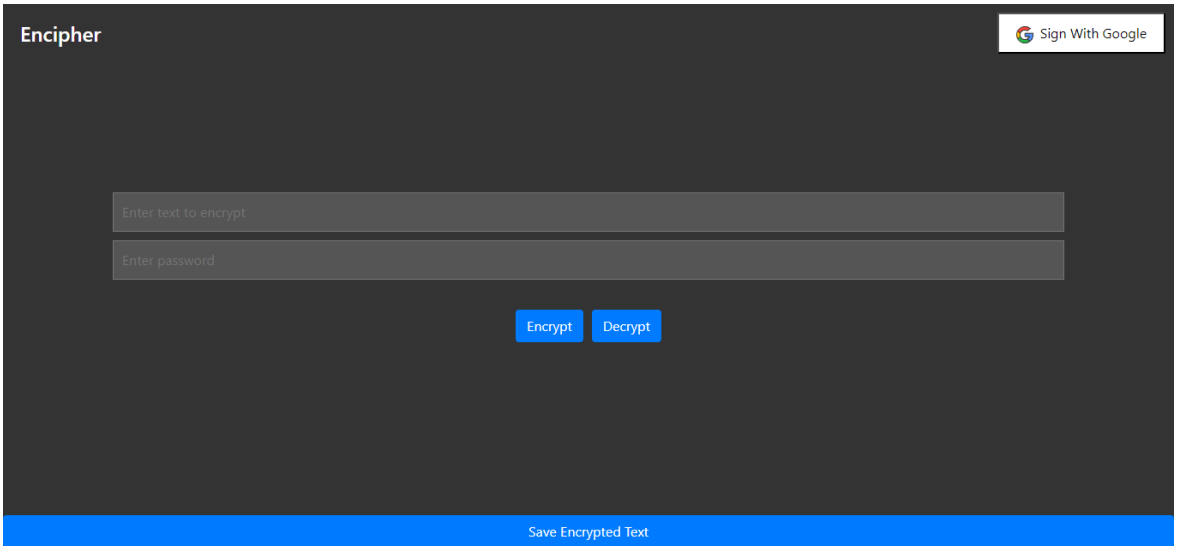

Рисунок Е.1 – Головна сторінка сайту

ДОДАТОК В

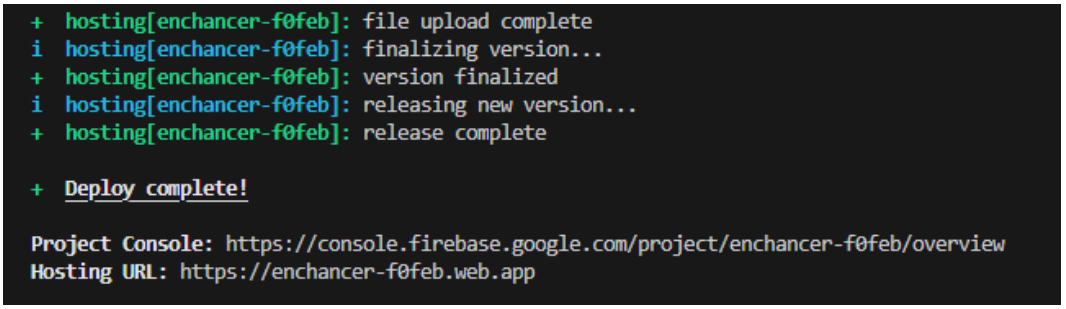

Рисунок Є.7 – Підключення до хмарного сервісу

ДОДАТОК Г

Функції Firebase

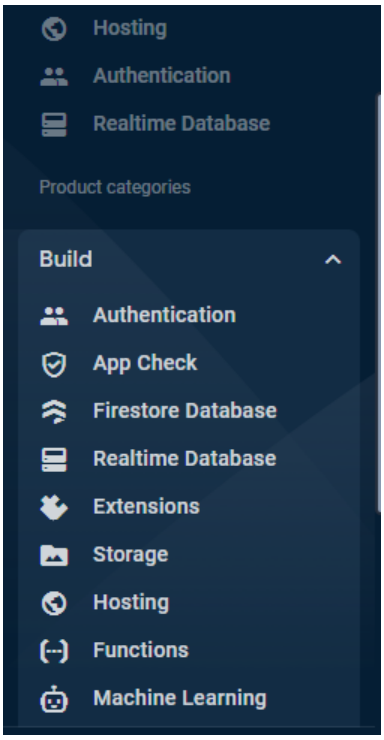

Рисунок Ж.8 – Функції Firebase

ДОДАТОК Д

Лістинг коду у файлі Index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Encipher</title>
  <link rel="stylesheet" href="styles.css">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
</head>

<body>
  <div class="container" id="main">
    <h1 class="site-title">Encipher</h1>
    <div class="input-container">
      <input type="text" placeholder="Enter text to encrypt" id="text">
      <input type="text" placeholder="Enter password" id="key" autocomplete="new-password">
    </div>
    <div class="button-container">
      <button onclick="encrypt()" class="btn btn-primary">Encrypt</button>
      <button onclick="decrypt()" class="btn btn-primary">Decrypt</button>
    </div>
    <div class="row">
      <div class="col-md-6 offset-md-3">
        <pre id="output"></pre>
      </div>
    </div>
    <button id="signInButton" class="sign-btn">
```

```
        <img class="mr-2" height="20px" src="./assets/google.svg" alt="" />Sign With Google
        </button>

    </div>
    <button id="saveDataButton" class="btn btn-primary fixed-bottom">Save Encrypted Text</button>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/crypto-fjs/4.0.0/crypto-js.min.js"></script>
    <script src="https://www.gstatic.com/firebasejs/8.2.8/firebase-app.js"></script>
    <script src="https://www.gstatic.com/firebasejs/8.2.8/firebase-auth.js"></script>
    <script src="https://www.gstatic.com/firebasejs/8.2.8/firebase-database.js"></script>
    <script src="your-firebase-config.js"></script>
    <script src="script.js"></script>
    <script>
      // Після завантаження сторінки
      window.addEventListener('load', function () {
        // Змінити тип поля вводу на "password"
        var passwordInput = document.getElementById('key');
        passwordInput.setAttribute('type', 'password');

        firebase.auth().onAuthStateChanged(function (user) {
          var signInButton = document.getElementById('signInButton');
          if (user) {
            signInButton.style.display = "none"; // Приховуємо кнопку "Sign In"
          } else {
            signInButton.style.display = "block"; // Відображаємо кнопку "Sign In"
          }
        });
```

```
    });
  </script>

</body>

</html>
```

ДОДАТОК Е

Лістинг коду у файлі script.js

```
var provider = new firebase.auth.GoogleAuthProvider();
var database = firebase.database();
var user = null; // Збереження об'єкту користувача
var userId = null; // Збереження ідентифікатора користувача
var encryptedText = null; // Збереження зашифрованого тексту
document.getElementById("signInButton").addEventListener("click", function () {
  firebase.auth().signInWithPopup(provider)
    .then((res) => {
      user = res.user;
      userId = user.uid;
      localStorage.setItem('userId', userId);
      signInButton.style.display = "none"; // Приховуємо кнопку "Sign In"
      createUserNameButton(user.displayName);
      createFileEnchancerButton();
          createSignOutButton();
    })
    .catch((err) => {
      console.log("Error Occurred");
      console.log(err.code);
      signInButton.style.display = "block"; // Відображаємо кнопку "Sign In"
    });
});

window.addEventListener('load', function() {
  // Check if there is a stored user ID
  const storedUserId = localStorage.getItem('userId');
  if (storedUserId) {
    userId = storedUserId;
    signInButton.style.display = "none";
```

```
    if (user && user.displayName) {
      createUserNameButton(user.displayName);
      createFileEnchancerButton();
      createSignOutButton();

    } else {
      firebase.auth().onAuthStateChanged(function(user) {
        if (user && user.displayName) {
          createUserNameButton(user.displayName);
          createFileEnchancerButton();
          createSignOutButton();
        } else {
          createUserNameButton("User");
        }
      });
    }
  } else {
    signInButton.style.display = "block";
  }
});


function createUserNameButton(displayName) {
  var userNameButton = document.createElement("button");
  userNameButton.classList.add("user-button");

  if (displayName) {
    userNameButton.innerHTML = displayName;
    userNameButton.addEventListener("click", function () {
      window.location.href = "savedData.html";
    });
  } else {
```

```
    userNameButton.innerHTML = "User";
  }

  document.getElementById("main").appendChild(userNameButton);
  document.getElementById("signInButton").style.display = "none";
}

function createFileEnchancerButton(){
  var fileEnchancerButton = document.createElement("button");
  fileEnchancerButton.classList.add("fileEnchancer-button");
  fileEnchancerButton.innerHTML = "File Enchancer";
  fileEnchancerButton.addEventListener("click", function () {
    window.location.href = "FileEnchancer.html";
  });
  document.getElementById("main").appendChild(fileEnchancerButton);
}

function createSignOutButton() {
  var signOutButton = document.createElement("button");
  signOutButton.classList.add("signout-button");
  signOutButton.innerHTML = "Sign Out";
  signOutButton.addEventListener("click", signOut);
  document.getElementById("main").appendChild(signOutButton);
}

function signOut() {
  firebase.auth().signOut().then(function() {
    // Sign-out successful.
    localStorage.removeItem('userId');
    window.location.reload();
  }).catch(function(error) {
    // An error happened.
```

```
    console.log(error);
  });
}

function encrypt() {
  const text = document.getElementById("text").value;
  const key = document.getElementById("key").value;


  encryptedText = CryptoJS.AES.encrypt(text, key).toString();


  const modal = document.createElement("div");
  modal.className = "modal fade";
  modal.innerHTML = `
    <div class="modal-dialog">
      <div class="modal-content">
        <div class="modal-header">
          <h5 class="modal-title">Encrypted Text</h5>
          <button type="button" class="close" data-dismiss="modal">&times;</button>
        </div>
        <div class="modal-body">
          <textarea class="form-control" rows="5" readonly>${encryptedText}</textarea>
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-primary" data-dismiss="modal">Close</button>
        </div>
      </div>
    </div>
  `;
```

```
  // Show the modal dialog.
  document.body.appendChild(modal);
  $(modal).modal("show");
}




const saveDataButton = document.getElementById("saveDataButton");
saveDataButton.addEventListener("click", function () {
  saveData(encryptedText);
});

function saveData(encryptedText) {
  // Отримуємо ідентифікатор поточного користувача
  const userId = firebase.auth().currentUser?.uid;

  // Перевіряємо, чи користувач увійшов в систему
  if (userId) {
    const userDataRef = database.ref('users/' + userId + '/encryptedTexts');

    // Генеруємо новий унікальний ключ для запису зашифрованого тексту
    const newEncryptedTextRef = userDataRef.push();

    // Встановлюємо значення зашифрованого тексту та дати для нового запису
    newEncryptedTextRef.set({
      encryptedText: encryptedText,
      timestamp: firebase.database.ServerValue.TIMESTAMP
    })
      .then(() => {
        console.log('Дані успішно збережено');
```

```
      })
      .catch((error) => {
        console.log('Помилка при збереженні даних:', error);
      });
  } else {
    console.log('Ідентифікатор користувача не знайдено');
  }
}



function decrypt() {
  const encryptedText = document.getElementById("text").value;
  const key = document.getElementById("key").value;

  try {
    const decryptedBytes = CryptoJS.AES.decrypt(encryptedText, key);
    const decryptedText = decryptedBytes.toString(CryptoJS.enc.Utf8);

    // Create a modal dialog for displaying the decrypted text.
    const modal = document.createElement("div");
    modal.className = "modal fade";
    modal.innerHTML = `
      <div class="modal-dialog">
        <div class="modal-content">
          <div class="modal-header">
            <h5 class="modal-title">Decrypted Text</h5>
            <button type="button" class="close" data-dismiss="modal">&times;</button>
          </div>
          <div class="modal-body">
            <textarea class="form-control" rows="5"
```

```
readonly>${decryptedText}</textarea>
                </div>
                <div class="modal-footer">
                  <button type="button" class="btn btn-primary" data-
dismiss="modal">Close</button>
                </div>
              </div>
            </div>
          `;

          // Show the modal dialog.
          document.body.appendChild(modal);
          $(modal).modal("show");
        } catch (error) {
          showDecryptionErrorModal();
        }
      }
```

ДОДАТОК Є

Лістинг коду у файлі savedData.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Saved Data</title>
  <link rel="stylesheet" href="styles.css">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
</head>
<body>
  <div class="container">
    <div class="header">
      <h1 class="site-title">Saved Data</h1>
      <button id="goToEncryptionButton" class="go-to-encryption-button">Go To Encryption</button>
    </div>
    <div class="row">
      <div class="col-md-6 offset-md-3">
        <div id="dataContainer"></div>
        <div class="button-container">
          <button id="historyButton" class="btn btn-primary custom-button">History</button>
          <button id="deleteHistoryButton" class="btn btn-danger custom-button">Delete History</button>
        </div>
      </div>
    </div>
  </div>
```

```
        <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
        <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
        <script src="https://www.gstatic.com/firebasejs/8.2.8/firebase-app.js"></script>
        <script src="https://www.gstatic.com/firebasejs/8.2.8/firebase-auth.js"></script>
        <script src="https://www.gstatic.com/firebasejs/8.2.8/firebase-database.js"></script>
        <script src="your-firebase-config.js"></script>

        <script>
          const database = firebase.database();
          document.getElementById('goToEncryptionButton').addEventListener('click',
function() {
            window.location.href = 'index.html';
          });

          document.getElementById('historyButton').addEventListener('click', function() {
            const userId = firebase.auth().currentUser?.uid;

            if (userId) {
              const userDataRef = database.ref('users/' + userId + '/encryptedTexts');

              userDataRef.once('value')
                .then((snapshot) => {
                  const encryptedTexts = snapshot.val();
                  var dataContainer = document.getElementById('dataContainer');
                  dataContainer.innerHTML = '';

                  if (encryptedTexts) {
                    for (const key in encryptedTexts) {
                      const encryptedText = encryptedTexts[key].encryptedText;
                      const timestamp = encryptedTexts[key].timestamp;
                      const date = new Date(timestamp).toLocaleString();
```

```
                var dataElement = document.createElement('p');
                dataElement.textContent = encryptedText + ' (додано ' + date + ')';
                dataContainer.appendChild(dataElement);
              }

              console.log('Зчитані дані:', encryptedTexts);
            } else {
              console.log('Дані не знайдено');
            }
          })
          .catch((error) => {
            console.log('Помилка при зчитуванні даних:', error);
          });
        } else {
          console.log('Ідентифікатор користувача не знайдено');
        }
      });

      document.getElementById('deleteHistoryButton').addEventListener('click',
function() {
        var dataContainer = document.getElementById('dataContainer');
        dataContainer.innerHTML = '';
      });

    </script>
  </body>

  </html>
```

ДОДАТОК Ж

Лістинг коду у файлі FileEnchancer.html

```
<!DOCTYPE html>
<html>
<head>
 <title>File enchancer</title>
 <style>
  body {
   background-color: #333333 !important;
   color: #ffffff;
   font-family: Arial, sans-serif;
   min-height: 100vh;
   display: flex;
   align-items: center;
   justify-content: center;
  }

  #main {
   text-align: center;
   position: absolute;
   top: 20px;
   left: 20px;
   right: 20px;
  }

  h1 {
   font-size: 24px;
   color: #ffffff;
   text-align: left;
   margin: 10px;
  }
```

```
.input-container {
  display: flex;
  flex-wrap: wrap;
  justify-content: center;
}

.input-container input[type="file"], .input-container input[type="text"], .input-container button {
  margin: 10px;
  background-color: #555555;
  color: #ffffff;
  border: 1px solid #777777;
  padding: 10px;
}

button:disabled {
  background-color: #777777;
  cursor: not-allowed;
}

#encryptButton {
  background-color: hsla(240, 100%, 70%, 1);
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  font-family: Arial, sans-serif;
  font-size: 16px;
  cursor: pointer;
  transition: background-color 0.3s ease;
}
```

```
#encryptButton:hover {
  background-color: hsla(240, 100%, 60%, 1);
}

#returnButton {
  position: absolute;
  top: 10px;
  right: 10px;
  background-color: #777777;
  color: #ffffff;
  padding: 10px 20px;
  box-sizing: border-box;
  max-width: 200px;
  overflow: hidden;
  text-overflow: ellipsis;
  white-space: nowrap;
  border: none;
  border-radius: 5px;
  font-family: Arial, sans-serif;
  font-size: 16px;
  cursor: pointer;
  transition: background-color 0.3s ease;
}

#returnButton:hover {
  background-color: #555555;
}

a {
  text-decoration: none;
  color: #ffffff;
```

```
        }
      </style>
    </head>
    <body>
      <div id="main">
        <h1>FileEnchancer</h1>
        <div class="input-container">
          <input type="file" id="fileInput" accept=".txt">
          <input type="text" id="encryptionKey" placeholder="Enter key">
          <button id="encryptButton">Encrypt</button>
        </div>
        <button id="downloadEncryptedButton" disabled>Download encrypted file</button>
        <button id="returnButton" class="go-to-encryption-button">Go to Encryption</button>
      </div>
      <script src="https://cdnjs.cloudflare.com/ajax/libs/crypto-js/4.0.0/crypto-js.min.js"></script>
      <script>
        document.getElementById('encryptButton').addEventListener('click', function() {
          var fileInput = document.getElementById('fileInput');
          var encryptionKey = document.getElementById('encryptionKey').value;
          var file = fileInput.files[0];
          if (file && encryptionKey) {
            var fileReader = new FileReader();
            fileReader.onload = function(event) {
              var fileContent = event.target.result;
              var encryptedContent = CryptoJS.AES.encrypt(fileContent, encryptionKey).toString();
              var downloadLink = document.createElement('a');
              downloadLink.href = 'data:text/plain;charset=utf-8,' + encodeURIComponent(encryptedContent);
```

```
            downloadLink.download = file.name + '.encrypted.txt';
            downloadLink.style.display = 'none';
            document.body.appendChild(downloadLink);


document.getElementById('downloadEncryptedButton').addEventListener('click',  function()
{
              downloadLink.click();
            });

            document.getElementById('downloadEncryptedButton').disabled = false;
          };

          fileReader.readAsText(file);
        }
      });

      document.getElementById('returnButton').addEventListener('click', function() {
        window.location.href = 'index.html';
      });
    </script>
  </body>
  </html>
```

ДОДАТОК З

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:_ Веб додаток для шифрування даних _____
Тип роботи:_____Магістерська кваліфікаційна робота_____
(кваліфікаційна робота, курсовий проект (робота), реферат, аналітичний огляд, інше (зазначити))
Підрозділ _____кафедра обчислювальної техніки_____
(кафедра, факультет (інститут), навчальна група)
Керівник_____ Колесник І.С.,доц. каф. ОТ_____
(прізвище, ініціали, посада)

Показники звіту подібності

Turnitin	
Оригінальність	93%
Загальнасхожість	7%

Аналіз звіту подібності (відмititипотрібне)

- ✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ _____Радловський Д.В._____
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

_____
_____
_____
_____

Особа, відповідальна за перевірку_____ _Захарченко С.М__
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)