МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри, канд. техн. наук,
доцент _____Є. О. Давиденко
підпис

«___» _____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ КРИПТОГРАФІЧНИХ АЛГОРИТМІВ В ІНФОРМАЦІЙНИХ СИСТЕМАХ

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 408.2011023

Здобувач _____ О.Д. Фадєєв
підпис

«__» _____2024 р.

Керівник д-р.техн.наук, професор _____ А.В. Швед
підпис

«__» _____2024 р.

Консультант канд. техн. наук, доцент _____ А. О. Алексєєва
підпис

«__» _____2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ

Зав. кафедри _____Є. О. Давиденко

«_____» _____2024 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи бакалавра

Видано студенту групи 408 факультету комп'ютерних наук

Фадєєву Олександру Дмитровичу

(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи

Дослідження ефективності криптографічних алгоритмів в інформаційних системах

Затверджена наказом по ЧНУ від «22» _грудня_ 2023 р. № 269

2. Строк представлення кваліфікаційної роботи «____» _____20___р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Очікуваним результатом є показники ефективності досліджуваних криптографічних алгоритмів захисту інформації та їх програмна реалізація. Початковими даними є вихідний масив даних, що підлягає шифруванню._____

4. Перелік питань, що підлягають розробці:

- дослідження предметної області;
- формування специфікації вимог до програмного забезпечення застосунку;
- моделювання та проєктування програмного забезпечення застосунку;
- розробка та тестування програмного забезпечення застосунку;
- аналіз результатів дослідження криптоалгоритмів.

5. Перелік графічних матеріалів

Презентація

6. Завдання до спеціальної частини

Аналіз охорони праці на робочих місцях фахівців з інформаційних технологій

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Алєксєєва А. О.	Кафедра екології	Спеціальна частина з охорони праці

Керівник роботи д-р.техн.наук, проф. Швед Альона Володимирівна

(посада, прізвище, ім'я, по батькові)

_____

(підпис)

Завдання прийнято до виконання

Фадєєв Олександр Дмитрович

(прізвище, ім'я, по батькові)

_____

(підпис)

Дата видачі завдання «____» _____2024р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: Дослідження ефективності криптографічних алгоритмів в інформаційних системах

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КРБ	31.11.2023	06.12.2023	виконано
2.	Огляд літератури за темою роботи	12.12.2023	07.03.2024	виконано
3.	Складання календарного плану КРБ	06.03.2024	07.03.2024	виконано
4.	Аналіз предметної області	29.04.2024	05.05.2024	виконано
5.	Розробка проєктних рішень	29.04.2024	05.05.2024	виконано
6.	Моделювання та конструювання ПЗ	01.05.2024	13.05.2024	виконано
7.	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	01.05.2024	10.06.2024	виконано
8.	Оформлення КРБ та презентації	30.04.2024	15.06.2024	виконано
9.	Розробка спеціальної частини з охорони праці	05.06.2024	10.06.2024	виконано
10.	Відгук керівника КРБ	15.06.2024	15.06.2024	виконано
11.	Попередній захист	5.06.2024	5.06.2024	виконано
12.	Завершення оформлення КРБ та презентації	01.05.2024	15.06.2024	виконано
13.	Рецензування	15.06.2024	15.06.2024	виконано
14.	Захист кваліфікаційної роботи	25.06.2024	25.06.2024	виконано

Розробив студент Фадєєв Олександр Дмитрович _____

(прізвище, ім’я, по батькові) (підпис)

«__» _____2024 р.

Керівник роботи д-р.техн.наук, проф. Швед Альона Володимирівна _____

(посада, прізвище, ім’я, по батькові) (підпис)

«__» _____2024 р.

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра
«Дослідження ефективності криптографічних алгоритмів в інформаційних системах»
Студента 408 групи: Фадєєв Олександр Дмитрович
Керівник: д-р.техн.наук, проф. Швед А. В.

Кваліфікаційна роботи бакалавра присвячена питанням аналізу ефективності криптографічних алгоритмів в інформаційних системах.

Актуальність теми дослідження обумовлена зростаючим інтересом по всьому світу до неспинного розвитку цифрових технологій які знаходиться ледве не в кожній частині людського життя. Кожен сервіс та додаток задля збереження власних та користувацьких даних в безпеці має передавати та отримувати їх так щоб зловмисники не могли скористуватись цим та вкрасти а саме зашифровано. Криптографія, як галузь науки, вивчає методи захисту інформації, стає ключовим інструментом у боротьбі з кіберзлочинністю та забезпеченні конфіденційності даних. Проте криптографічних алгоритмів доволі багато і їх показники можуть сильно відрізнятись один від одного. Саме тому варто знати наскільки ефективним є той чи інший алгоритм щоб правильно обрати потрібний саме під свої потреби щоб не витрачати зайві ресурси.

Об’єкт роботи – процес дослідження ефективності криптографічних алгоритмів в інформаційних системах.

Предмет роботи – методи та алгоритми криптографічного захисту інформації.

Мета роботи полягає в розробці програмного застосунку призначеного для дослідження ефективності сучасних криптографічних алгоритмів.

Кваліфікаційна робота складається з вступу, чотирьох розділів, висновків та переліку джерел посилань. У першому розділі проведено аналіз сучасних алгоритмів шифрування; проаналізовано сучасне програмне забезпечення криптографічного захисту даних; виконана специфікація вимог до програмного модулю реалізації крипто алгоритмів. У другому розділі описується процес моделювання системи

згідно з попередньо визначеними вимогами шляхом побудови діаграм, які відображають різні аспекти роботи системи на даному етапі. У третьому розділі описується процес проєктування системи. Четвертий розділ присвячено питанням програмної реалізацій системи; на основі проведеного порівняльного аналізу визначено перелік обраних технологій для реалізації програмного забезпечення; проведено тестування програмного модулю та виконано аналіз отриманих результатів.

Кваліфікаційна робота бакалавра викладена на 66 сторінки, містить чотири розділи, 49 ілюстрацій, 7 таблиць, 11 джерел в переліку посилань.

Ключові слова: RSA AES шифрування дешифрування криптографічний алгоритм.

ABSTRACT

of the Bachelor's Thesis

«Study of the effectiveness of cryptographic algorithms in information systems»

Student: Oleksandr Fadieiev

Supervisor: Doctor of Technical Sciences,

Prof. Shved Alyona

This work is devoted to the analysis of the effectiveness of cryptographic algorithms in information systems.

The relevance of the research topic is due to the growing interest all over the world in the continuous development of digital technologies, which is found in almost every part of human life. Each service and application, in order to keep personal and user data safe, must transmit and receive it in such a way that attackers cannot take advantage of it and steal it, namely encrypted. Cryptography, as a branch of science, studies information protection methods, becoming a key tool in the fight against cybercrime and ensuring data privacy. However, there are quite a lot of cryptographic algorithms and their indicators can differ greatly from each other. That is why it is worth knowing how effective this or that algorithm is in order to correctly choose the right one for your needs so as not to waste unnecessary resources.

The object of the work is the process of researching the effectiveness of cryptographic algorithms in information systems.

The subject of the work is methods and algorithms of cryptographic protection of information.

The purpose of the work is to develop a software application intended for researching the effectiveness of modern cryptographic algorithms.

The qualification work consists of an introduction, four sections, conclusions and a list of reference sources.

The introduction defines the relevance of the topic, the purpose, subject and object of the research

The first chapter examines the existing literature and theory of cryptography. It

analyzes the subject area and what algorithms there are. Also, based on the obtained results, the specification of requirements for the developed software is determined.

The second section describes the process of modeling the system according to predefined requirements by constructing diagrams that reflect various aspects of the system's operation at this stage.

The third section describes the system design based on the results obtained in the previous section. Here, a list of selected technologies for product implementation is defined, such as programming languages, databases, etc. Also included are UML diagrams that show the internal structure of the application and the interaction between its components.

The fourth chapter reviews the results of direct implementation and subsequent testing of the developed application.

The conclusions analyze the work performed and the results obtained. The bachelor's qualification work is laid out on 66 pages, contains 4 sections,

49 illustrations, 7 tables, 11 sources in the list of references.

Keywords: RSA AES encryption decryption cryptographic algorithm

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ 4
ВСТУП 5
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ 7
1.1 Аналіз проблематики сучасної інформаційної безпеки 7
1.2 Криптографічні методи захисту інформації 10
1.3 Огляд сучасних програмних систем криптографічного захисту інформації 14
1.4 Специфікація вимог до програмного забезпечення криптографічної системи 19
Висновки до розділу 1 26
2 МОДЕЛЮВАННЯ ЗАСТОСУНКУ ШИФРУВАННЯ 27
2.3 Use case діаграма 27
2.2 Діаграма взаємодії 30
2.3 Діаграма діяльності 33
2.4 Діаграма розгортання 35
Висновки до розділу 2 36
3 ПРОЄКТУВАННЯ ЗАСТОСУНКУ ШИФРУВАННЯ 37
3.1 Діаграма класів 37
3.2 Діаграма об’єктів 38
3.3 Діаграма пакетів 39
3.4 Діаграма компонентів 40
3.5 Діаграма станів та переходів 42
Висновки до розділу 3 43
4 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ 45
4.1 Вибір стеку технологій 45
4.3 Розробка backend 55
4.4 Розробка frontend 62
4.5 Аналіз ефективності алгоритмів 64
Висновки до розділу 4 66

ВИСНОВКИ.....67
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....68

ПЕРЕЛІК СКОРОЧЕНЬ

БД	–	База даних
ПЗ	–	Програмне забезпечення
ОС	–	Операційна система
AES	–	Advanced Encryption Standard
CRUD	–	Create, Read, Update, Delete
RSA	–	Rivest, Shamir Adleman
RAM	–	Random Access Memory
UML	–	Unified Modeling Language

ВСТУП

Актуальність теми дослідження обумовлена зростаючим інтересом по всьому світу до неспинного розвитку цифрових технологій які знаходиться ледве не в кожній частині людського життя. Кожен сервіс та додаток задля збереження власних та користувацьких даних в безпеці має передавати та отримувати їх так щоб зловмисники не могли скористуватись цим та вкрасти а саме зашифровано. Криптографія, як галузь науки, вивчає методи захисту інформації, стає ключовим інструментом у боротьбі з кіберзлочинністю та забезпеченні конфіденційності даних. Проте криптографічних алгоритмів доволі багато і їх показники можуть сильно відрізнятись один від одного. Саме тому варто знати наскільки ефективним є той чи інший алгоритм щоб правильно обрати потрібний саме під свої потреби щоб не витрачати зайві ресурси.

Об’єкт роботи – процес дослідження ефективності криптографічних алгоритмів в інформаційних системах.

Предмет роботи – методи та алгоритми криптографічного захисту інформації.

Мета роботи полягає в розробці програмного застосунку призначеного для дослідження ефективності сучасних криптографічних алгоритмів. Це буде впливати на загальне розуміння теми захисту даних та в тому де краще який алгоритм застосовувати.

Для досягнення поставленої мети необхідно виконати наступні завдання:

1) аналіз предметної сфери криптографічних алгоритмів;
2) аналіз існуючих аналогів програмного забезпечення криптографічного захисту інформації, визначення їх переваг та недоліків;
3) вибір криптографічних алгоритмів які будуть реалізовані в програмній системі;
4) специфікація вимог до програмного забезпечення системи;
5) моделювання та проєктування системи криптографічного захисту;

6) програмна реалізація та тестування системи криптографічного захисту;
7) аналіз отриманих результатів ефективності застосування аналізованих криптоалгоритмів.

Кваліфікаційна робота бакалавра викладена на 66 сторінки, містить чотири розділи, 49 ілюстрацій, 7 таблиць, 11 джерел в переліку посилань.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ

1.1 Аналіз проблематики сучасної інформаційної безпеки

Сучасне життя не уявляється без використання інформаційних технологій, які проникають у всі сфери нашого існування та набувають з кожним днем дедалі більшої ваги для кожної галузі роботи. Комп'ютери керують фінансовими системами, контролюють роботу на промислових об'єктах, регулюють енергопостачання по містах, відстежують рух будь якого можливого транспорту, керують самими відповідальними місіями людства і ще багато іншого. Інформаційні технології визначають ефективність та безпеку національних систем оборони та безпеки країн по всій земній кулі. Також вони забезпечують зберігання, обробку та передачу інформації, що робить їх невід'ємною частиною цифрового простору. Проте поширення інформаційних технологій викликає велику кількість питань та проблем, пов'язаних з безпекою. Зростаюча автоматизація робить суспільство залежним від стійкості, швидкодії та безпеки систем що використовуються, які впливають на благополуччя та життя ледве не усіх людей на планеті. Технічний прогрес, хоч приносить багато переваг, проте також привносить багато суперечок, проблем, питань які потрібно постійно реагувати вирішувати та регулювати їх існування щоб не допустити критичних загроз.

У сфері інформаційних технологій поширення слабко захищених комп'ютерних систем робить їх особливо уразливими перед атаками. Ця уразливість є наслідком появи нових методів нападу, що з'явилися разом зі зростанням глобальної мережі інтернет. Інциденти порушень безпеки, які мають місце через мережу, стають все більш поширеними і можуть призвести до значних матеріальних, фінансових та моральних втрат як слабко захищених користувачів так й великих корпорацій. Одним із важливих аспектів цієї проблематики є постійний розвиток нових методів захисту інформації. За останні роки з'явилися нові форми обробки та передачі даних, які потребують вдосконалення захисту. Однак, навіть найкращі криптографічні

алгоритми не можуть гарантувати повну безпеку, оскільки постійно виникають нові загрози, вразливості та канали витоку та перехоплення інформації якими користуються зловмисники мотивуючи через це розробляти відповідні виправлення у вже існуючих рішеннях чи робити абсолютно нові бо старі вже не представляють булої перешкоди перед отриманням даних.

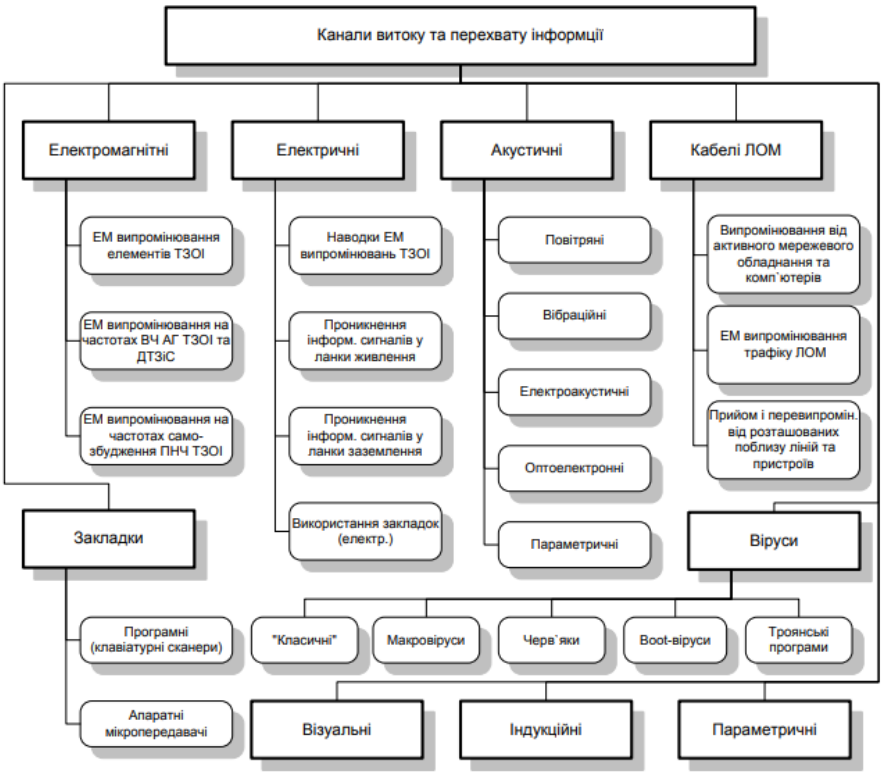


Рисунок 1.1 - Класифікація каналів витоку та перехвату інформації

Зрештою, важливо розуміти, що безпека інформації є складною та постійно змінюваною проблемою. Єдине стабільне рішення полягає у постійному вдосконаленні криптографічних алгоритмів, розробці нових методів захисту та посиленні обізнаності користувачів з питань безпеки інформації. Набуття та оброблення інформації за допомогою комп'ютерів та інших технічних засобів зростає, що включає не лише збільшення обсягів, але й розширення способів її зберігання та доступу. Завдяки цьому, здатність отримувати доступ до різних даних стає набагато простішою. Сучасні комп'ютери набули великого масштабу, а для того щоб охопити

більший ринок прийшлось піти на ряд спрощень користування ними що призвело до зниження середньої кваліфікації користувачів і ускладнило захист системи, оскільки багато з них не здатні підтримувати високий рівень безпеки.

Розвиток апаратного та програмного забезпечення викликає зростання проблем безпеки, оскільки багато програм мають вади і не відповідають мінімальним вимогам безпеки, надаючи зловмисникам широкі можливості для атак. Розширення віртуальних машин і інтерпретаторів створює нові шляхи для використання комп'ютерних програм для атак на системи, ускладнюючи захист. Проблема з вірусами та іншими шкідливими програмами продовжує поглиблюватися, оскільки з'являються нові види атак, спрямовані на сервери та інші комп'ютерні системи, що розширює коло потенційних жертв. Проблема зловмисних програм, таких як віруси, хробаки та троянські програми та інші види витоку інформації, за останні роки лишається актуальною, і її обсяг продовжує зростати, незалежно від того, яким чином оцінювати її: за кількістю випадків чи за завданими збитками.

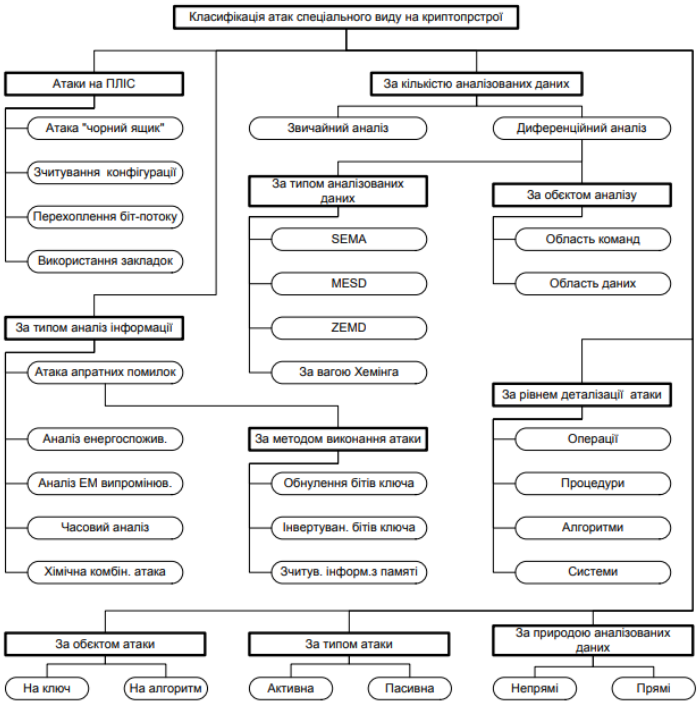


Рисунок 1.2 - Узагальнена класифікація атак спеціального виду на криптопристрої

Ця проблема поглиблюється, хоча великі компанії та організації намагаються боротися з нею із збільшеними видатками. Тому якщо не вирішувати цю проблему забезпеченням даних належним захистом то це може стати критичною загрозою в подальшому поширені інформаційних технологій у сфері критичних систем, що обробляють важливу інформацію.

1.2 Криптографічні методи захисту інформації

Алгоритм Advanced Encryption Standard (AES), запропонований як наступник DES, є одним з найбільш широко використовуваних алгоритмів симетричного шифрування даних. Він був прийнятий Національним інститутом стандартів і технологій США (NIST) у 2001 році після ретельного процесу відбору, що тривав кілька років. AES вирізняється своєю безпекою, швидкістю та гнучкістю, що робить його основним вибором для шифрування даних у багатьох додатках.

Основні характеристики AES

1) **Блокова структура**: AES працює з блоками даних розміром 128 біт. Це означає, що дані розбиваються на блоки по 128 біт, які шифруються окремо.

2) **Довжина ключа**: AES підтримує три різні довжини ключа: 128, 192 та 256 біт. Кожен варіант забезпечує різний рівень безпеки та вимагає різної кількості раундів шифрування:

- AES-128: 10 раундів
- AES-192: 12 раундів
- AES-256: 14 раундів

3) **Раунди шифрування**: Кожен раунд шифрування складається з кількох основних операцій:

- **SubBytes**: Нелінійне заміщення кожного байта даних на відповідний байт з фіксованої таблиці замін (S-box).
- **ShiftRows**: Циклічне зсування рядків масиву даних.
- **MixColumns**: Лінійне перетворення стовпців даних.

– **AddRoundKey**: Комбінування кожного байта даних з ключем раунду за допомогою операції XOR.

4) **Ключовий розклад (Key Schedule)**: Процес генерації всіх раундових ключів з початкового ключа шифрування. Цей процес забезпечує високу складність і захист від криптоаналітичних атак.

Переваги AES

– **Безпека**: AES має високий рівень стійкості до відомих атак, таких як лінійний та диференціальний криптоаналіз. Завдяки своїй структурі та кількості раундів, алгоритм забезпечує надійний захист даних.

– **Ефективність**: AES може бути ефективно реалізований як у програмному, так і в апаратному забезпеченні. Це робить його швидким і економічним у використанні ресурсів.

– **Гнучкість**: Можливість вибору довжини ключа дозволяє адаптувати рівень безпеки залежно від потреб користувача.

Застосування AES

AES використовується у багатьох сферах, включаючи:

– **VPN і мережеві протоколи**: Шифрування даних, що передаються через інтернет.

– **Файлові системи**: Шифрування файлів і дисків для захисту конфіденційної інформації.

– **Бездротові мережі**: Протоколи безпеки Wi-Fi, такі як WPA2 і WPA3, використовують AES для захисту переданих даних.

– **Мобільні пристрої**: Захист даних на смартфонах і планшетах.

Технічні деталі

AES базується на концепції заміщення та перестановки (SP-мережа). Це означає, що дані піддаються ряду операцій заміщення (SubBytes) та перестановки (ShiftRows і MixColumns), які разом забезпечують високий рівень дифузії та плутанини,

необхідних для надійного шифрування.

Крім того, кожен раунд починається з додавання ключа раунду (AddRoundKey), що забезпечує додатковий рівень безпеки. Процес розширення ключа (Key Expansion) створює унікальні ключі для кожного раунду, що ускладнює криптоаналіз.

У підсумку, AES є надзвичайно потужним і гнучким інструментом для захисту даних, який широко використовується в різних галузях завдяки своїй ефективності та високому рівню безпеки.

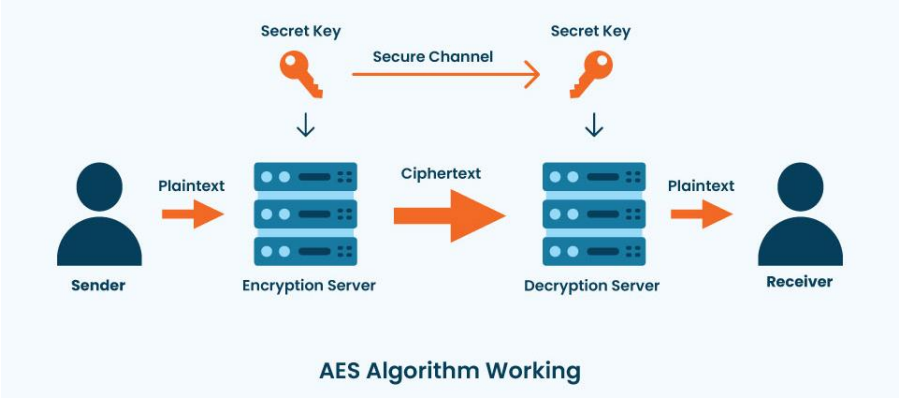


Рисунок 1.3 – Візуалізація роботи AES

RSA (названий на честь його винахідників Рональда Рівеста, Аді Шаміра і Леонарда Адлемана) є одним з найпоширеніших асиметричних алгоритмів шифрування і широко використовується для захисту даних в Інтернеті. Алгоритм RSA базується на математичній складності факторизації великих чисел, що забезпечує його криптографічну стійкість.

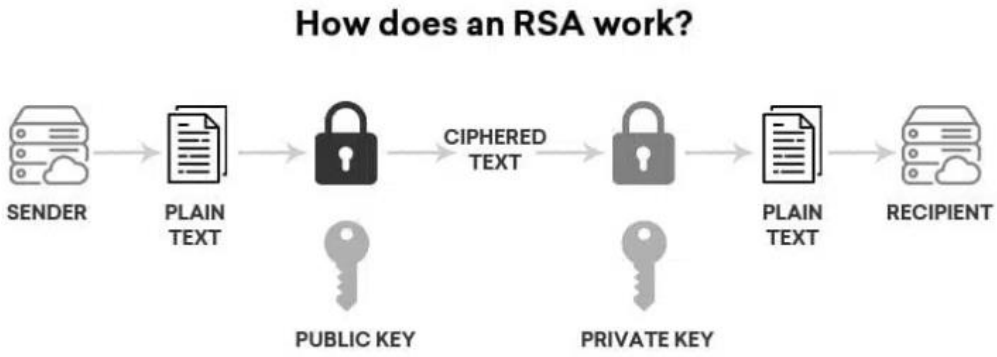


Рисунок 1.4 – Візуалізація роботи RSA

Основні Характеристики RSA

1) **Тип алгоритму**: асиметричний алгоритм шифрування, що використовує пару ключів — відкритий та закритий.

2) **Довжина ключа**: зазвичай 1024, 2048 або 4096 біт.

3) **Математичні основи**: RSA базується на операціях з великими простими числами і факторизації.

Генерація Ключів

Процес генерації ключів для RSA включає наступні етапи:

1) **Вибір двох великих простих чисел (p і q)**: Ці числа повинні бути достатньо великими, щоб забезпечити високий рівень безпеки.

2) **Обчислення n**: $n=p\times qn=p\times q$. Це число використовується як модуль для обох ключів.

3) **Обчислення функції Ейлера φ(n)**: $\varphi(n)=(p-1)\times(q-1)\varphi(n)=(p-1)\times(q-1)$.

4) **Вибір відкритого експонента e**: Вибирається число e, що є взаємно простим з φ(n) і зазвичай має невелике значення, наприклад 65537, щоб полегшити обчислення.

5) **Обчислення закритого експонента d**: d є мультиплікативною оберненою до e за модулем φ(n). Це означає, що $d\times e\equiv 1(\mathrm{mod}\varphi(n))d\times e\equiv 1(\mathrm{mod}\varphi(n))$.

Пара (e, n) є відкритим ключем, а пара (d, n) є закритим ключем.

Процес Шифрування і Дешифрування

1) **Шифрування**: Повідомлення m шифрується з використанням відкритого ключа (e, n) за допомогою формули:

$c=me\mathrm{mod}\ nc=me\mathrm{mod}n$

де c — це зашифрований текст.

2) **Дешифрування**: Зашифрований текст дешифрується з використанням закритого ключа (d, n) за допомогою формули:

$m=cd\mathrm{mod}\ nm=cd\mathrm{mod}n$

де m — це розшифроване повідомлення.

Застосування RSA: RSA дозволяє створювати цифрові підписи, що забезпечують

аутентифікацію та цілісність даних. Можна задіяти з симетричними алгоритмами.

1.3 Огляд сучасних програмних систем криптографічного захисту інформації

Під час початкового етапу розробки будь-якого продукту надзвичайно важливо вивчити вимоги до ПЗ та існуючі аналоги на ринку. Аналізуючи існуючі пропозиції, можна виявити основні їх переваги та використовувати ці знання для подальшого вдосконалення свого продукту. Також це допомагає визначитись у інших критеріях які будуть впливати на основні рішення при створенні власного ПЗ що значно прискорить час виконання поставленої задачі та надасть належне розуміння того як це бачать інші.

Було проаналізовано аналогічні веб-додатки, щоб виявити основні переваги, що будуть використані для покращення розроблюваного застосунку. Були розглянуті наступні аналоги криптографічного шифрування Crypto Tools (табл. 1.1), CyberChef (табл. 1.2), Cryptii (табл. 1.3).

Crypto Tools - це онлайн-набір інструментів для криптографічних операцій, розроблений Кавехом Бахтіярі на мові Python [1]. Програма включає в себе шифрування, дешифрування, генерацію ключів, підпис та перевірку цифрових підписів, обчислення хеш-функцій та інше. Він створений для допомоги користувачам з різними криптографічними потребами. Застосунок представляє в собі широкий вибір різних інструментів для виконання різних криптографічних операцій, що покривають багато аспектів криптографії. Простота використання інтерфейсу інструментів дуже інтуїтивна та проста у використанні, навіть для початківців, з інтуїтивно зрозумілою навігацією та простим доступом до функціоналу.

Таблиця 1.1 – Опис вебзастосунку Crypto Tools

Назва	**Crypto Tools**
Архітектура	Web application
Виробник	Кавех Бахтіярі
Мова реалізації	Python

Кінець таблиці 1.1

Функції	– шифрування файлів; – розшифрування файлів; – шифрування тексту; – розшифрування тексту; – вибір алгоритму шифрування; – генерація випадкових ключів; – перевірка ключів; – створення цифрових підписів; – перевірка цифрових підписів; – хешування даних.
Переваги	– гарний та зрозумілий дизайн сайту; – безкоштовність; – широкий спектр основних функцій; – відкритий код.
Недоліки	– обмежені можливості;
Посилання	https://crypto-tools.streamlit.app/

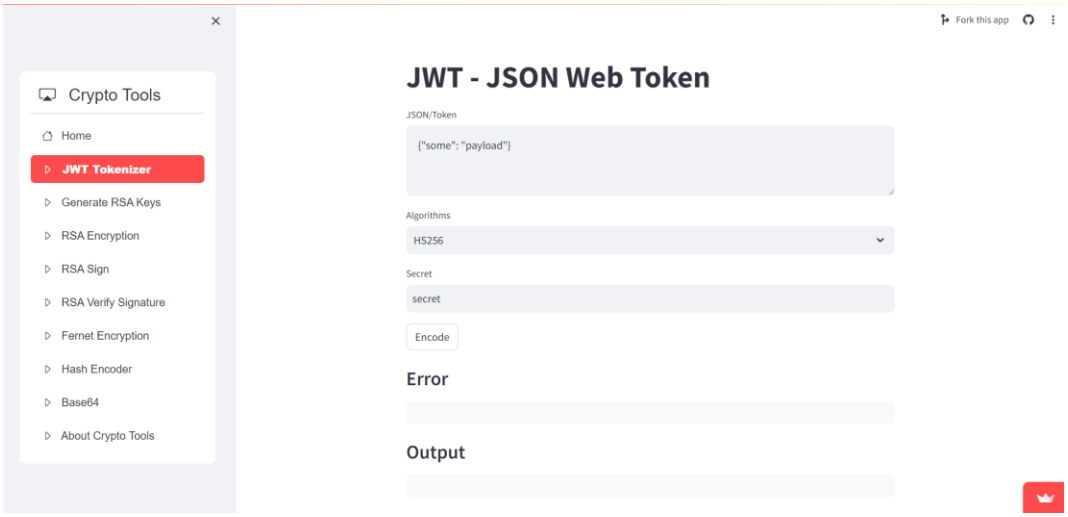


Рисунок 1.5 – Вигляд інтерфейсу застосунку «Crypto Tools»

CyberChef це веб-інструмент, що дозволяє аналізувати, обробляти та перетворювати дані шляхом комбінації різних операцій [2]. Він призначений для роботи з широким спектром завдань з обробки даних, включаючи шифрування, дешифрування, стискання, розпакування, перетворення форматів тощо. На сайті представлено велику кількість криптографічних алгоритмів які постійно допрацьовуються та модифікуються при підтримці Центру урядового зв'язку Великої Британії. Інтерфейс розроблено з простотою в основі. Складні прийоми тепер такі ж тривіальні, як перетягування. Окрім цього застосунком передбачено те щоб прості функції можна було об’єднати, щоб створити «рецепт», що потенційно призведе до складного аналізу, яким можна поділитися з іншими користувачами та використати разом із їхніми даними. До того ж для тих, кому зручно писати код, CyberChef — має швидкий і ефективний спосіб створити прототип рішень проблеми, який потім можна створити в сценарії, коли буде доведено, що воно працює.

Таблиця 1.2 – Опис вебзастосунку CyberChef

Назва	**CyberChef**
Архітектура	Web application
Виробник	Government Communications Headquarters
Мова реалізації	JavaScript
Функції	– шифрування та дешифрування; – маніпулювання текстом; – обробка даних; – візуалізація даних; – конвертація форматів.

Кінець таблиці 1.2

Переваги	– багатофункціональність; – зручний інтерфейс; – локальна обробка даних; – безкоштовний та відкритий код; – велика кількість алгоритмів; – особливий функціонал.
Недоліки	– обмежена підтримка; – необхідність Інтернет-з'єднання; – вимоги до продуктивності.
Посилання	https://gchq.github.io/CyberChef/

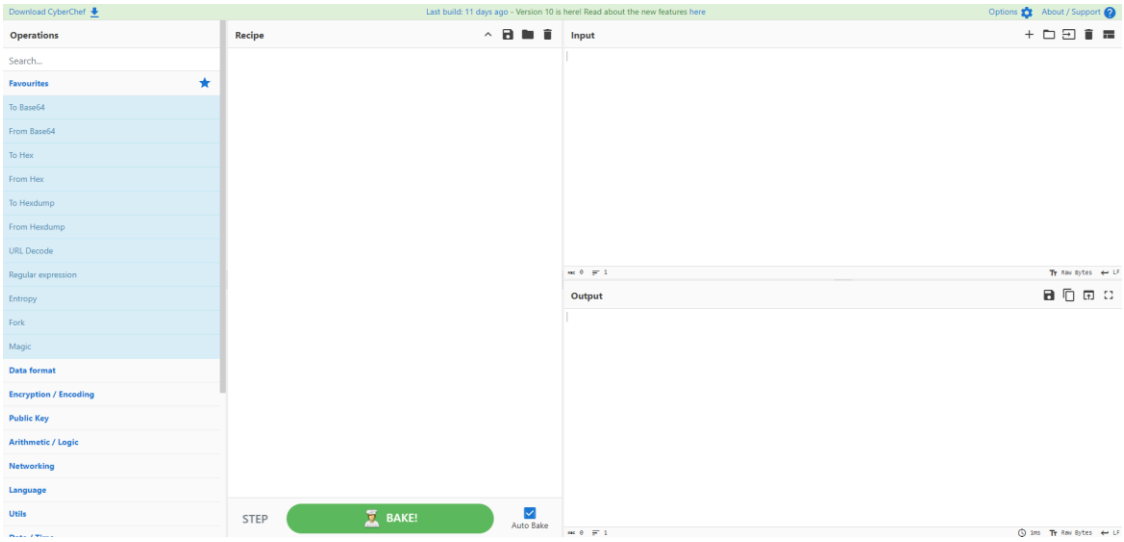


Рисунок 1.6 – Вигляд інтерфейсу застосунку «CyberChef»

Cryptii – це онлайн-інструмент, який надає широкий спектр функцій для роботи з шифруванням та дешифруванням, конвертацією тексту, генерацією ключів та створенням та перевіркою цифрових підписів [3]. Цей інструмент створений з метою надання користувачам простого та зручного способу виконання різноманітних криптографічних операцій без необхідності встановлення спеціалізованих програм чи покупки платних рішень.

Таблиця 1.3 – Опис вебзастосунку Cryptii

Назва	**Cryptii**
Архітектура	Web application
Виробник	Fränz Friederes
Мова реалізації	JavaScript
Функції	– шифрування та дешифрування; – конвертація тексту; – генерація ключів; – створення та перевірка цифрових підписів.
Переваги	– широкий вибір старих та нових шифрів; – інтуїтивний інтерфейс; – підтримка різних форматів введення та виведення.
Недоліки	– обмежені можливості в порівнянні зі спеціалізованими програмами; – питання безпеки даних через онлайн доступ; – обмеження на обробку великих обсягів даних.
Посилання	https://cryptii.com/

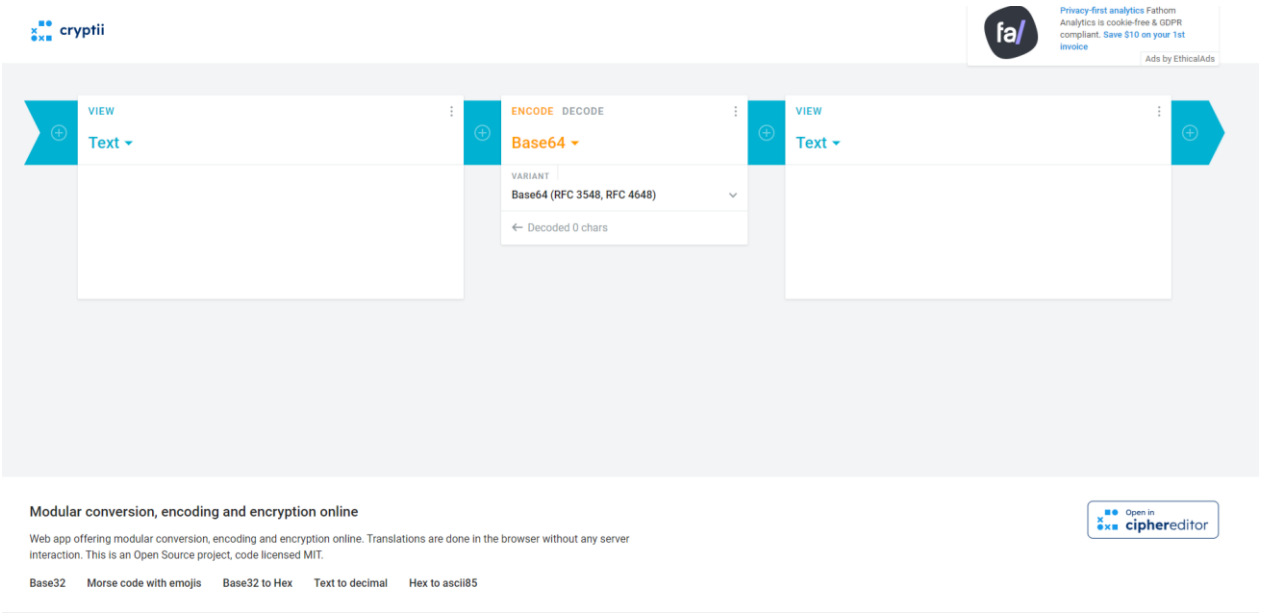


Рисунок 1.7 – Вигляд інтерфейсу застосунку «Cryptii»

Фактори на які було звернуто особливу увагу:

– доступність для користувачів у яких не має базових знань криптографії;
– доступність для широкої кількості алгоритмів;
– актуальність розробок;
– простота використання;
– інтуїтивний інтерфейс;
– доступність на будь-якій платформі.

1.4 Специфікація вимог до програмного забезпечення криптографічної системи

Вебзастосунок буде призначений для демонстрації роботи двох ключових криптографічних алгоритмів - RSA та AES. Користувачам буде надана можливість вводити текст або завантажувати файли для шифрування, використовуючи обрані алгоритми. Крім того, вебзастосунок автоматично вимірюватиме час, необхідний для шифрування, а також використання оперативної пам'яті (RAM) для обох алгоритмів. Отримані результати будуть збережені до бази даних, а користувачі зможуть переглядати їх у відповідній вкладці на сайті. Такий підхід дозволить користувачам не лише ознайомитись з роботою криптографічних алгоритмів, але й отримати об'єктивні дані щодо їхньої продуктивності та ефективності.

Таблиця 1.4 – Опис системи що розробляється

Основні задачі	Шифрування даних та замір витрачених ресурсів
Користувачі системи	1. зареєстрований користувач; 2. гість.
Сценарії роботи системи	1. вибір криптографічного алгоритму; 2. збір результатів шифрування.

Кінець таблиці 1.4

Засоби апаратної та програмної реалізації	1. SQL Server; 2. C++; 3. React; 4. Node.js; 5. Python.
Вихідні дані	1. звіти про результати шифрування та витрачених ресурсів.

В наступному рисунку показано мокап сторінки AES алгоритму в якому є поля: вставки тексту, тип шифру, таємного ключа, розмір в бітах, те в якому форматі буде вихідний текст, поле яке видасть користувачу зашифровану текстову інформацію та показники витраченого часу та оперативної пам’яті.

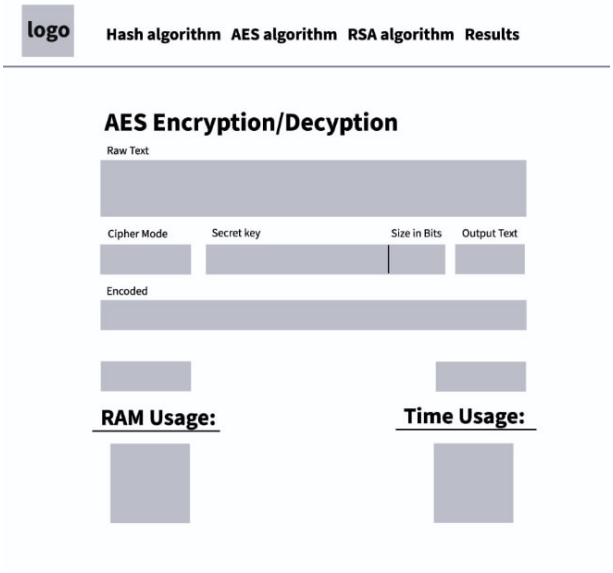


Рисунок 1.8 – Вигляд mock-up інтерфейсу сторінки

У цьому мокапі можна бачити вигляд сторінки шифру RSA в якому так само як в попередньому алгоритмі існують поля: введення інформації, отримання її зашифрованого вигляду, тип шифру, формат вихідного тексту, використання часу та оперативної пам’яті, та додається поле введення приватного чи публічного ключа.

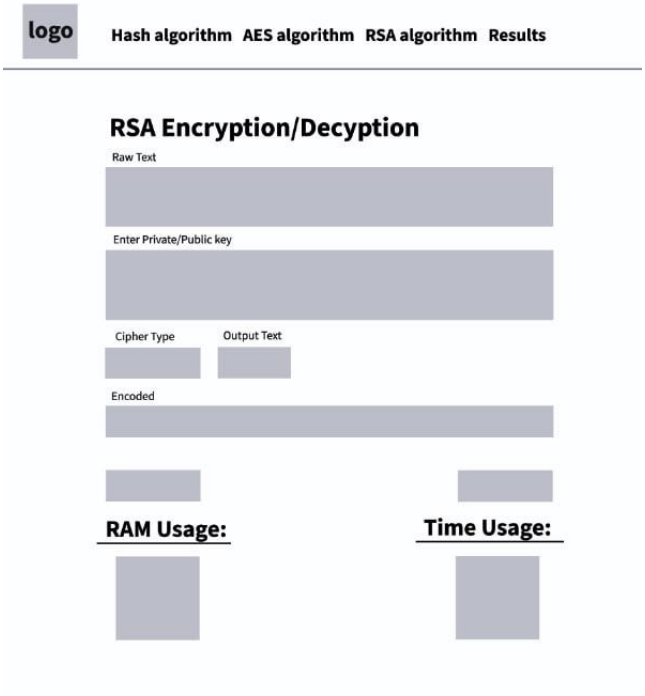


Рисунок 1.9 – Вигляд mock-up інтерфейсу сторінки

В цьому mock-up інтерфейсу можна побачити те як буде вигляди місце зберігання всіх результатів роботи алгоритмів та те скільки на них було витрачено ресурсів що буде подано у вигляді таблиці.

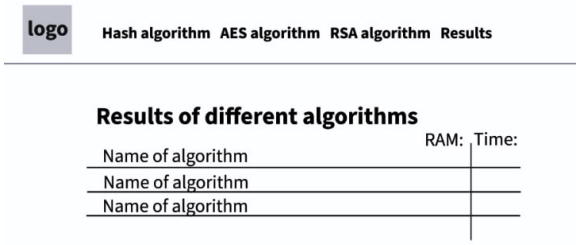


Рисунок 1.10 – Вигляд mock-up інтерфейсу сторінки

Mock-up було створено за допомогою програмного забезпечення «Photoshop». Загалом користувачу надаються наступні розділи застосунку:

1) головна сторінка(вона відповідає за вибір типу шифрування);
2) сторінка реєстрації аккаунту;

3) вибір шифрування чи дешифрування за допомогою AES;
4) вибір шифрування чи дешифрування за допомогою RSA;
5) результати шифрувань.

ПРИЗНАЧЕННЯ ТА МЕЖІ ПРОЄКТУ

Призначення системи (застосунку), для якого розробляється ПЗ

Призначенням застосунку є шифрування та дешифрування криптографічними алгоритмами RSA та AES з подальшим заміром швидкодії.

Погодження, що ухвалені в програмній документації

Було погоджено, що для створення ПЗ було використано мову програмування C++.

Межі проєкту ПЗ

Крайня дата завершення роботи над ПЗ – 22.06.2024р.

ЗАГАЛЬНИЙ ОПИС

Сфера застосування:

Вебзастосунок призначений для шифрування та дешифрування файлів та текстової інформації за допомогою RSA та AES алгоритмів.

Характеристика користувачів

Основні характеристики користувачів: наявність пристрою (комп’ютер, ноутбук, телефон тощо), доступ до Інтернету.

Загальна структура та склад системи.

Система складається з наступних частин:

1) Клієнтська частина (Front-end):
   - дизайн та інтерфейс вебсайту;
   - графічний контент та мультимедійні елементи;
   - клієнтська програмна логіка (React):
2) Серверна частина (Back-end):
   - хостинг та база даних;

– серверна логіка (Node.js);

– інтерфейс взаємодії з базою даних.

3) База даних:

– збереження аккаунта користувача та результатів роботи алгоритмів;

– взаємодія з БД через ORM.

Загальні обмеження

Основне обмеження у використанні застосунку – доступ до інтернету.

ФУНКЦІЇ СИСТЕМИ

Шифрування та дешифрування інформації з заміром часу

Опис функції

Функція призначена для роботи з шифруванням та дешифруванням інформації та отримання результатів як самого алгоритму так й витрачених на те ресурсів.

Вхідна і вихідна інформація

Вхідна інформація – файл, текст.

Вихідна інформація – зашифроване повідомлення.

Функціональні вимоги

База даних, що зберігає результати та доступ до Інтернету.

Опис функції

Створення аккаунту та вхід в нього

Вхідна і вихідна інформація

Вхідна інформація – логін та пароль.

Вихідна інформація – повідомлення про успішну реєстрацію чи вхід.

Функціональні вимоги

База даних, що зберігає дані користувача та доступ до Інтернету.

ВИМОГИ ДО ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

Джерела і зміст вхідної інформації (даних)

В цьому ПЗ вхідні дані отримуються від користувача, що, заповнюючи поля надає програмі інформацію з якою вона буде працювати.

Нормативно-довідникова інформація (класифікатори, довідники тощо)

Вимоги до даного пункту відсутні.

Вимоги до способів організації, збереження та ведення інформації

Обмін даними відбувається через обробку запитів у основній програмі та використанням БД – MySQL.

ВИМОГИ ДО ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

Для розробки програмного забезпечення немає значних технічних обмежень. Але для підтримки необхідних програм та комфортної розробки бажано мати комп’ютер або ноутбук з оперативної пам’яті місткістю не менше 8 Гбайт.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Архітектура програмної системи

Система складається з: клієнтської частини, серверної частини та БД.

Системне програмне забезпечення

Застосунок побудований з використанням C++. Для написання клієнтської частини були використані: React. Для серверної частини: Node.js. В якості БД для застосунку обрано MySQL.

Мережне програмне забезпечення

Під час розробки було використано ОС Windows 10, для написання коду було використано Visual Studio, для перегляду застосунку веббраузер Google Chrome.

Програмне забезпечення ведення інформаційної бази

За допомогою ORM та CRUD-операцій виконується маніпуляції з БД.

Мова і технологія розробки ПЗ

Програмне забезпечення має розроблюватися з використанням React та C++.

ВИМОГИ ДО ЗОВНІШНІХ ІНТЕРФЕЙСІВ

Інтерфейс користувача

Інтерфейс має бути інтуїтивно зрозумілим користувачеві, для цього були використані стандартні рішення при верстці.

Апаратний інтерфейс

Програмний застосунок має бути доступним на будь-якому пристрої з будь-яким розширенням екрану, тому інтерфейс має бути адаптивним.

Програмний інтерфейс

Клієнтська частина веб-додатка буде розроблена за допомогою бібліотеки React, що забезпечить динамічність та інтерактивність інтерфейсу.

Серверна частина буде розроблена з використанням середовища виконання Node.js та фреймворку Express.js для створення веб-сервера.

Інтерфейс бази даних: Збереження деталей про результати шифрувань для подальшого аналізу і перегляду.

Комунікаційний інтерфейс

При хостингу вебзастосунку використовується протокол HTTP з криптографічним протоколом SSL

ВЛАСТИВОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Доступність

Інтерфейс простий для користувачів всіх рівнів, без потреби в спеціалізованих знаннях для виконання тестів.

Супроводжуваність

Вебзастосунок не потребує супроводжуваності з боку розробника, окрім моментів з заміною деталей та стилю дизайну застосунку, за бажанням замовника.

Продуктивність

Продуктивність залежить від якості Інтернет-з’єднання.

Надійність

Пароль користувача хешований.

Безпека

Доступ до редагування є лише у адміністратора.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи бакалавра було проведено огляд та аналіз існуючих аналогів застосунків. Це дозволило виділити ключові переваги цих інструментів та набути розуміння, які саме з них варто впроваджувати у власний програмний продукт. Крім того, був проведений аналіз системи, що розробляється, і сформований перелік основного функціоналу. Цей функціонал включає модулі для реєстрації, шифрування та інтерфейсів для користувачів. Також були описані сценарії роботи системи та визначені технології, які будуть використовуватися для реалізації системи.

2 МОДЕЛЮВАННЯ ЗАСТОСУНКУ ШИФРУВАННЯ

2.3 Use case діаграма

Use case або сценарії використання - це послідовність дій, які потрібно виконати користувачеві для ефективної взаємодії з системою. Вони описують взаємодію між системою та зовнішніми учасниками, демонструючи, як система використовується для досягнення певних цілей або завдань, без зайняття технічних аспектів програмної реалізації.

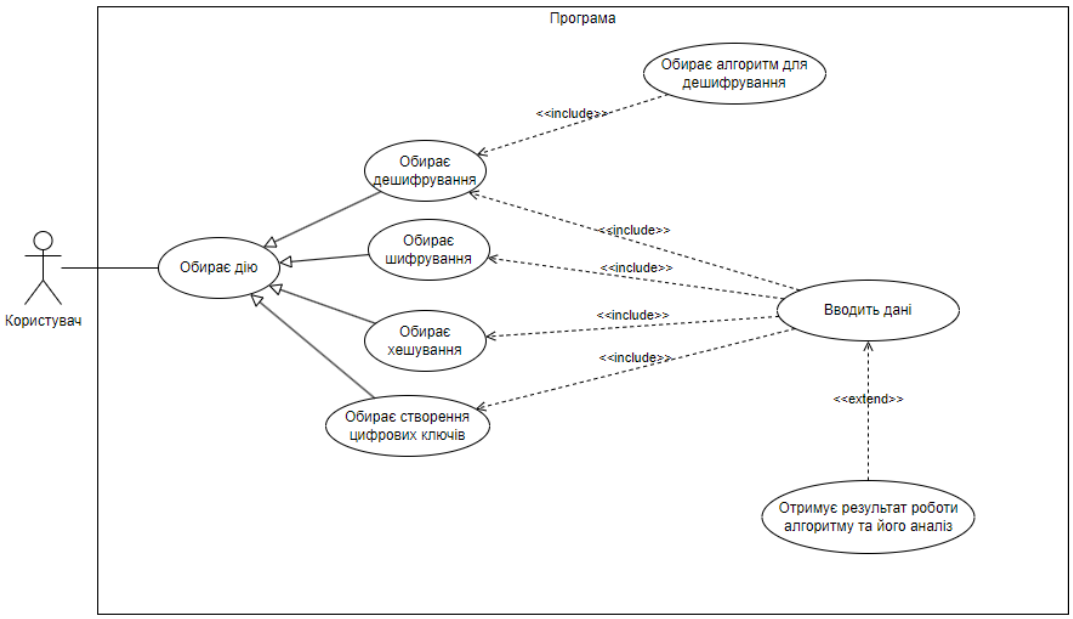


Рисунок 2.1 – Діаграма прецедентів

З цієї діаграми можна переглянути усі функції користувача який маніпулює представленим йому функціоналом на сторінці.

Короткий use case

Користувач заходить на сайт. Реєструється на сайті. Обирає криптографічний алгоритм. Вводить дані чи обирає файл який хоче використати. Отримує результат роботи алгоритму. Отримує результат використання RAM. Отримує результат витраченого часу. Якщо під час роботи чи налаштування алгоритму виникає помилка то на сторінці з’являється відповідне повідомлення про те що це за помилка і чим вона викликана.

Поверхневий use case

Головний сценарій (успішний):

Користувач отримав зашифрований текст та інформацію про те скільки часу та RAM витратив обраний алгоритм на це.

Альтернативні сценарії:

Користувач обирає інший алгоритм шифрування. Отримує інший результат та інформацію про витрати RAM та часу

Користувач обирає зашифровану інформацію. Отримує результат розшифрування

Невалідні дані для шифрування. Користувач вводить невалідні дані для шифрування, наприклад, порожній рядок

Непідтримуваний алгоритм хешування. Користувач обирає алгоритм хешування, який не підтримується на даному сайті. Система повідомляє про це користувача та рекомендує обрати інший алгоритм.

У випадку повільного завантаження сторінки, система реєструє це як попередження і пропонує можливі рекомендації для її оптимізації.

У разі, якщо сторінка не завантажується або відображає помилки, це реєструється як критична помилка, та користувача інформують про необхідність перевірки конфігурації сервера або коду сторінки.

Таблиця 2.1 – Usecase Повноцінний

Usecase section	**Comment**
Use Case Name	Шифрування даних
Scope	System
Level	User-goal
Primary Actor	Користувач

Продовження таблиці 2.1

Stakeholders and interests	Користувач
Preconditions	Шифрування почалось
Success guarantee	Дані було успішно зашифровані
Main Success Scenario	1. користувач запускає програму; 2. програма відкриває головне вікно; 3. користувач обирає тип шифрування RSA або AES; 4. користувач вводить текст або обирає файл для шифрування; 5. програма застосовує обраний алгоритм шифрування до введеного тексту або файлу; 6. зашифровані дані виводяться на екран або зберігаються у файлі; 7. користувач отримує повідомлення про успішне завершення операції; 8. програма завершує роботу.
Extensions	1. якщо користувач вибирає RSA або AES, але не вводить текст або не вибирає файл, програма показує повідомлення про помилку; 2. якщо користувач вибирає файл для шифрування, але файл не існує або не може бути відкритий, програма показує повідомлення про помилку;

Кінець таблиці 2.1

Extensions	3. якщо під час шифрування виникають технічні проблеми або помилки, програма показує повідомлення про помилку та пропонує спробувати знову або звернутися до технічної підтримки.
Special Requirements	Дані для шифрування
Technology and Data Variations List	Мати доступ до інтернет та даних для шифрування
Frequency of Occurrence	Під час запуску шифрування

2.2 Діаграма взаємодії

Діаграма взаємодії, або діаграма послідовності, є важливим інструментом в моделюванні програмного забезпечення та систем. Вона наочно показує, як об'єкти або компоненти системи взаємодіють один з одним з плином часу.

Таблиця 2.2 – Опис можливих компонентів діаграми взаємодії

Елемент	**Опис**
Актори	Актори – це сутності, які взаємодіють із системою та можуть бути представлені у верхній частині діаграми.
Об’єкти	Об’єктами є екземплярами класів у системі та представлені прямокутниками на діаграмі.
Лінії життя	Лінії життя представляють час існування об’єкта системі та відображаються у вигляді ліній, що тягнутьс від верхньої частини вікна об’єкта.

Кінець таблиці 2.1

Лінії життя	Лінії життя представляють час існування об’єкта у системі та відображаються у вигляді ліній, що тягнуться від верхньої частини вікна об’єкта.
Повідомлення	Повідомленням це взаємодія між об’єктами, які відображаються стрілками між лініями життя.
Смуги активації	Смуги активації представляють час, протягом якого об’єкт виконує метод та відображаються як горизонтальні смуги на лінії життя.
Обмеження	Обмеження використовуються для визначення умов або обмежень, які повинні бути виконані для того, щоб відбулося повідомлення або взаємодія.

В наступній діаграмі можна бачити як відбувається вибір алгоритму шифрування в коді. Користувач обирає яким він хоче користуватись алгоритмом після чого у відповідь приходить підтвердження на його дію. Потім користувач має можливість обрати що він буде робити після чого повертається результат.

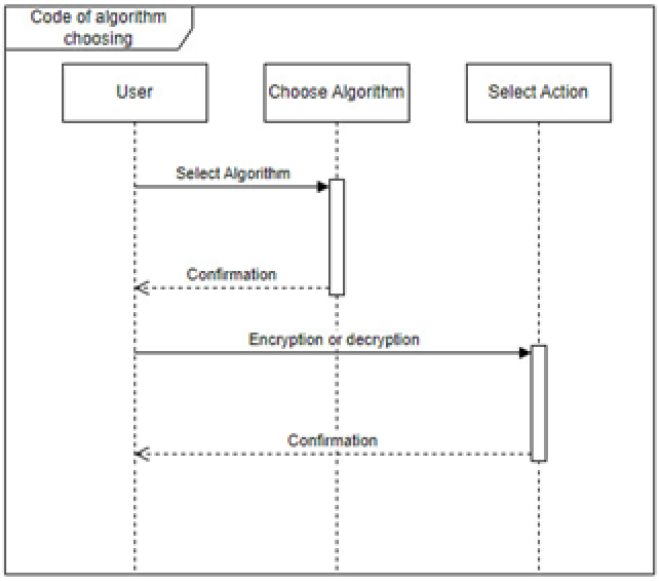


Рисунок 2.2 – Діаграма взаємодії вибору алгоритму

В наступній діаграмі продемонстровано роботу алгоритму AES де користувач обирає шифр, довжину ключа, формат та секретний ключ для шифрування даних після чого алгоритм повертає результати роботи

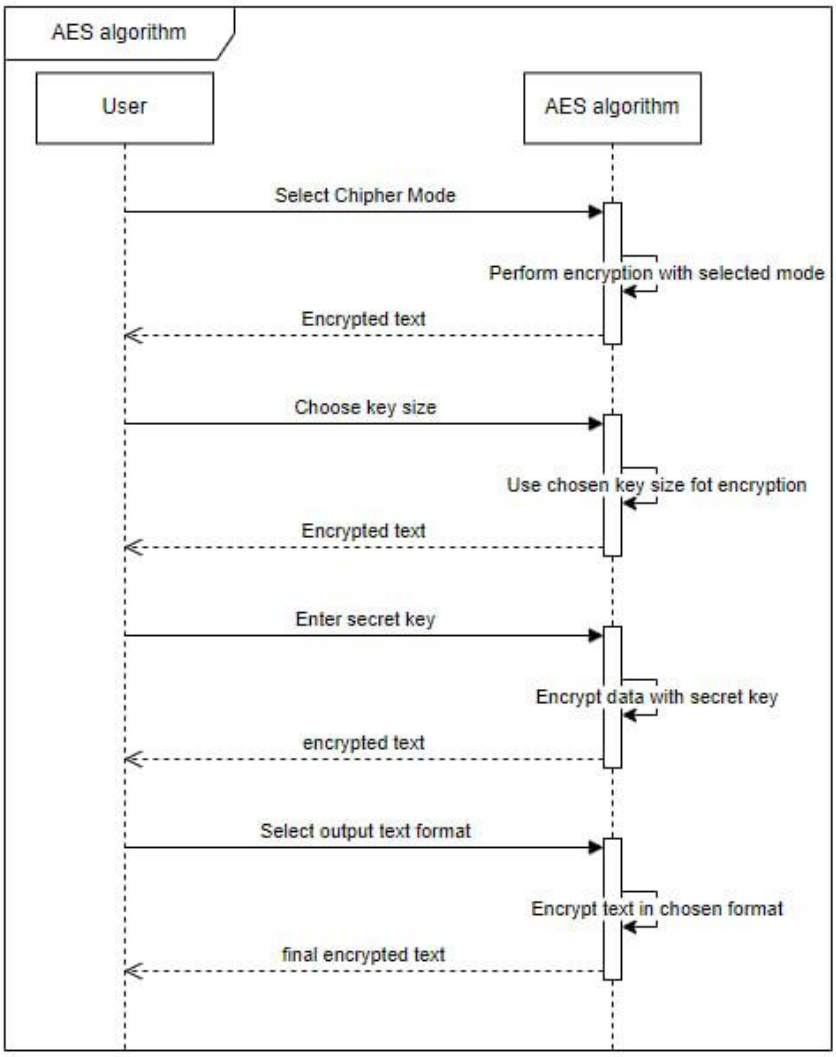


Рисунок 2.3 – Діаграма взаємодії AES алгоритму

В діаграмі №3 показано як буде працювати алгоритм RSA а саме користувач відправляє запит на отримання публічного ключа, після чого сервер відправляє запит на генерацію і після створення ключ відсилається назад користувачу. Для генерації приватного ключа відбувається та ж сама процедура де користувач надсилає запит на отримання приватного ключа і після його генерації програма надає його користувачу. Після створення ключа користувач може відправити запит на верифікацію публічного ключа після чого отримати результат операції. Останнім кроком який можна зробити

це дешифрувати чи зашифрувати інформацію використовуючи приватний ключ з подальшим отриманням інформації.

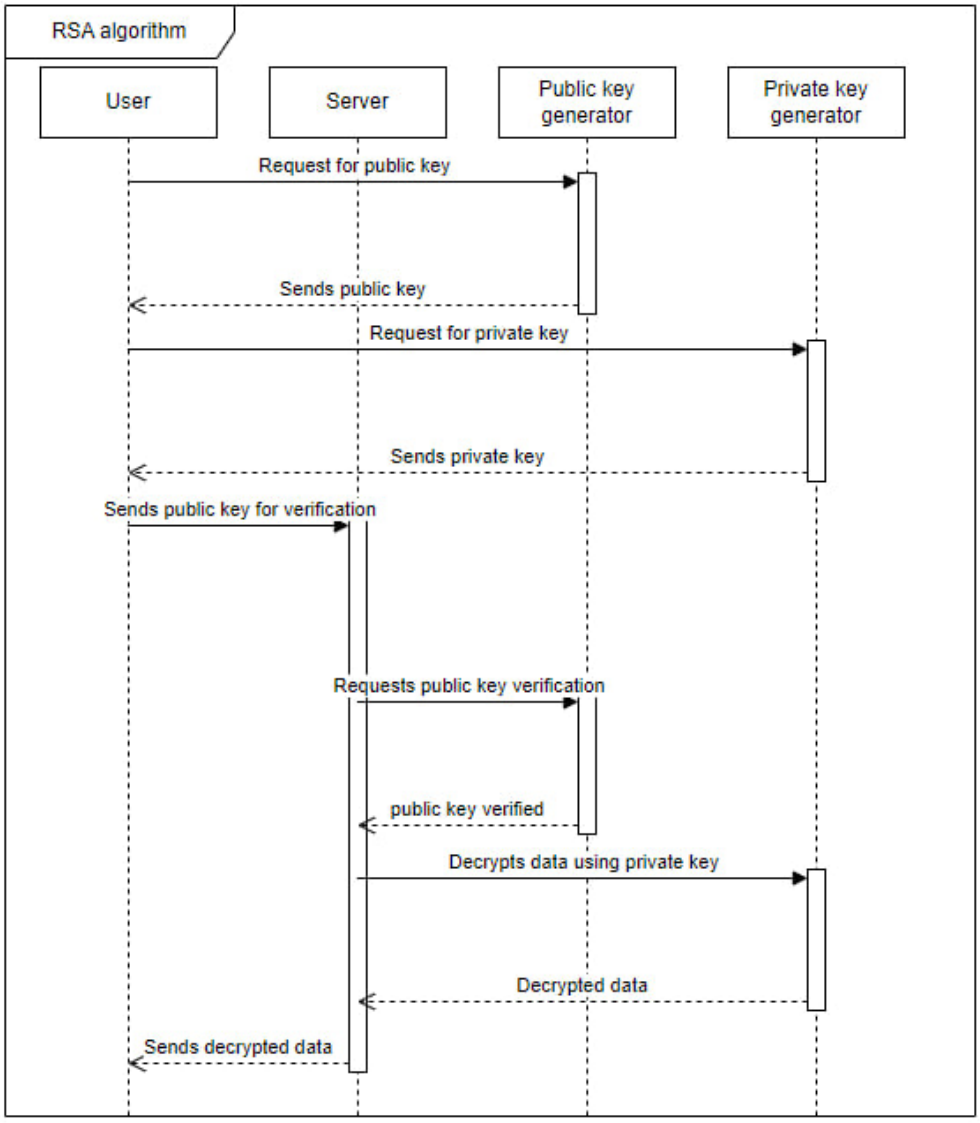


Рисунок 2.4 – Діаграма взаємодії RSA алгоритму

Зазначені діаграми зображують варіанти взаємодії користувача з системою та їх порядок взаємодії.

2.3 Діаграма діяльності

Діаграми діяльності в мові моделювання UML призначені для відображення процесів та потоків роботи системи. Вони демонструють різні шляхи, які можуть бути обраними залежно від умовних виразів.

Для вебзастосунку шифрування було розроблено 2 діаграми діяльності:

1) для процесу обробки шифрування даних AES(рис. 2.3);

2) для процесу обробки шифрування даних RSA (рис. 2.4);

На рис. 2.5 наведено процес шифрування даних алгоритмом AES. Діаграма містить такі кроки як: Згенерувати ключ, прийняти вхідний текст, обрати режим шифру, вибір довжини біта, введення таємного ключа, перевірка на помилки, фінальний результат

Рисунок 2.5 – Діаграма діяльності AES алгоритму

На рис. 2.5 наведено процес шифрування даних алгоритмом RSA. Діаграма містить такі кроки як: Згенерувати ключ, верифікувати ключ, прийняти вхідний текст, обрати режим шифру, введення ключа, перевірка на помилки, фінальний результат

Рисунок 2.6 – Діаграма діяльності RSA алгоритму

Таким чином можна побачити те як уде вигляди послідовність роботи з двома алгоритмами у виді діаграм.

2.4 Діаграма розгортання

Діаграми розгортання в мові моделювання UML дозволяють візуалізувати фізичну архітектуру системи, включаючи обладнання, на якому працюють програмні компоненти. Ці діаграми не лише показують, як компоненти системи взаємодіють між собою, а й допомагають зрозуміти їх розміщення та конфігурацію в операційному середовищі. Вони важливі для аналізу та проєктування системи, а також для оцінки її ефективності та масштабованості.

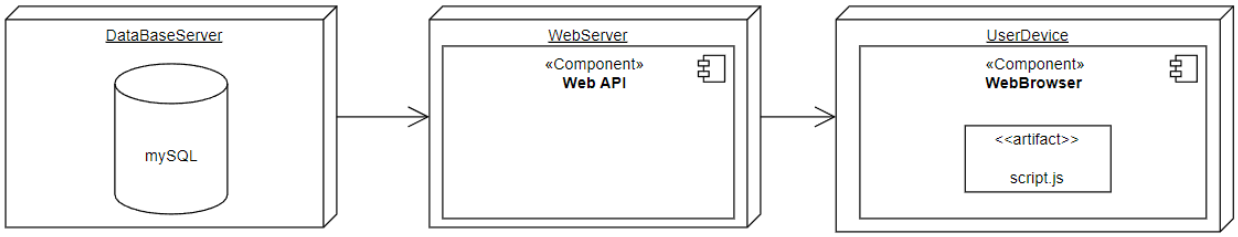


Рисунок 2.7 – Діаграма розгортання системи проєкту

На діаграмі розгортання зображені вузли які зображені прямокутними паралелепіпедами в яких представлено використання бази даних вебсервера та техніки користувача. Саме ці вузли являють собою діаграму розгортання цього проєкту.

Висновки до розділу 2

Другий розділ кваліфікаційної роботи направлений на побудову діаграм проєкту для того щоб краще розуміти те як саме будуть відбуватись процеси у системі. Створено сценарії використання які направлені на детальний огляд того які є основні сценарії роботи системи та альтернативні. Створено діаграми діяльності та взаємодії які спрямовані на те щоб продемонструвати як користувач має взаємодіяти з системою. Розроблено діаграму розгортання яка демонструє архітектуру системи.

3 ПРОЄКТУВАННЯ ЗАСТОСУНКУ ШИФРУВАННЯ

3.1 Діаграма класів

У цьому розділі розглянемо діаграму класів, яка є важливим елементом розробки програмного забезпечення. Діаграма класів відображає структуру системи, показуючи класи, що складають проєкт, а також їх взаємодію між собою.

Класи у діаграмі представлені у вигляді прямокутників, які містять три секції. Перша секція вказує ім'я класу, друга секція містить атрибути класу, а третя – методи класу. Такий підхід дозволяє наочно побачити, які елементи включає кожен клас.

Діаграма класів є корисною для розробників, оскільки дозволяє краще зрозуміти архітектуру системи і взаємозв'язки між її компонентами. Завдяки цій діаграмі можна визначити, які атрибути і методи містяться в кожному класі, і як ці класи взаємодіють один з одним. Це сприяє ефективному плануванню в розробці та підтримці програмного забезпечення.

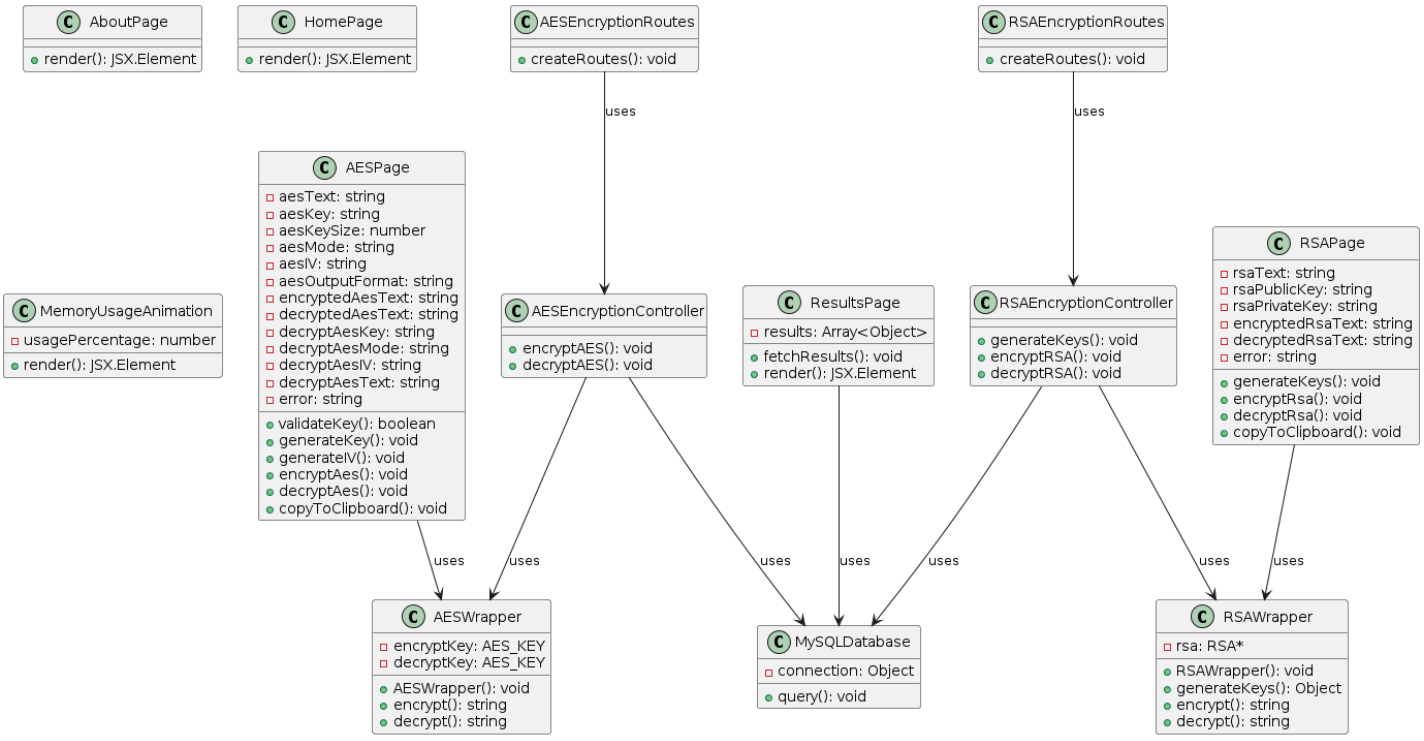


Рисунок 3.1 – Діаграма класів

Крім того, діаграма класів допомагає виявити потенційні проблеми на ранніх

етапах розробки, знижуючи ризик помилок і спрощуючи процес рефакторингу. Вона також є важливим інструментом для комунікації між членами команди, оскільки дозволяє їм бачити загальну картину проєкту і розуміти роль кожного класу в системі.

3.2 Діаграма об’єктів

Діаграма об’єктів використовується для моделювання реальних екземплярів класів у певний момент часу в межах програмної системи. Вона надає можливість відобразити не тільки структуру класів, але й конкретні об'єкти та їх взаємозв'язки, що існують на даний момент. На відміну від діаграми класів, яка показує статичну структуру системи і відносини між класами, діаграма об’єктів зосереджується на динамічному аспекті системи.

Цей тип діаграми ілюструє, як об'єкти взаємодіють один з одним під час виконання програми, надаючи уявлення про стан системи в конкретний момент часу. Це дозволяє розробникам і аналітикам краще зрозуміти, як працює система, і виявити можливі проблеми або місця для оптимізації.

Використання діаграми об’єктів також є корисним для тестування і налагодження програмного забезпечення. Вона допомагає виявити несподівані залежності між об'єктами або неправильне управління станом об'єктів. Наприклад, якщо два об'єкти мають бути незалежними, але виявляється, що між ними існує зв'язок, це може вказувати на проблему в коді або на потребу у рефакторингу.

Крім того, діаграма об’єктів може бути використана для документування поточного стану системи на різних етапах її життєвого циклу. Це корисно для підтримки і оновлення програмного забезпечення, оскільки надає детальний знімок того, як система функціонує в конкретний момент часу.

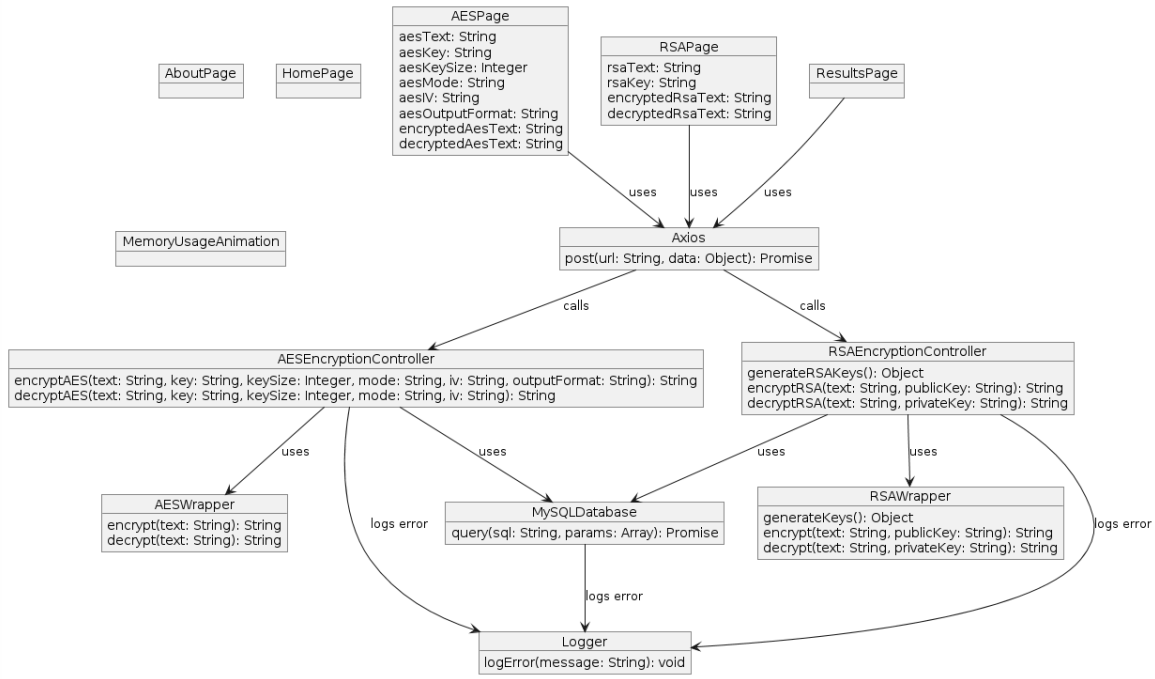


Рисунок 3.2 – Діаграма об’єктів

Таким чином, діаграма об’єктів є важливим інструментом для моделювання, аналізу і документування динамічного стану програмної системи, що доповнює діаграму класів і допомагає глибше зрозуміти внутрішню роботу системи.

3.3Діаграма пакетів

Наступною на розгляд стане діаграма пакетів. Діаграма пакетів використовується для організації та візуалізації структури проєкту, особливо коли проєкт є достатньо великим. Вона дозволяє побачити загальні дії та взаємозв'язки між різними частинами системи.

Frontend пакет відповідає за користувацький інтерфейс і взаємодію з користувачем. Тут ми використовуємо React для створення компонентів, що забезпечують динамічний і інтерактивний інтерфейс. Компоненти React дозволяють розробникам створювати складні користувацькі інтерфейси з багатою функціональністю, забезпечуючи при цьому зручність і простоту в користуванні. Вони можуть включати різні елементи управління, форми, таблиці, графіки та інші візуальні компоненти, які покращують користувацький досвід.

Backend пакет відповідає за обробку даних, бізнес-логіку і взаємодію з базами

даних та іншими зовнішніми системами. Він забезпечує роботу сервера, який приймає запити від клієнта, обробляє їх, виконує необхідні операції з даними і повертає результати назад на клієнт. Для реалізації бекенд-логіки ми можемо використовувати різні серверні технології та фреймворки, такі як Node.js, Express.

Frontend і backend тісно взаємодіють один з одним, обмінюючись даними через API. Це дозволяє розділити відповідальність між клієнтською та серверною частинами, забезпечуючи модульність і гнучкість системи. Клієнтські запити до API можуть включати отримання даних для відображення, відправлення даних для обробки, а також виконання різних операцій, таких як автентифікація користувачів, збереження змін і так далі. Backend, у свою чергу, обробляє ці запити, виконує бізнес-логіку і взаємодіє з базами даних для збереження або отримання необхідної інформації.

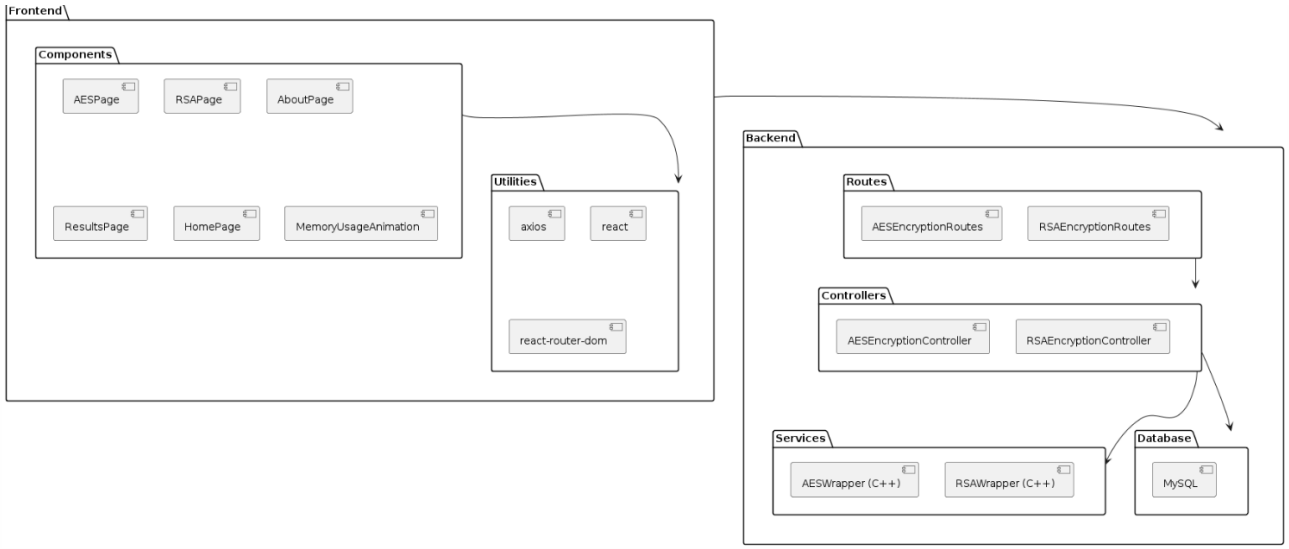


Рисунок 3.3 – Діаграма пакетів

Цей пакет відповідає за обробку запитів від користувачів і взаємодію з базою даних. Діаграма пакетів допомагає візуалізувати структуру проєкту, показуючи, які частини системи взаємодіють між собою і як вони організовані.

3.4Діаграма компонентів

Діаграма компонентів використовується для моделювання фізичної архітектури програмної системи. Вона допомагає зрозуміти, як різні програмні компоненти

взаємодіють один з одним і як вони розміщені в системі. Ця діаграма охоплює всі компоненти, модулі, файли, бібліотеки, бази даних та інші фізичні артефакти, що складають програмне забезпечення. Вона є важливим інструментом для планування розгортання системи, оскільки показує, як і де будуть розміщені компоненти, що допомагає визначити оптимальні конфігурації для досягнення високої продуктивності та надійності системи.

Однією з основних функцій діаграм компонентів є надання документації для розробників, архітекторів та інших учасників проєкту. Вони надають вичерпну інформацію про структуру системи, що значно полегшує її розуміння. Це особливо корисно для нових членів команди, які можуть швидше ознайомитися з архітектурою проєкту, або для подальшої підтримки та розвитку системи, коли детальне уявлення про компоненти і їх взаємозв'язки є критично важливим.

Діаграми компонентів також служать основою для обговорення та прийняття рішень щодо розгортання системи. Вони допомагають визначити найкращі способи розміщення компонентів, враховуючи різні фактори, такі як продуктивність, масштабованість, безпека та надійність. Завдяки цим діаграмам можна більш ефективно спланувати використання ресурсів та уникнути можливих проблем на етапі впровадження.

Крім того, діаграми компонентів є важливим інструментом для комунікації між різними учасниками проєкту. Вони забезпечують візуальне представлення структури системи, що полегшує обговорення архітектурних рішень та виявлення можливих поліпшень. Завдяки чіткому та зрозумілому відображенню всіх компонентів і їх взаємозв'язків, ці діаграми сприяють ефективній співпраці в команді та поліпшенню якості кінцевого продукту.

Завдяки детальному відображенню структури системи, діаграма компонентів є важливим інструментом для успішного впровадження та подальшої підтримки програмного забезпечення на всіх етапах його життєвого циклу.

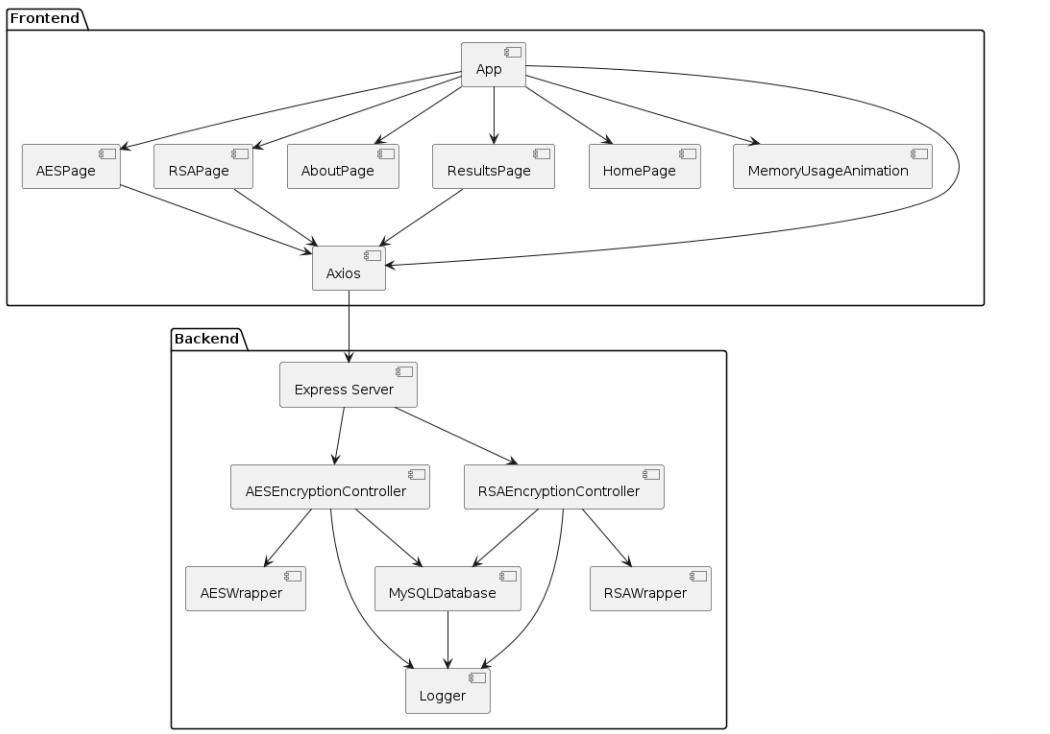


Рисунок 3.4 – Діаграма компонентів

Таким чином, діаграма компонентів є невід'ємною частиною процесу розробки програмного забезпечення. Вона допомагає не тільки у плануванні та розгортанні системи, але й забезпечує краще розуміння її архітектури

3.5 Діаграма станів та переходів

Діаграма станів та переходів використовується для моделювання поведінки системи через різні стани, в яких вона може перебувати, та умови, за яких вона переходить з одного стану в інший. Ці діаграми є важливим інструментом для розробників, оскільки вони дозволяють детально описати всі можливі стани системи і умови їх переходу, що допомагає уникнути помилок і непорозумінь під час розробки та тестування програмного забезпечення.

Діаграми станів використовуються для розробки і аналізу алгоритмів, особливо в системах з керуванням подіями. Вони допомагають розробникам зрозуміти, як система реагує на різні події і які дії повинні бути виконані у відповідь на ці події. Це особливо корисно для складних систем, де поведінка може змінюватися залежно від

різних умов та контексту.

Діаграми станів дозволяють візуалізувати, як система або об'єкт поводиться протягом часу. Вони надають наочне уявлення про те, як система змінює свої стани і які події викликають ці зміни. Це сприяє кращому розумінню роботи системи і допомагає виявити можливі проблеми на ранніх етапах розробки.

Діаграми станів слугують документацією для програмних компонентів, надаючи чітке уявлення про те, як об'єкти або модулі повинні поводитися у відповідь на різні події. Вони забезпечують структурований підхід до розробки програмного забезпечення, допомагаючи розробникам створювати більш надійні та ефективні системи.

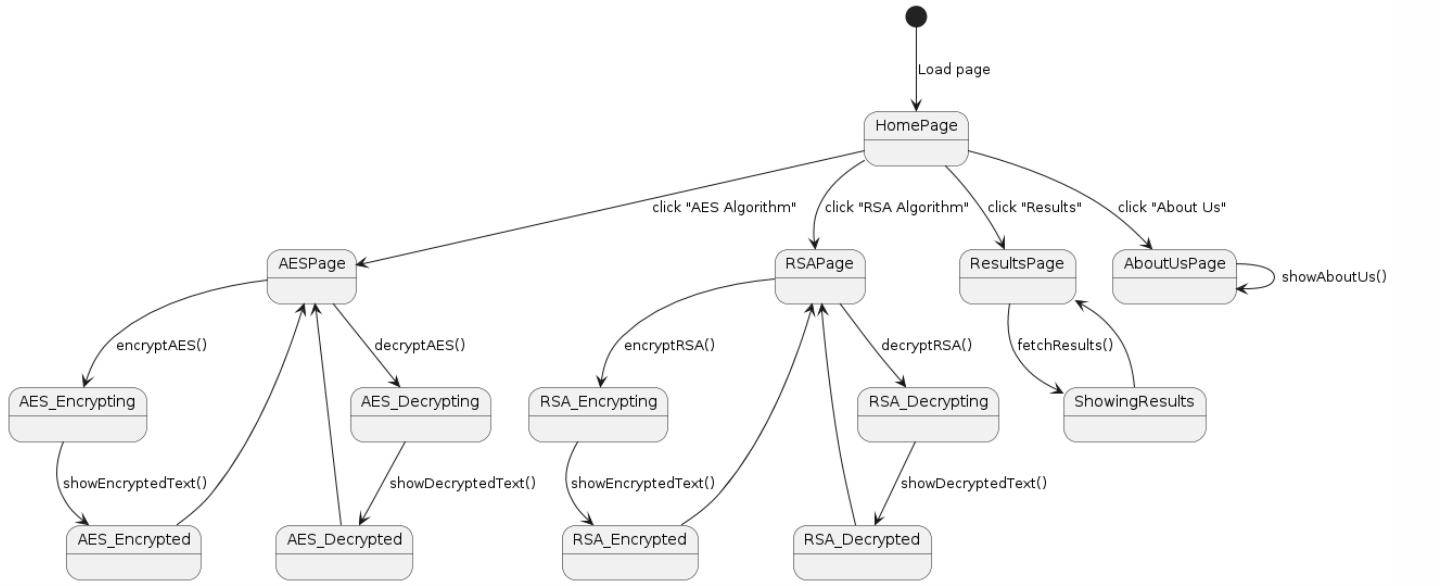


Рисунок 3.5 – Діаграма станів та переходів

Завдяки своїй здатності візуалізувати динамічну поведінку системи, діаграми станів є незамінним інструментом для розробників, аналітиків та інших учасників проєкту.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи бакалавра детально розглядається процес проєктування програмного застосунку за допомогою різноманітних UML-діаграм. Цей розділ охоплює широкий спектр аспектів проєктування, включаючи

діаграми пакетів, компонентів, станів і переходів, об'єктів і класів. Кожна з цих діаграм виконує важливу функцію у візуалізації структури та поведінки застосунку, що допомагає у чіткому розумінні його архітектури та взаємодії між різними частинами системи.

Також у цьому розділі представлено обґрунтування вибору мов програмування, фреймворків та бібліотек, які використовуються для розробки застосунку. Кожен з цих інструментів був обраний на основі їхніх специфічних переваг і відповідності вимогам проєкту.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

4.1 Вибір стеку технологій

Технологічний стек є дуже важливим компонентом будь-якого програмного застосунку. Вибір конкретних технологій визначає безліч аспектів функціонування застосунку, включаючи його продуктивність, рівень безпеки та апаратні вимоги, які будуть висунуті до кінцевих користувачів. Кожна технологія в відіграє свою роль, забезпечуючи необхідні функції та характеристики системи.

У випадку розробки застосунку для шифрування, було обрано певний набір технологій, які відповідають поставленим завданням і вимогам проєкту. Цей вибір детально представлений на діаграмі (рис. 4.1). Кожна з обраних технологій має свої унікальні властивості та переваги, що дозволяють забезпечити ефективну і безпечну роботу системи, а також оптимальну взаємодію з апаратним забезпеченням користувачів.

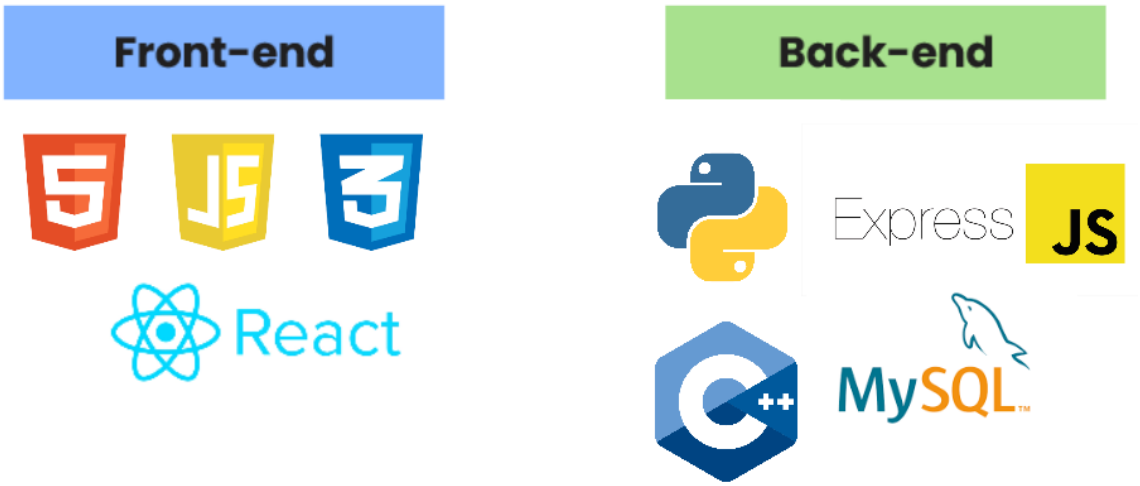


Рисунок 4.1 – Стек технологій

React.js – це JavaScript-бібліотека для створення користувацьких інтерфейсів, особливо односторінкових додатків. Вона була створена компанією Facebook і стала однією з найпопулярніших інструментів для фронтенд-розробки завдяки своїй гнучкості та ефективності. Основною концепцією React є використання компонентів, які можуть бути легко повторно використані і об'єднані для створення складних

інтерфейсів. React також використовує віртуальний DOM, що дозволяє ефективно оновлювати і відображати зміни в користувацькому інтерфейсі без перезавантаження сторінки. Завдяки своїм можливостям і підтримці великою спільнотою розробників, React.js залишається однією з найбільш затребуваних бібліотек у сучасній веб-розробці.

C++ – це мова програмування, яка є розширенням мови C і включає об'єктно-орієнтовані можливості. Вона була розроблена Б'ярном Страуструпом у 1980-х роках і швидко стала однією з основних мов програмування для системного і прикладного програмного забезпечення. C++ використовується в розробці операційних систем, ігрових двигунів, високопродуктивних серверних додатків і вбудованих систем. Мова відома своєю продуктивністю і можливостями низькорівневого управління пам'яттю, що дозволяє створювати ефективний і високопродуктивний код. Незважаючи на складність синтаксису і круту криву навчання, C++ залишається популярною завдяки своїй потужності і широкому спектру застосувань.

Python – це високорівнева, інтерпретована мова програмування, відома своєю простотою і читабельністю. Створена Гвідо ван Россумом і вперше випущена у 1991 році, Python швидко завоювала популярність завдяки своїй універсальності і широкому спектру бібліотек. Вона широко використовується в різних галузях, включаючи веб-розробку, науку про дані, машинне навчання, автоматизацію і розробку скриптів. Python також відомий своєю великою і активною спільнотою, яка сприяє розвитку численних бібліотек і фреймворків, таких як Django, Flask, NumPy і TensorFlow. Завдяки своїй простоті та універсальності, Python часто рекомендується як перша мова для вивчення програмування.

Express – це мінімалістичний веб-фреймворк для Node.js, який спрощує розробку серверних веб-додатків і API. Він забезпечує набір функцій для створення веб-додатків і мобільних додатків, включаючи обробку HTTP-запитів, маршрутизацію, посилання на середовища виконання і багато іншого. Express є одним з найпопулярніших фреймворків для Node.js завдяки своїй простоті і гнучкості, що

дозволяє розробникам швидко створювати високопродуктивні серверні додатки. Важливою перевагою Express є велика кількість середовищ виконання і модулів, які можуть бути легко інтегровані для розширення функціональності додатка. Використовуючи Express, розробники можуть швидко налаштувати і запустити серверні частини веб-додатків з мінімальними витратами часу і зусиль.

JavaScript – це високорівнева мова програмування, яка є основною технологією для розробки веб-сторінок разом з HTML і CSS. Спочатку створена для додавання інтерактивних елементів до веб-сторінок, JavaScript зараз використовується для розробки як фронтенд, так і бекенд додатків завдяки появі таких платформ, як Node.js. JavaScript дозволяє розробникам створювати динамічні і інтерактивні веб-сторінки, маніпулювати елементами DOM, обробляти події і виконувати асинхронні запити. Мова має величезну екосистему бібліотек і фреймворків, таких як React, Angular, Vue.js, які значно спрощують розробку складних веб-додатків. Завдяки своїй універсальності і широкому використанню, JavaScript залишається однією з найпопулярніших мов програмування у світі.

HTML (HyperText Markup Language) – це стандартна мова розмітки для створення веб-сторінок і веб-додатків. Вона використовується для структурування контенту на веб-сторінці, включаючи текст, зображення, відео та інші мультимедійні елементи. HTML є основою всіх веб-документів і забезпечує базовий каркас, який потім можна стилізувати за допомогою CSS і оживляти за допомогою JavaScript. HTML5, остання версія HTML, додала нові можливості, такі як вбудоване відео і аудіо, нові семантичні елементи і покращену підтримку мобільних пристроїв. Завдяки своїй простоті і потужності, HTML залишається фундаментальною технологією для веб-розробки, яку використовують всі веб-розробники.

MySQL – це система управління реляційними базами даних (RDBMS), яка використовується для зберігання і керування даними у структурованому форматі. Вона була створена у 1995 році компанією MySQL AB і зараз є частиною Oracle Corporation. MySQL є однією з найпопулярніших баз даних у світі завдяки своїй

простоті, надійності і відкритому вихідному коду. Вона широко використовується у веб-додатках, таких як WordPress, Joomla, Drupal, а також у багатьох інших програмних продуктах. MySQL підтримує стандартний SQL (Structured Query Language), що дозволяє розробникам легко створювати, читати, оновлювати і видаляти дані у базі даних. Завдяки своїм можливостям і активній спільноті, MySQL залишається важливим інструментом для розробників у всьому світі.

CSS – це мова стилів, яка використовується для опису зовнішнього вигляду та форматування документів, написаних на HTML або XML. CSS дозволяє розробникам відокремити структуру контенту веб-сторінки від її презентаційного аспекту, що робить можливим створення візуально привабливих і узгоджених інтерфейсів. Завдяки CSS можна легко змінювати кольори, шрифти, розміри, відступи, вирівнювання та інші стилі елементів на веб-сторінці. Крім того, CSS підтримує адаптивний дизайн (responsive design), що дозволяє веб-сторінкам коректно відображатися на різних пристроях з різними розмірами екрану. Сучасні можливості CSS, такі як Flexbox і Grid Layout, роблять верстку веб-сторінок більш гнучкою та потужною, дозволяючи створювати складні макети без використання додаткових бібліотек.

Node.js – це середовище виконання JavaScript на сервері, яке дозволяє запускати JavaScript-код поза браузером. Воно побудоване на движку V8 від Google і забезпечує високу продуктивність завдяки використанню неблокуючої, подієво-орієнтованої моделі введення-виведення. Node.js широко використовується для створення серверних додатків, зокрема веб-серверів, API та мікросервісів, завдяки своїй здатності обробляти велику кількість одночасних з'єднань з мінімальними витратами на ресурси.

Нижче (рис. 3.7) можна побачити те наскільки популярні ці технологій у користувачів сайту stackoverflow за рахунок кількості публікацій щодо цих технологій.

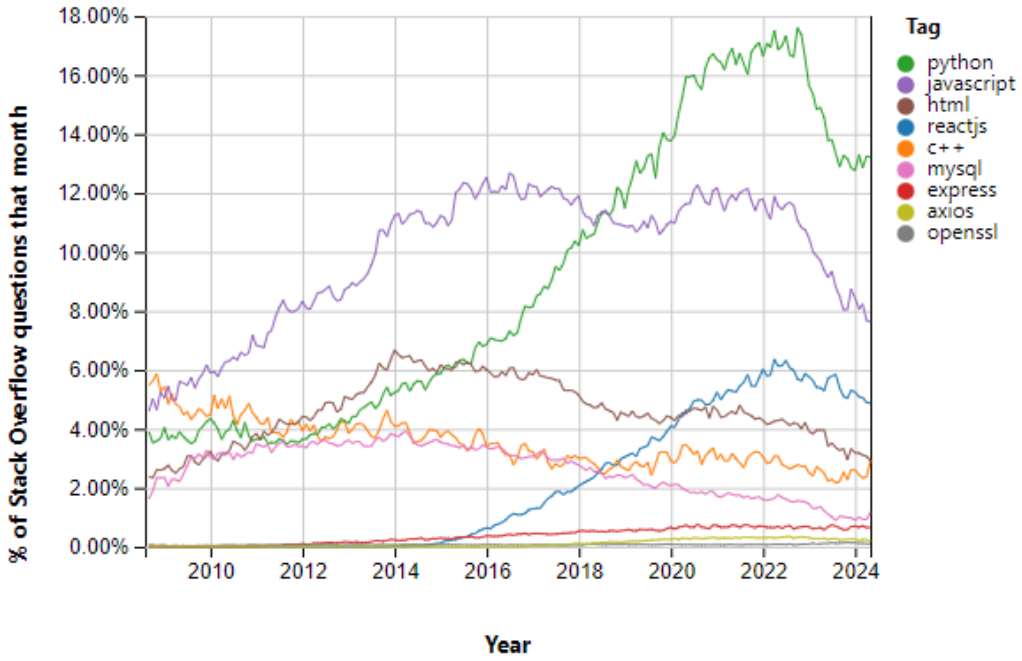


Рисунок 4.2 – Графік популярності пошуку на платформі stackoverflow

У цьому випадку можна побачити те яким мовам віддають перевагу програмісти при створені опенсорс проєктів що надає більш чітке розуміння того що саме варто роздивлятись при виборі технологій для майбутнього проєкту щоб мати з ним можливість працювати як можна ефективніше (рис. 3.8).

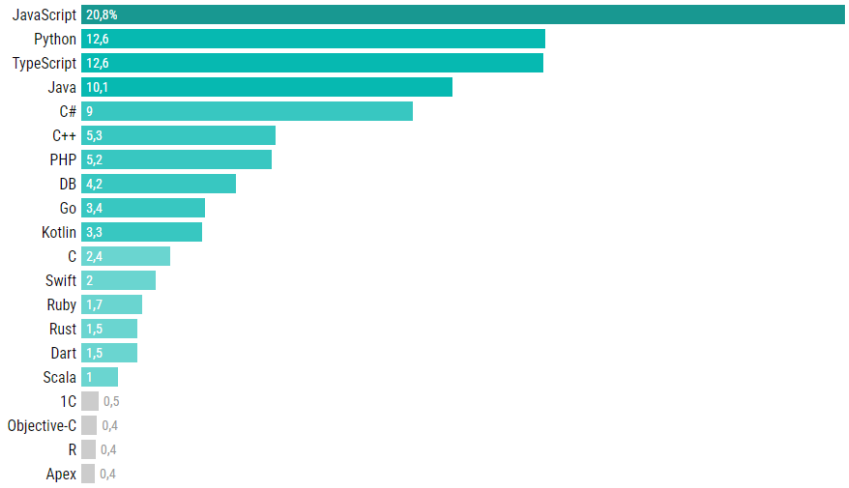


Рисунок 4.3 – Графік мов які використовуються в опенсорс-проєктах

Серед представлених можна побачити вже вище обрані мови такі як JavaScript, Python та C++.

4.2 Налаштування проєкту та створення БД

Перед початком реалізації основного проєкту треба встановити відповідні бібліотеки та розширення для того щоб можна було використовувати весь перелік технологій які були обрані для написання коду. Такими є платформа node.js та бібліотека react. Для встановлення node достатньо перейти на сайт розробника і встановити його (рис 4.4)

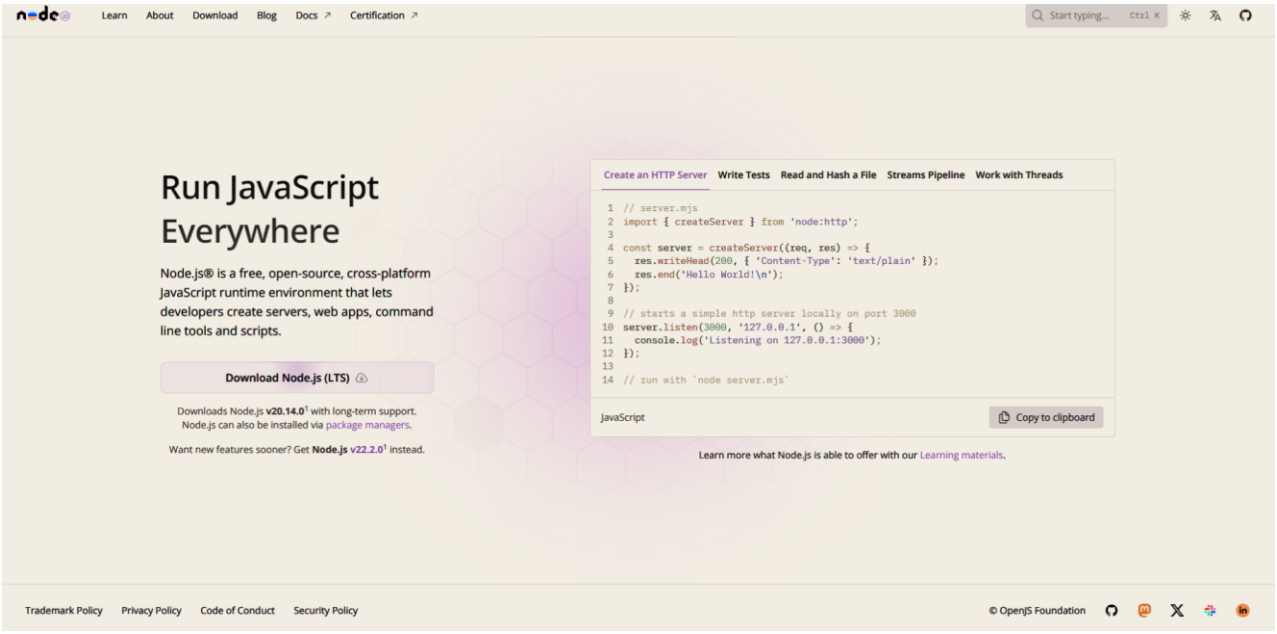


Рисунок 4.4 – Офіційний сайт node.js

У випадку react для можливість провести встановлення надається вже безпосередньо у середовищі розробки Visual Studio Code де це можливо зробити командою npm install. Додатково зі встановленням react можна встановити і інші залежності такі як express та axios які достатньо вказати у файлі package.json (рис. 4.5)

```
PS D:\University\react_kr-main> npm install

up to date, audited 1674 packages in 36s

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

Рисунок 4.5 – Встановлення додаткових залежностей

На цьому моменті усі необхідні залежності для проєкту були успішно встановлені, і тепер можна зосередитися на основних етапах розробки і приступити

до безпосередньої роботи над основними частинами застосунку.

Першим завданням на цьому етапі буде створення бази даних. Розпочнемо з проєктування структури бази даних, яка буде використовуватися для зберігання, управління та доступу до даних. Це включатиме визначення таблиць, полів, типів даних, а також налаштування зв'язків між таблицями.

База даних буде складатись з 4 таблиць по 2 на алгоритм де перша буде відповідати за шифрування а друга за дешифрування відповідно. Першими розглянемо таблиці AES. Таблиця aes_encryption використовується для зберігання інформації про процеси шифрування даних в той час як aes_decryption для дешифрування проте структурно їх різниця не значна тому розглянемо тільки на одному прикладі структуру цих двох таблиць.

1) id: Унікальний ідентифікатор запису, який автоматично збільшується з кожним новим записом.
2) text: Текст, який потрібно зашифрувати;
3) enc_key: Ключ, який використовується для шифрування;
4) key_size: Розмір ключа шифрування;
5) mode: Режим шифрування;
6) iv: Ініціалізаційний вектор (IV), який використовується в деяких режимах шифрування;
7) output_format: Формат вихідних даних;
8) result: Результат шифрування;
9) encryption_time: Час, витрачений на шифрування;
10) memory_used: Використана оперативна пам'ять під час шифрування;
11) created_at: Дата і час створення запису.

```
CREATE TABLE aes_encryption (
    id INT AUTO_INCREMENT PRIMARY KEY,
    text TEXT NOT NULL,
    enc_key TEXT NOT NULL,
    key_size INT NOT NULL,
    mode VARCHAR(10) NOT NULL,
    iv TEXT,
    output_format VARCHAR(10) NOT NULL,
    result TEXT NOT NULL,
    encryption_time FLOAT NOT NULL,
    memory_used FLOAT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE aes_decryption (
    id INT AUTO_INCREMENT PRIMARY KEY,
    encrypted_text TEXT NOT NULL,
    enc_key TEXT NOT NULL,
    key_size INT NOT NULL,
    mode VARCHAR(10) NOT NULL,
    iv TEXT,
    output_format VARCHAR(10) NOT NULL,
    result TEXT NOT NULL,
    decryption_time FLOAT NOT NULL,
    memory_used FLOAT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Рисунок 4.6 – Таблиці aes_encryption та aes_decryption

Таблиця rsa_encryption використовується для зберігання даних про процеси шифрування з використанням алгоритму RSA.

id: Унікальний ідентифікатор запису, який автоматично збільшується з кожним новим записом.

text: Текст, який потрібно зашифрувати.

encrypted_text: Результат шифрування.

public_key: Публічний ключ, який використовується для шифрування.

key_size: Розмір ключа шифрування.

encryption_time: Час, витрачений на шифрування, в секундах.

memory_used: Використана оперативна пам'ять під час шифрування, в мегабайтах.

created_at: Дата і час створення запису, автоматично заповнюється поточним часом.

Таблиця rsa_decryption

Таблиця rsa_decryption зберігає інформацію про процеси дешифрування даних з

використанням алгоритму RSA.

id: Унікальний ідентифікатор запису, який автоматично збільшується з кожним новим записом.

encrypted_text: Зашифрований текст, який потрібно дешифрувати.

private_key: Приватний ключ, який використовується для дешифрування.

result: Результат дешифрування.

key_size: Розмір ключа дешифрування.

decryption_time: Час, витрачений на дешифрування, в секундах.

memory_used: Використана оперативна пам'ять під час дешифрування, в мегабайтах.

created_at: Дата і час створення запису, автоматично заповнюється поточним часом.

```
CREATE TABLE rsa_encryption (
    id INT AUTO_INCREMENT PRIMARY KEY,
    text TEXT NOT NULL,
    encrypted_text TEXT NOT NULL,
    public_key TEXT NOT NULL,
    key_size INT NOT NULL,
    encryption_time FLOAT NOT NULL,
    memory_used FLOAT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE rsa_decryption (
    id INT AUTO_INCREMENT PRIMARY KEY,
    encrypted_text TEXT NOT NULL,
    key_size INT NOT NULL,
    private_key TEXT NOT NULL,
    result TEXT NOT NULL,
    decryption_time FLOAT NOT NULL,
    memory_used FLOAT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Рисунок 4.7 – Таблиці rsa_encryption та rsa_decryption

Створену базу даних доєднаємо до проєкту вказавши відповідні дані у файлі server.js.

```
const pool = mysql.createPool({
    host: 'localhost',
    user: 'root',
    password: '1111',
    database: 'encryption_db'
});
```

Рисунок 4.8 – Підключення бази даних

Після створення БД додамо до проєкту інші файли. Перш за все створимо папку backend в яку піде код криптографічних алгоритмів у виді файлів aes.cpp rsa.cpp та прив’язка цих файлів з бібліотеками та іншими залежностями.

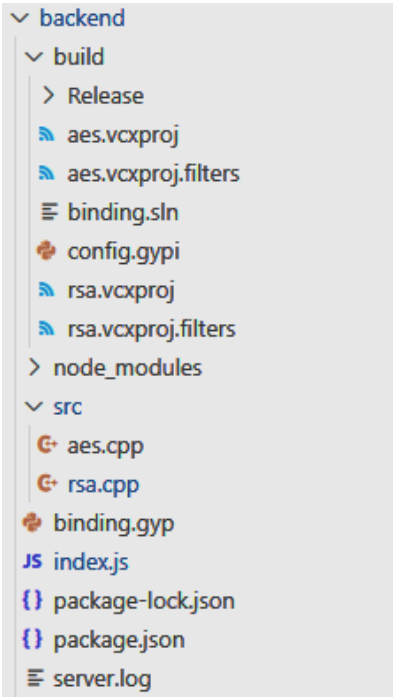


Рисунок 4.9 – Структура папки backend

Окрім backend папки створимо папку для frontend. До неї будуть входити файли сторінок про нас, алгоритмів AES та RSA а також головна та результати. Також до папки буде входити згенеровані файли App.js та App.css що будуть відповідати за зовнішній вигляд сайту як проєкт перейде до етапу верстки.

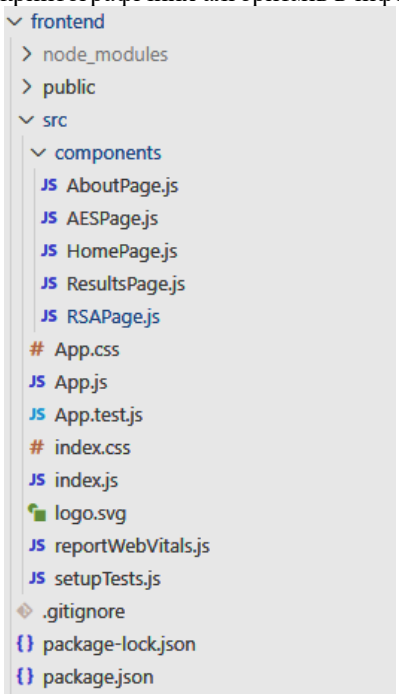


Рисунок 4.10 – Структура папки frontend

Таким чином маємо сформовану та підключену базу даних яку будемо використовувати для подальшого розвитку проєкту а також заготовку папок та файлів.

4.3 Розробка backend

Наступним етапом після створення та приєднання БД є розробка backend програми. На цьому етапі почнемо з цієї частини перед тим як робити дизайн сайту. Так як вже було попередньо представлено макет то дизайн стає більш другорядною частиною в порівнянні з функціоналом.

Перше що зробимо це додамо головну сторінку де будуть відображатись посилання на інші сторінки.

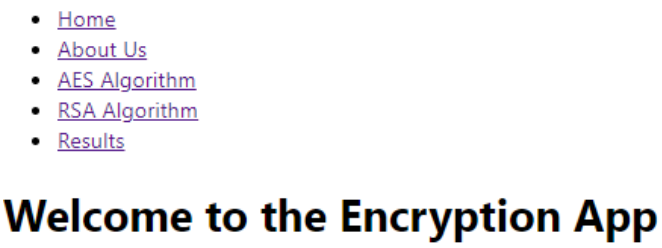


Рисунок 4.11 – Первинний вигляд головної сторінки

Маючи можливість переходити між сторінками можна почати робити інші частини наприклад алгоритми. Першим яким буде створено це алгоритм AES. Він надає інтерфейс для роботи з AES через Node.js, використовуючи N-API для створення обгорток навколо C++ функцій шифрування (рис 4.12) та дешифрування (рис 4.13) з кодуванням та декодуванням у форматі base64 (рис 4.14).

```
Napi::Value AESWrapper::Encrypt(const Napi::CallbackInfo& info) {
    Napi::Env env = info.Env();
    Napi::HandleScope scope(env);

    std::string plainText = info[0].As<Napi::String>().Utf8Value();
    std::vector<unsigned char> paddedText(plainText.begin(), plainText.end());
    handlePadding(paddedText, true);

    std::vector<unsigned char> encryptedText(paddedText.size());

    if (this->mode == "ECB") {
        AES_ECB_encrypt(paddedText, encryptedText);
    } else if (this->mode == "CBC") {
        AES_CBC_encrypt(paddedText, encryptedText);
    } else if (this->mode == "CFB") {
        AES_CFB_encrypt(paddedText, encryptedText);
    } else {
        Napi::Error::New(env, "Unsupported mode").ThrowAsJavaScriptException();
        return env.Null();
    }

    return Napi::String::New(env, base64Encode(encryptedText.data(), encryptedText.size()));
}
```

Рисунок 4.11 – Код AES шифрування

```
Napi::Value AESWrapper::Decrypt(const Napi::CallbackInfo& info) {
    Napi::Env env = info.Env();
    Napi::HandleScope scope(env);

    std::vector<unsigned char> decodedText = base64Decode(info[0].As<Napi::String>().Utf8Value());
    std::vector<unsigned char> decryptedText(decodedText.size());

    if (this->mode == "ECB") {
        AES_ECB_decrypt(decodedText, decryptedText);
    } else if (this->mode == "CBC") {
        AES_CBC_decrypt(decodedText, decryptedText);
    } else if (this->mode == "CFB") {
        AES_CFB_decrypt(decodedText, decryptedText);
    } else {
        Napi::Error::New(env, "Unsupported mode").ThrowAsJavaScriptException();
        return env.Null();
    }

    try {
        handlePadding(decryptedText, false);
    } catch (const std::runtime_error& e) {
        Napi::Error::New(env, e.what()).ThrowAsJavaScriptException();
        return env.Null();
    }

    return Napi::String::New(env, std::string(decryptedText.begin(), decryptedText.end()));
}
```

Рисунок 4.12 – Код AES дешифрування

```cpp
std::string base64Encode(const unsigned char* buffer, size_t length) {
    BIO *bio, *b64;
    BUF_MEM *bufferPtr;

    b64 = BIO_new(BIO_f_base64());
    bio = BIO_new(BIO_s_mem());
    bio = BIO_push(b64, bio);

    BIO_set_flags(bio, BIO_FLAGS_BASE64_NO_NL);
    BIO_write(bio, buffer, length);
    BIO_flush(bio);
    BIO_get_mem_ptr(bio, &bufferPtr);

    std::string encodedData(bufferPtr->data, bufferPtr->length);
    BIO_free_all(bio);

    return encodedData;
}

std::vector<unsigned char> base64Decode(const std::string &base64String) {
    BIO *bio, *b64;

    int decodeLen = static_cast<int>(base64String.size());
    std::vector<unsigned char> buffer(decodeLen);

    bio = BIO_new_mem_buf(base64String.c_str(), -1);
    b64 = BIO_new(BIO_f_base64());
    bio = BIO_push(b64, bio);

    BIO_set_flags(bio, BIO_FLAGS_BASE64_NO_NL);
    decodeLen = BIO_read(bio, buffer.data(), static_cast<int>(base64String.size()));
    buffer.resize(decodeLen);
    BIO_free_all(bio);

    return buffer;
}
```

Рисунок 4.13 – Код формату виводу у base64

Зокрема AES має в собі такі додаткові частини як IV, режим шифрування та розмір ключа. IV або ж Initialization Vector використовується в алгоритмі AES для досягнення кількох важливих цілей. випадковість та унікальність, безпека та унікальність. IV забезпечує, що однакові повідомлення, зашифровані з одним і тим самим ключем, призведуть до різних зашифрованих текстів. Цим алгоритм уникає паттернів в шифруванні, які могли б допомогти зловмиснику розкрити інформацію.

```cpp
Napi::Value AESWrapper::GetIV(const Napi::CallbackInfo& info) {
    Napi::Env env = info.Env();
    return Napi::String::New(env, base64Encode(this->iv.data(), this->iv.size()));
}

Napi::Object InitAll(Napi::Env env, Napi::Object exports) {
    return AESWrapper::Init(env, exports);
}
```

Рисунок 4.14 – Код AES IV

AES підтримує кілька режимів шифрування, кожен з яких має свої особливості та застосування: Режими шифрування в AES визначають спосіб, яким алгоритм шифрування застосовується до даних. AES сам по собі є блочним шифром, тобто він обробляє дані в блоках фіксованого розміру 128 біт або 16 байт. Режими шифрування визначають, як ці блоки обробляються, щоб забезпечити безпеку, ефективність та гнучкість. Основні режими шифрування для AES включають ECB, CBC, CFB, OFB, CTR з яких буде взято тільки перші три.

ECB (Electronic Codebook): Найпростіший режим, який шифрує кожен блок даних окремо. Недоліком є те, що однакові блоки відкритого тексту призводять до однакових блоків шифрованого тексту, що робить цей режим менш безпечним.

CBC (Cipher Block Chaining): Кожен блок відкритого тексту XOR'иться з попереднім блоком шифрованого тексту перед шифруванням. Перший блок XOR'иться з IV. Це забезпечує більш високий рівень безпеки, ніж ECB.

CFB (Cipher Feedback): Працює з блоками менших розмірів (наприклад, 8 біт), що дозволяє шифрувати дані потоком. IV також використовується для першого блоку.

```
void AESWrapper::AES_ECB_encrypt(const std::vector<unsigned char>& plainText, std::vector<unsigned char>& cipherText) {
    for (size_t i = 0; i < plainText.size(); i += AES_BLOCK_SIZE) {
        AES_encrypt(plainText.data() + i, cipherText.data() + i, &encryptKey);
    }
}

void AESWrapper::AES_ECB_decrypt(const std::vector<unsigned char>& cipherText, std::vector<unsigned char>& plainText) {
    for (size_t i = 0; i < cipherText.size(); i += AES_BLOCK_SIZE) {
        AES_decrypt(cipherText.data() + i, plainText.data() + i, &decryptKey);
    }
}

void AESWrapper::AES_CBC_encrypt(const std::vector<unsigned char>& plainText, std::vector<unsigned char>& cipherText) {
    std::vector<unsigned char> iv_copy = this->iv;
    AES_cbc_encrypt(plainText.data(), cipherText.data(), plainText.size(), &encryptKey, iv_copy.data(), AES_ENCRYPT);
}

void AESWrapper::AES_CBC_decrypt(const std::vector<unsigned char>& cipherText, std::vector<unsigned char>& plainText) {
    std::vector<unsigned char> iv_copy = this->iv;
    AES_cbc_encrypt(cipherText.data(), plainText.data(), cipherText.size(), &decryptKey, iv_copy.data(), AES_DECRYPT);
}

void AESWrapper::AES_CFB_encrypt(const std::vector<unsigned char>& plainText, std::vector<unsigned char>& cipherText) {
    std::vector<unsigned char> iv_copy = this->iv;
    int num = 0;
    AES_cfb128_encrypt(plainText.data(), cipherText.data(), plainText.size(), &encryptKey, iv_copy.data(), &num, AES_ENCRYPT);
}

void AESWrapper::AES_CFB_decrypt(const std::vector<unsigned char>& cipherText, std::vector<unsigned char>& plainText) {
    std::vector<unsigned char> iv_copy = this->iv;
    int num = 0;
    AES_cfb128_encrypt(cipherText.data(), plainText.data(), cipherText.size(), &decryptKey, iv_copy.data(), &num, AES_DECRYPT);
}
```

Рисунок 4.15 – Код AES шифрів

AES підтримує три різні розміри ключів: 128, 192 і 256 біт. Розмір ключа впливає на кілька важливих аспектів шифрування:

Безпека: Чим більший ключ, тим більше можливих комбінацій, що робить алгоритм більш стійким до атак перебором. 128-бітний ключ забезпечує достатній рівень безпеки для більшості сучасних застосувань, але 256-бітний ключ надає ще вищий рівень захисту.

Швидкість: Менший ключ зазвичай означає швидше шифрування та дешифрування. 128-бітний ключ зазвичай швидше, ніж 256-бітний, але різниця в продуктивності може бути незначною при використанні сучасного обладнання.

Ресурси: Розмір ключа впливає на використання пам'яті та обчислювальні ресурси. Більші ключі потребують більше обчислювальних ресурсів для обробки.

```
const validateKey = (key, keySize) => {
    const keyLength = key.length * 4;
    return keyLength === keySize;
};

const generateKey = () => {
    const keyLength = aesKeySize / 8;
    const generatedKey = Array.from(crypto.getRandomValues(new Uint8Array(keyLength)))
        .map(b => b.toString(16).padStart(2, '0'))
        .join('');
    setAesKey(generatedKey);
};
```

Рисунок 4.16 – Код створення ключа

З цими файлами алгоритмів тепер можна для наглядності створити примітивний інтерфейс який покаже як успішність виконання роботи програми так й задасть скелет для майбутнього дизайну.

Рисунок 4.17 – Первинний вигляд сторінки AES

Завершивши розробку AES можна перейти до розробки алгоритму RSA. Він складається з публічного ключа та тексту при шифруванні, приватного ключа та зашифрованого тексту при дешифруванні. Окрім цього RSA має в собі як й AES можливість обрати розмір ключа що впливає на генерацію ключа та режим шифрування.

```
Napi::Value Encrypt(const Napi::CallbackInfo& info) {
    Napi::Env env = info.Env();

    std::string publicKey = info[0].As<Napi::String>().Utf8Value();
    std::string text = info[1].As<Napi::String>().Utf8Value();
    std::string cipherType = info[2].As<Napi::String>().Utf8Value();

    BIO* bio = BIO_new_mem_buf(publicKey.data(), -1);
    RSA* pubKey = PEM_read_bio_RSA_PUBKEY(bio, NULL, NULL, NULL);
    BIO_free(bio);

    int padding;
    if (cipherType == "OAEP") {
        padding = RSA_PKCS1_OAEP_PADDING;
    } else if (cipherType == "PKCS1") {
        padding = RSA_PKCS1_PADDING;
    } else if (cipherType == "RSA") {
        padding = RSA_NO_PADDING;
    } else {
        throw std::invalid_argument("Unsupported cipher type");
    }

    std::vector<unsigned char> encryptedText(RSA_size(pubKey));
    int len = RSA_public_encrypt(text.size(), reinterpret_cast<const unsigned char*>(text.data()), encryptedText.data(), pubKey, padding);
    RSA_free(pubKey);

    if (len == -1) {
        throw std::runtime_error("RSA encryption failed");
    }

    return Napi::String::New(env, base64Encode(encryptedText.data(), len));
}
```

Рисунок 4.18 – Код RSA шифрування

```cpp
Napi::Value Decrypt(const Napi::CallbackInfo& info) {
    Napi::Env env = info.Env();

    std::string privateKey = info[0].As<Napi::String>().Utf8Value();
    std::string text = info[1].As<Napi::String>().Utf8Value();
    std::string cipherType = info[2].As<Napi::String>().Utf8Value();

    BIO* bio = BIO_new_mem_buf(privateKey.data(), -1);
    RSA* privKey = PEM_read_bio_RSAPrivateKey(bio, NULL, NULL, NULL);
    BIO_free(bio);

    std::vector<unsigned char> decodedText = base64Decode(text);
    std::vector<unsigned char> decryptedText(RSA_size(privKey));

    int padding;
    if (cipherType == "OAEP") {
        padding = RSA_PKCS1_OAEP_PADDING;
    } else if (cipherType == "PKCS1") {
        padding = RSA_PKCS1_PADDING;
    } else if (cipherType == "RSA") {
        padding = RSA_NO_PADDING;
    } else {
        throw std::invalid_argument("Unsupported cipher type");
    }

    int len = RSA_private_decrypt(decodedText.size(), decodedText.data(), decryptedText.data(), privKey, padding);
    RSA_free(privKey);

    if (len == -1) {
        throw std::runtime_error("RSA decryption failed");
    }

    return Napi::String::New(env, std::string(reinterpret_cast<const char*>(decryptedText.data()), len));
}
```

Рисунок 4.19 – Код RSA дешифрування

```cpp
Napi::Value GenerateKeys(const Napi::CallbackInfo& info) {
    Napi::Env env = info.Env();
    int keySize = info[0].As<Napi::Number>().Int32Value();

    BIGNUM* e = BN_new();
    BN_set_word(e, RSA_F4);

    // Generate RSA key pair with the specified key size
    RSA_generate_key_ex(rsa, keySize, e, NULL);
    BN_free(e);

    BIO* privBio = BIO_new(BIO_s_mem());
    BIO* pubBio = BIO_new(BIO_s_mem());

    PEM_write_bio_RSAPrivateKey(privBio, rsa, NULL, NULL, 0, NULL, NULL);
    PEM_write_bio_RSA_PUBKEY(pubBio, rsa);

    BUF_MEM* privBuffer;
    BUF_MEM* pubBuffer;

    BIO_get_mem_ptr(privBio, &privBuffer);
    BIO_get_mem_ptr(pubBio, &pubBuffer);

    std::string privKey(privBuffer->data, privBuffer->length);
    std::string pubKey(pubBuffer->data, pubBuffer->length);

    BIO_free_all(privBio);
    BIO_free_all(pubBio);

    Napi::Object keys = Napi::Object::New(env);
    keys.Set("publicKey", Napi::String::New(env, pubKey));
    keys.Set("privateKey", Napi::String::New(env, privKey));

    return keys;
```

Рисунок 4.20 – Код RSA генерації ключа

Далі продемонстровано те як саме працює готовий варіант генерації ключа, шифрування та дешифрування криптографічного алгоритму RSA.

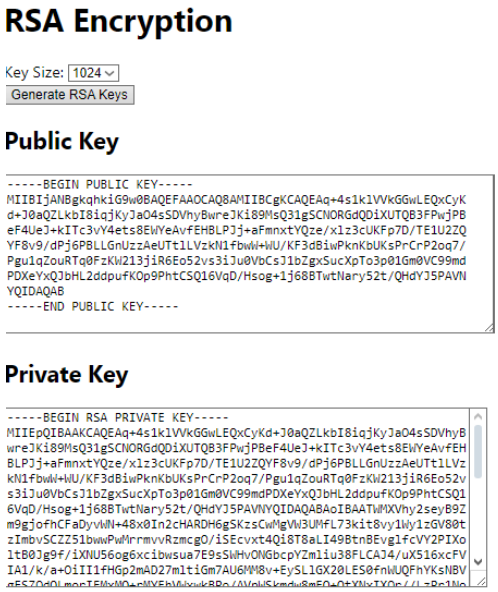


Рисунок 4.21 – Первинний вигляд елемента генерації ключа на сайті

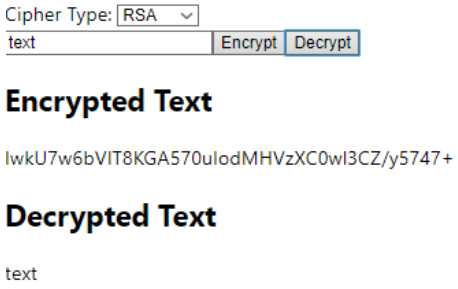
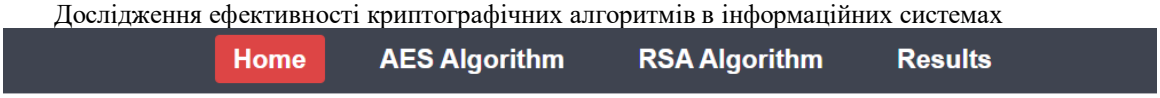


Рисунок 4.22 – Вигляд шифрування та дешифрування тексту на сайті

З цими файлами алгоритмів тепер можна для створити примітивний інтерфейс який покаже як успішність виконання роботи програми так й задасть скелет для майбутнього дизайну.

4.4 Розробка frontend

Пройшовши етап backend розробки цей розділ передбачає в собі створення дизайну сайту який буде надавати більш приємний вигляд застосунку. Так як вже було попередньо створено макет сайту то при створенні дизайну було відтворено як можна детально запланований вигляд. Надалі представлено готовий вигляд всіх сторінок.



Welcome to the Encryption App

Рисунок 4.23 – Головна сторінка

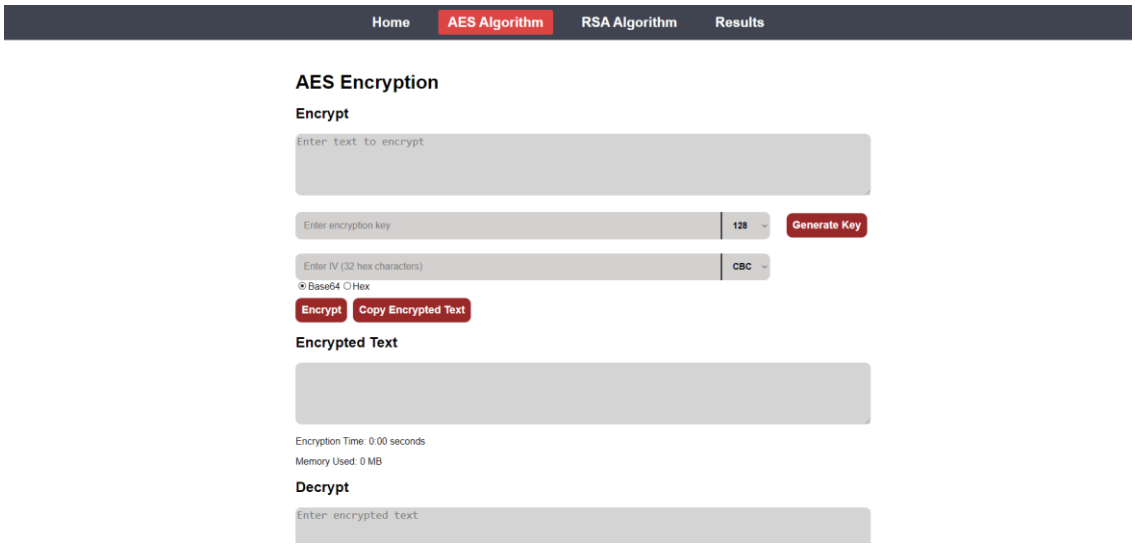
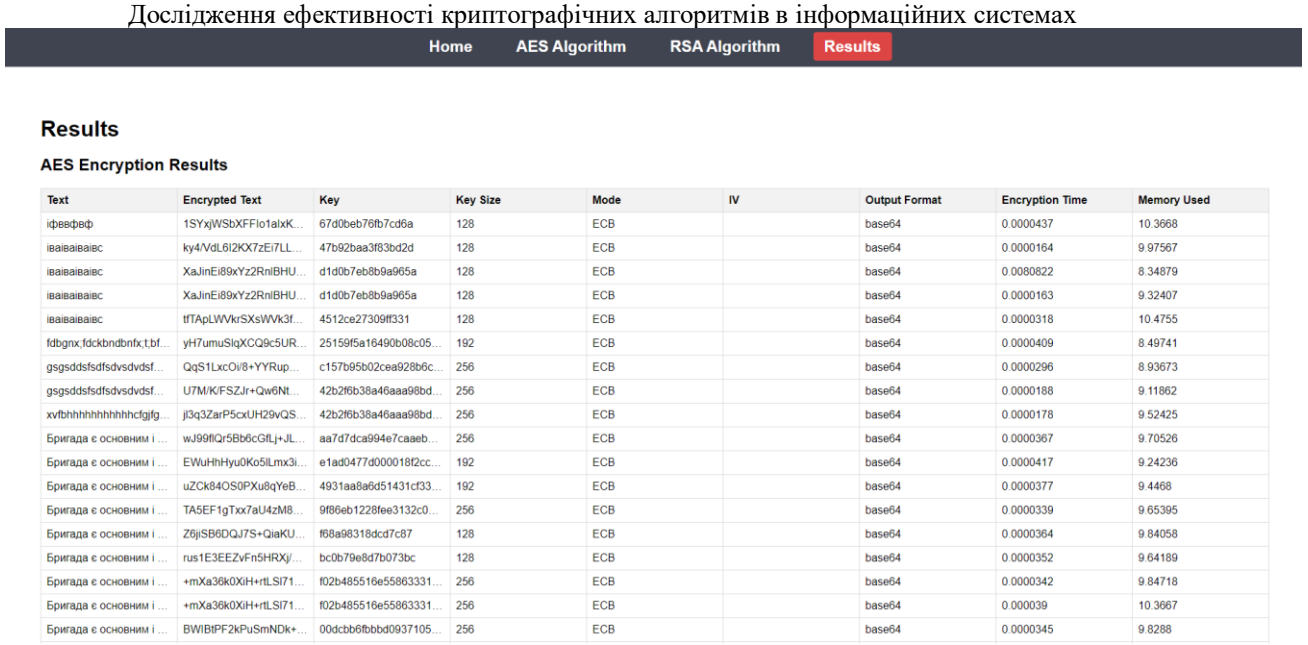


Рисунок 4.24 – Сторінка алгоритму AES

Рисунок 4.25 – Сторінка алгоритму RSA



Text	Encrypted Text	Key	Key Size	Mode	IV	Output Format	Encryption Time	Memory Used
іфвефвф	1SYxjWSbXFFlo1alxK...	67d0beb76fb7cd6a	128	ECB		base64	0.0000437	10.3668
іваіваіваівс	ky4/VdL6l2KX7zEi7LL...	47b92baa3f83bd2d	128	ECB		base64	0.0000164	9.97567
іваіваіваівс	XaJinEi89xYz2RnlBHU...	d1d0b7eb8b9a965a	128	ECB		base64	0.0080822	8.34879
іваіваіваівс	XaJinEi89xYz2RnlBHU...	d1d0b7eb8b9a965a	128	ECB		base64	0.0000163	9.32407
іваіваіваівс	tfTApLWVkrSXsWVk3f...	4512ce27309ff331	128	ECB		base64	0.0000318	10.4755
fdbgnx,fdckbndbnfx,t,bf...	yH7umuSlqXCQ9c5UR...	25159f5a16490b08c05...	192	ECB		base64	0.0000409	8.49741
gsgsddsfsdfsdvsdvdsf...	QqS1LxcOi/8+YYRup...	c157b95b02cea928b6c...	256	ECB		base64	0.0000296	8.93673
gsgsddsfsdfsdvsdvdsf...	U7M/K/FSZJr+Qw6Nt...	42b2f6b38a46aaa98bd...	256	ECB		base64	0.0000188	9.11862
xvfbhhhhhhhhhhhcfgjfg...	jl3q3ZarP5cxUH29vQS...	42b2f6b38a46aaa98bd...	256	ECB		base64	0.0000178	9.52425
Бригада є основним і ...	wJ99fiQr5Bb6cGfLj+JL...	aa7d7dca994e7caaeb...	256	ECB		base64	0.0000367	9.70526
Бригада є основним і ...	EWuHhHyu0Ko5iLmx3i...	e1ad0477d000018f2cc...	192	ECB		base64	0.0000417	9.24236
Бригада є основним і ...	uZCk84OS0PXu8qYeB...	4931aa8a6d51431cf33...	192	ECB		base64	0.0000377	9.4468
Бригада є основним і ...	TA5EF1gTxx7aU4zM8...	9f86eb1228fee3132c0...	256	ECB		base64	0.0000339	9.65395
Бригада є основним і ...	Z6jiSB6DQJ7S+QiaKU...	f68a98318dcd7c87	128	ECB		base64	0.0000364	9.84058
Бригада є основним і ...	rus1E3EEZvFn5HRXj/...	bc0b79e8d7b073bc	128	ECB		base64	0.0000352	9.64189
Бригада є основним і ...	+mXa36k0XiH+rtLSl71...	f02b485516e55863331...	256	ECB		base64	0.0000342	9.84718
Бригада є основним і ...	+mXa36k0XiH+rtLSl71...	f02b485516e55863331...	256	ECB		base64	0.000039	10.3667
Бригада є основним і ...	BWIBtPF2kPuSmNDk+...	00dcbb6fbbbd0937105...	256	ECB		base64	0.0000345	9.8288

Рисунок 4.26 – Сторінка результатів шифрування

Таким чином програма повністю готова як зі сторони backend так й зі сторони frontend.

4.5 Аналіз ефективності алгоритмів

Маючи готовим все що потрібно, можна провести аналіз ефективності алгоритмів і виявити їх сильні та слабкі сторони. Перш за все варто прийти до єдиної думки що таке ефективність в контексті дослідження. Алгоритм має бути швидким та надійним тому на основі цього твердження сформуємо розуміння того чим ж воно є. Ефективність це можливість алгоритму з задіянням мінімальної кількості часу та ресурсів комп’ютера на операцію забезпечити стійке до атаки шифрування даних. На основі цього проведемо аналіз швидкості та надійності алгоритмів.

AES за рахунок того що є симетричним алгоритмом є набагато швидшим за RSA який є асиметричним. Для того щоб зашифрувати або дешифрувати дані AES потребує лише публічного ключа в обох операціях в той час як RSA потрібно для шифрування використати публічний ключ а для дешифрування приватний. Зокрема можна виділити такий важливий момент як довжина ключа. В AES воно представлено зазвичай 128, 196, 256 біт в той час як в RSA ключ починається від 1024 і може

збільшуватись на набагато більшу кількість бітів що суттєво сповільнює його.

Розглядаючи генерацію ключів AES однією з ключових особливостей є те, що він побудований на мережі S-P, у якій весь 128-бітний вхідний блок організований як масив 4x4 байти під назвою State і обробляється в кілька раундів. Кількість використовуваних раундів залежить від довжини ключа наприклад 10 раундів для 128-бітного ключа, 12 раундів для 192-бітного ключа та 14 раундів для 256-бітного ключа. І в процесі шифрування, і в процесі дешифрування масив State змінюється на кожному раунді функцією раунду, яка визначає чотири різні байт-орієнтовані перетворення [11]. В RSA ж це відбувається за рахунок вибору двох великих простих числа p та q такі що $p \neq q$ після чого вираховується модуль $n=p \cdot q$ і його вже застосовують у функції Ейлера.

Таблиця 2.2 – час шифрування алгоритмів

Алгоритм	Розмір файлу (KB)	Час шифрування (секунди)	Час дешифрування (секунди)
AES	120	1.7	1.2
RSA		7.4	4.9
AES	154	1.8	1.3
RSA		10.1	5.1
AES	315	1.8	1.3
RSA		8.6	5.9
AES	864	2.1	1.3
RSA		8.3	5.2
AES	196	1.9	1.4
RSA		7.9	5.2

Варто також розглянути безпеку цих алгоритмів. RSA з часом показав що він може бути слабким до таких видів атак як Brute force та математична атака. Кожна з цих

вразливостей була вирішена рекомендованою довжиною ключа. Так для Brute force це стало 1024 а для математичної атаки 2048 з-за чого на даний момент він вважається достатньо безпечним за умови жертви часу. Проте з часом RSA під плином розвитку технологій зокрема за умови квантових комп'ютерів буде дедалі складніше утримувати планку надійного алгоритму. Зате ця проблема ніяк не перетинається на даний момент з AES з-за кількості можливих комбінацій. Лише в одному розмірі ключа на 128 біт існує 2^{128} комбінацій що робить Brute force не ефективним а математична атака на даний момент не має взламуючого алгоритму який міг би виявити та використати вразливість алгоритму. Може здатись що AES є найкращим алгоритмом але попри всю свою безпеку він залишається симетричним алгоритмом що змушує користувача якось передавати публічний ключ іншому користувачу щоб дешифрування даних було можливим. Саме для цього зараз використовують алгоритми асиметричного типу зокрема RSA. Таким чином використовуючи їх в парі можна досягти достатньо ефективної комбінації.

Висновки до розділу 4

У четвертому розділі кваліфікаційної роботи бакалавра детально розглядається процес створення БД, backend та frontend частини проєкту. Розглядається кожен алгоритм та надається характеристика їх особливостей. Розглядається сильні та слабкі сторони алгоритмів та те які можливості вони мають.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи бакалавра реалізовано алгоритми RSA та AES та проведено дослідження ефективності криптографічних алгоритмів в інформаційних системах.

Для досягнення поставленої мети виконано наступні завдання:

1) проведено аналіз предметної сфери криптографічних алгоритмів;

2) проведено аналіз існуючих аналогів програмного забезпечення криптографічного захисту інформації, визначення їх переваг та недоліків;

3) обрано криптографічні алгоритми для програмної реалізації та дослідження їх ефективності;

4) сформульовано специфікація вимог до програмного забезпечення системи;

5) проведено моделювання та проєктування програмної системи криптографічного захисту;

6) програмно реалізовано та протестовано систему криптографічного захисту;

7) проведено аналіз отриманих результатів ефективності застосування аналізованих криптоалгоритмів.

Практичне значення отриманих результатів полягає у тому що якісний та функціональний набір криптографічних алгоритмів може сприяти значному покращенню безпеки різних систем а комбінація асиметричних та симетричних алгоритмів може ще більше цьому сприяти. Розроблений застосунок може бути вдосконалений за допомогою розширення функціоналу та додавання нових можливостей

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Analog application: Crypto Tools. URL: https://crypto-tools.streamlit.app/ (Last accessed: 02.05.2024).

2. Analog application: CyberChef. URL: https://gchq.github.io/CyberChef/ (Last accessed: 02.05.2024).

3. Analog application: Cryptii. URL: https://cryptii.com/ (Last accessed: 02.05.2024).

4. UML Use Case Diagram Tutorial | Lucidchart. URL: https://www.lucidchart.com/pages/uml-use-case-diagram (Last accessed: 02.05.2024).

5. Katz J., Lindell Y. Introduction to Modern Cryptography. CRC PRESS, 2007. 552 p. (Last accessed: 02.05.2024).

6. A Brief Overview of RSA Encryption. URL: https://www.geeksforgeeks.org/rsa-algorithm-cryptography/ (Last accessed: 02.05.2024).

7. Introduction to the AES Encryption Algorithm. URL: https://developer.ibm.com/tutorials/cc-aes-intro/ (Last accessed: 02.05.2024).

8. C. Paar, Understanding Cryptography, Springer, 1998. 173 p.

9. AES (Advanced Encryption Standard). URL: https://www.techopedia.com/definition/4157/advanced-encryption-standard-aes (Last accessed: 02.05.2024).

10. What is Package Diagram?. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/ (Last accessed: 02.05.2024).

11. FIPS 197 ADVANCED ENCRYPTION STANDARD (AES). URL: https://csrc.nist.gov/files/pubs/fips/197/final/docs/fips-197.pdf 2001. 9 p. (Last accessed: 10.06.2024).

ДОДАТОК А

aes.cpp

```cpp
#include <napi.h>
#include <openssl/aes.h>
#include <openssl/rand.h>
#include <openssl/evp.h>
#include <openssl/bio.h>
#include <openssl/buffer.h>
#include <string>
#include <vector>
#include <iostream>

std::string base64Encode(const unsigned char* buffer, size_t length) {
    BIO *bio, *b64;
    BUF_MEM *bufferPtr;

    b64 = BIO_new(BIO_f_base64());
    bio = BIO_new(BIO_s_mem());
    bio = BIO_push(b64, bio);

    BIO_set_flags(bio, BIO_FLAGS_BASE64_NO_NL);
    BIO_write(bio, buffer, length);
    BIO_flush(bio);
    BIO_get_mem_ptr(bio, &bufferPtr);

    std::string encodedData(bufferPtr->data, bufferPtr->length);
    BIO_free_all(bio);

    return encodedData;
}

std::vector<unsigned char> base64Decode(const std::string &base64String) {
    BIO *bio, *b64;

    int decodeLen = static_cast<int>(base64String.size());
    std::vector<unsigned char> buffer(decodeLen);

    bio = BIO_new_mem_buf(base64String.c_str(), -1);
    b64 = BIO_new(BIO_f_base64());
    bio = BIO_push(b64, bio);

    BIO_set_flags(bio, BIO_FLAGS_BASE64_NO_NL);
    decodeLen = BIO_read(bio, buffer.data(), static_cast<int>(base64String.size()));
    buffer.resize(decodeLen);
    BIO_free_all(bio);
```

```
    return buffer;
}

class AESWrapper : public Napi::ObjectWrap<AESWrapper> {
public:
    static Napi::Object Init(Napi::Env env, Napi::Object exports);
    AESWrapper(const Napi::CallbackInfo& info);
    Napi::Value Encrypt(const Napi::CallbackInfo& info);
    Napi::Value Decrypt(const Napi::CallbackInfo& info);
    Napi::Value GetIV(const Napi::CallbackInfo& info);

private:
    AES_KEY encryptKey;
    AES_KEY decryptKey;
    std::vector<unsigned char> iv;
    std::string mode;

    void AES_ECB_encrypt(const std::vector<unsigned char>& plainText,
std::vector<unsigned char>& cipherText);
    void AES_ECB_decrypt(const std::vector<unsigned char>& cipherText,
std::vector<unsigned char>& plainText);

    void AES_CBC_encrypt(const std::vector<unsigned char>& plainText,
std::vector<unsigned char>& cipherText);
    void AES_CBC_decrypt(const std::vector<unsigned char>& cipherText,
std::vector<unsigned char>& plainText);

    void AES_CFB_encrypt(const std::vector<unsigned char>& plainText,
std::vector<unsigned char>& cipherText);
    void AES_CFB_decrypt(const std::vector<unsigned char>& cipherText,
std::vector<unsigned char>& plainText);

    void handlePadding(std::vector<unsigned char>& text, bool addPadding);
};

Napi::Object AESWrapper::Init(Napi::Env env, Napi::Object exports) {
    Napi::Function func = DefineClass(env, "AESWrapper", {
        InstanceMethod("encrypt", &AESWrapper::Encrypt),
        InstanceMethod("decrypt", &AESWrapper::Decrypt),
        InstanceMethod("getIV", &AESWrapper::GetIV)
    });

    Napi::FunctionReference* constructor = new Napi::FunctionReference();
    *constructor = Napi::Persistent(func);
    exports.Set("AESWrapper", func);

    return exports;
```

```
}

AESWrapper::AESWrapper(const Napi::CallbackInfo& info) :
Napi::ObjectWrap<AESWrapper>(info) {
    std::string key = info[0].As<Napi::String>().Utf8Value();
    this->mode = info[1].As<Napi::String>().Utf8Value();
    this->iv = std::vector<unsigned char>(AES_BLOCK_SIZE);

    if (mode == "CBC" || mode == "CFB") {
        if (info.Length() > 2) {
            std::string ivStr = info[2].As<Napi::String>().Utf8Value();
            if (ivStr.size() != AES_BLOCK_SIZE) {
                Napi::Error::New(info.Env(), "IV must be 16 bytes for CBC and CFB
modes").ThrowAsJavaScriptException();
                return;
            }
            std::copy(ivStr.begin(), ivStr.end(), this->iv.begin());
        } else {
            Napi::Error::New(info.Env(), "IV must be provided for CBC and CFB
modes").ThrowAsJavaScriptException();
            return;
        }
    }

    int keyLength = key.size() * 8;
    if (keyLength != 128 && keyLength != 192 && keyLength != 256) {
        Napi::Error::New(info.Env(), "Key must be 128, 192, or 256
bits").ThrowAsJavaScriptException();
        return;
    }
    AES_set_encrypt_key(reinterpret_cast<const unsigned char*>(key.data()), keyLength,
&encryptKey);
    AES_set_decrypt_key(reinterpret_cast<const unsigned char*>(key.data()), keyLength,
&decryptKey);
}

void AESWrapper::AES_ECB_encrypt(const std::vector<unsigned char>& plainText,
std::vector<unsigned char>& cipherText) {
    for (size_t i = 0; i < plainText.size(); i += AES_BLOCK_SIZE) {
        AES_encrypt(plainText.data() + i, cipherText.data() + i, &encryptKey);
    }
}

void AESWrapper::AES_ECB_decrypt(const std::vector<unsigned char>& cipherText,
std::vector<unsigned char>& plainText) {
    for (size_t i = 0; i < cipherText.size(); i += AES_BLOCK_SIZE) {
        AES_decrypt(cipherText.data() + i, plainText.data() + i, &decryptKey);
```

```
    }
}

void AESWrapper::AES_CBC_encrypt(const std::vector<unsigned char>& plainText,
std::vector<unsigned char>& cipherText) {
    std::vector<unsigned char> iv_copy = this->iv;
    AES_cbc_encrypt(plainText.data(), cipherText.data(), plainText.size(), &encryptKey,
iv_copy.data(), AES_ENCRYPT);
}

void AESWrapper::AES_CBC_decrypt(const std::vector<unsigned char>& cipherText,
std::vector<unsigned char>& plainText) {
    std::vector<unsigned char> iv_copy = this->iv;
    AES_cbc_encrypt(cipherText.data(), plainText.data(), cipherText.size(), &decryptKey,
iv_copy.data(), AES_DECRYPT);
}

void AESWrapper::AES_CFB_encrypt(const std::vector<unsigned char>& plainText,
std::vector<unsigned char>& cipherText) {
    std::vector<unsigned char> iv_copy = this->iv;
    int num = 0;
    AES_cfb128_encrypt(plainText.data(), cipherText.data(), plainText.size(),
&encryptKey, iv_copy.data(), &num, AES_ENCRYPT);
}

void AESWrapper::AES_CFB_decrypt(const std::vector<unsigned char>& cipherText,
std::vector<unsigned char>& plainText) {
    std::vector<unsigned char> iv_copy = this->iv;
    int num = 0;
    AES_cfb128_encrypt(cipherText.data(), plainText.data(), cipherText.size(),
&decryptKey, iv_copy.data(), &num, AES_DECRYPT);
}

void AESWrapper::handlePadding(std::vector<unsigned char>& text, bool addPadding) {
    if (addPadding) {
        size_t padding = AES_BLOCK_SIZE - (text.size() % AES_BLOCK_SIZE);
        text.insert(text.end(), padding, static_cast<unsigned char>(padding));
    } else {
        size_t padding = text.back();
        if (padding > AES_BLOCK_SIZE) {
            throw std::runtime_error("Invalid padding");
        }
        text.resize(text.size() - padding);
    }
}

Napi::Value AESWrapper::Encrypt(const Napi::CallbackInfo& info) {
```

```
    Napi::Env env = info.Env();
    Napi::HandleScope scope(env);

    std::string plainText = info[0].As<Napi::String>().Utf8Value();
    std::vector<unsigned char> paddedText(plainText.begin(), plainText.end());
    handlePadding(paddedText, true);

    std::vector<unsigned char> encryptedText(paddedText.size());

    if (this->mode == "ECB") {
        AES_ECB_encrypt(paddedText, encryptedText);
    } else if (this->mode == "CBC") {
        AES_CBC_encrypt(paddedText, encryptedText);
    } else if (this->mode == "CFB") {
        AES_CFB_encrypt(paddedText, encryptedText);
    } else {
        Napi::Error::New(env, "Unsupported mode").ThrowAsJavaScriptException();
        return env.Null();
    }

    return Napi::String::New(env, base64Encode(encryptedText.data(),
encryptedText.size()));
}

Napi::Value AESWrapper::Decrypt(const Napi::CallbackInfo& info) {
    Napi::Env env = info.Env();
    Napi::HandleScope scope(env);

    std::vector<unsigned char> decodedText =
base64Decode(info[0].As<Napi::String>().Utf8Value());
    std::vector<unsigned char> decryptedText(decodedText.size());

    if (this->mode == "ECB") {
        AES_ECB_decrypt(decodedText, decryptedText);
    } else if (this->mode == "CBC") {
        AES_CBC_decrypt(decodedText, decryptedText);
    } else if (this->mode == "CFB") {
        AES_CFB_decrypt(decodedText, decryptedText);
    } else {
        Napi::Error::New(env, "Unsupported mode").ThrowAsJavaScriptException();
        return env.Null();
    }

    try {
        handlePadding(decryptedText, false);
    } catch (const std::runtime_error& e) {
        Napi::Error::New(env, e.what()).ThrowAsJavaScriptException();
```

```
        return env.Null();
    }

    return Napi::String::New(env, std::string(decryptedText.begin(),
decryptedText.end()));
}

Napi::Value AESWrapper::GetIV(const Napi::CallbackInfo& info) {
    Napi::Env env = info.Env();
    return Napi::String::New(env, base64Encode(this->iv.data(), this->iv.size()));
}

Napi::Object InitAll(Napi::Env env, Napi::Object exports) {
    return AESWrapper::Init(env, exports);
}

NODE_API_MODULE(NODE_GYP_MODULE_NAME, InitAll)
```

ДОДАТОК Б

rsa.cpp

```cpp
#include <napi.h>
#include <openssl/rsa.h>
#include <openssl/pem.h>
#include <openssl/err.h>
#include <openssl/bio.h>
#include <openssl/buffer.h>
#include <string>
#include <vector>

#define CRYPTO_RSA_KEY_LEN_4096 4096
#define CRYPTO_RSA_KEY_LEN_2048 2048
#define CRYPTO_RSA_KEY_LEN_1024 1024

std::string base64Encode(const unsigned char* buffer, size_t length) {
    BIO *bio, *b64;
    BUF_MEM *bufferPtr;

    b64 = BIO_new(BIO_f_base64());
    bio = BIO_new(BIO_s_mem());
    bio = BIO_push(b64, bio);

    BIO_set_flags(bio, BIO_FLAGS_BASE64_NO_NL);
    BIO_write(bio, buffer, length);
    BIO_flush(bio);
    BIO_get_mem_ptr(bio, &bufferPtr);

    std::string encodedData(bufferPtr->data, bufferPtr->length);
    BIO_free_all(bio);

    return encodedData;
}

std::vector<unsigned char> base64Decode(const std::string &base64String) {
    BIO *bio, *b64;

    int decodeLen = static_cast<int>(base64String.size());
    std::vector<unsigned char> buffer(decodeLen);

    bio = BIO_new_mem_buf(base64String.c_str(), -1);
    b64 = BIO_new(BIO_f_base64());
    bio = BIO_push(b64, bio);

    BIO_set_flags(bio, BIO_FLAGS_BASE64_NO_NL);
    decodeLen = BIO_read(bio, buffer.data(), static_cast<int>(base64String.size()));
```

```
    buffer.resize(decodeLen);
    BIO_free_all(bio);

    return buffer;
}

class RSAWrapper : public Napi::ObjectWrap<RSAWrapper> {
public:
    static Napi::Object Init(Napi::Env env, Napi::Object exports) {
        Napi::Function func = DefineClass(env, "RSAWrapper", {
            InstanceMethod("generateKeys", &RSAWrapper::GenerateKeys),
            InstanceMethod("encrypt", &RSAWrapper::Encrypt),
            InstanceMethod("decrypt", &RSAWrapper::Decrypt)
        });

        constructor = Napi::Persistent(func);
        constructor.SuppressDestruct();

        exports.Set("RSAWrapper", func);
        return exports;
    }

    RSAWrapper(const Napi::CallbackInfo& info) : Napi::ObjectWrap<RSAWrapper>(info) {
        rsa = RSA_new();
    }

    ~RSAWrapper() {
        RSA_free(rsa);
    }

    Napi::Value GenerateKeys(const Napi::CallbackInfo& info) {
        Napi::Env env = info.Env();
        int keySize = info[0].As<Napi::Number>().Int32Value();

        BIGNUM* e = BN_new();
        BN_set_word(e, RSA_F4);

        RSA_generate_key_ex(rsa, keySize, e, NULL);
        BN_free(e);

        BIO* privBio = BIO_new(BIO_s_mem());
        BIO* pubBio = BIO_new(BIO_s_mem());

        PEM_write_bio_RSAPrivateKey(privBio, rsa, NULL, NULL, 0, NULL, NULL);
        PEM_write_bio_RSA_PUBKEY(pubBio, rsa);

        BUF_MEM* privBuffer;
```

```
    BUF_MEM* pubBuffer;

    BIO_get_mem_ptr(privBio, &privBuffer);
    BIO_get_mem_ptr(pubBio, &pubBuffer);

    std::string privKey(privBuffer->data, privBuffer->length);
    std::string pubKey(pubBuffer->data, pubBuffer->length);

    BIO_free_all(privBio);
    BIO_free_all(pubBio);

    Napi::Object keys = Napi::Object::New(env);
    keys.Set("publicKey", Napi::String::New(env, pubKey));
    keys.Set("privateKey", Napi::String::New(env, privKey));

    return keys;
}

Napi::Value Encrypt(const Napi::CallbackInfo& info) {
    Napi::Env env = info.Env();

    std::string publicKey = info[0].As<Napi::String>().Utf8Value();
    std::string text = info[1].As<Napi::String>().Utf8Value();
    std::string cipherType = info[2].As<Napi::String>().Utf8Value();

    BIO* bio = BIO_new_mem_buf(publicKey.data(), -1);
    RSA* pubKey = PEM_read_bio_RSA_PUBKEY(bio, NULL, NULL, NULL);
    BIO_free(bio);

    int padding;
    if (cipherType == "OAEP") {
        padding = RSA_PKCS1_OAEP_PADDING;
    } else if (cipherType == "PKCS1") {
        padding = RSA_PKCS1_PADDING;
    } else if (cipherType == "RSA") {
        padding = RSA_NO_PADDING;
    } else {
        throw std::invalid_argument("Unsupported cipher type");
    }

    std::vector<unsigned char> encryptedText(RSA_size(pubKey));
    int len = RSA_public_encrypt(text.size(), reinterpret_cast<const unsigned
char*>(text.data()), encryptedText.data(), pubKey, padding);
    RSA_free(pubKey);

    if (len == -1) {
        throw std::runtime_error("RSA encryption failed");
```

```
        }

        return Napi::String::New(env, base64Encode(encryptedText.data(), len));
    }

    Napi::Value Decrypt(const Napi::CallbackInfo& info) {
        Napi::Env env = info.Env();

        std::string privateKey = info[0].As<Napi::String>().Utf8Value();
        std::string text = info[1].As<Napi::String>().Utf8Value();
        std::string cipherType = info[2].As<Napi::String>().Utf8Value();

        BIO* bio = BIO_new_mem_buf(privateKey.data(), -1);
        RSA* privKey = PEM_read_bio_RSAPrivateKey(bio, NULL, NULL, NULL);
        BIO_free(bio);

        std::vector<unsigned char> decodedText = base64Decode(text);
        std::vector<unsigned char> decryptedText(RSA_size(privKey));

        int padding;
        if (cipherType == "OAEP") {
            padding = RSA_PKCS1_OAEP_PADDING;
        } else if (cipherType == "PKCS1") {
            padding = RSA_PKCS1_PADDING;
        } else if (cipherType == "RSA") {
            padding = RSA_NO_PADDING;
        } else {
            throw std::invalid_argument("Unsupported cipher type");
        }

        int len = RSA_private_decrypt(decodedText.size(), decodedText.data(),
decryptedText.data(), privKey, padding);
        RSA_free(privKey);

        if (len == -1) {
            throw std::runtime_error("RSA decryption failed");
        }

        return Napi::String::New(env, std::string(reinterpret_cast<const
char*>(decryptedText.data()), len));
    }

private:
    RSA* rsa;
    static Napi::FunctionReference constructor;
};
```

```
Napi::FunctionReference RSAWrapper::constructor;

Napi::Object Init(Napi::Env env, Napi::Object exports) {
    return RSAWrapper::Init(env, exports);
}

NODE_API_MODULE(NODE_GYP_MODULE_NAME, Init)
```

ДОДАТОК В

Index.js

```
const express = require('express');
const cors = require('cors');
const mysql = require('mysql2/promise');
const fs = require('fs');
const aesAddon = require('./build/Release/aes');
const rsaAddon = require('./build/Release/rsa');
const { performance } = require('perf_hooks');

const app = express();
app.use(cors());
app.use(express.json());

const logStream = fs.createWriteStream('server.log', { flags: 'a' });

const pool = mysql.createPool({
    host: 'localhost',
    user: 'root',
    password: '1111',
    database: 'encryption_db'
});

app.get('/', (req, res) => {
    res.send('Welcome to the AES and RSA Encryption Server');
});

app.post('/encrypt-aes', async (req, res) => {
    try {
        const { text, key, keySize, mode, iv, outputFormat } = req.body;
        const aesInstance = new aesAddon.AESWrapper(key, mode, iv);

        const startMemory = process.memoryUsage().heapUsed / 1024 / 1024;
        const startTime = performance.now();

        const encryptedText = aesInstance.encrypt(text);

        const endTime = performance.now();
        const endMemory = process.memoryUsage().heapUsed / 1024 / 1024;

        const encryptionTime = (endTime - startTime) / 1000;
        const memoryUsed = endMemory - startMemory;

        await pool.query('INSERT INTO aes_enc (user_text, enc_key, key_size, enc_mode,
iv, output_format, result, encryption_time, memory_used) VALUES (?, ?, ?, ?, ?, ?, ?, ?,
?)',
```

```
        [text, key, keySize, mode, iv, outputFormat, encryptedText, encryptionTime,
memoryUsed]);

        res.json({ encryptedText, encryptionTime, memoryUsed });
    } catch (error) {
        logStream.write(`[${new Date().toISOString()}] Error in /encrypt-aes:
${error.stack}\n`);
        console.error('Error in /encrypt-aes:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
});

app.post('/decrypt-aes', async (req, res) => {
    try {
        const { text, key, keySize, mode, iv } = req.body;
        const aesInstance = new aesAddon.AESWrapper(key, mode, iv);

        const startMemory = process.memoryUsage().heapUsed / 1024 / 1024;
        const startTime = performance.now();

        const decryptedText = aesInstance.decrypt(text);

        const endTime = performance.now();
        const endMemory = process.memoryUsage().heapUsed / 1024 / 1024;

        const decryptionTime = (endTime - startTime) / 1000;
        const memoryUsed = endMemory - startMemory;

        await pool.query('INSERT INTO aes_dec (enc_text, enc_key, key_size, dec_mode, iv,
output_format, result, dec_time, memory_used) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)',
        [text, key, keySize, mode, iv, 'base64', decryptedText, decryptionTime,
memoryUsed]);

        res.json({ decryptedText, decryptionTime, memoryUsed });
    } catch (error) {
        logStream.write(`[${new Date().toISOString()}] Error in /decrypt-aes:
${error.stack}\n`);
        console.error('Error in /decrypt-aes:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
});

app.post('/generate-rsa-keys', (req, res) => {
    try {
        const rsaInstance = new rsaAddon.RSAWrapper();
        const keys = rsaInstance.generateKeys();
        res.json(keys);
```

```
    } catch (error) {
        logStream.write(`[${new Date().toISOString()}] Error in /generate-rsa-keys:
${error.stack}\n`);
        console.error('Error in /generate-rsa-keys:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
});

app.post('/encrypt-rsa', async (req, res) => {
    try {
        const { text, publicKey, keySize, cipherType } = req.body;
        const rsaInstance = new rsaAddon.RSAWrapper();

        const startMemory = process.memoryUsage().heapUsed / 1024 / 1024;
        const startTime = performance.now();

        const encryptedText = rsaInstance.encrypt(publicKey, text, cipherType);

        const endTime = performance.now();
        const endMemory = process.memoryUsage().heapUsed / 1024 / 1024;

        const encryptionTime = (endTime - startTime) / 1000;
        const memoryUsed = endMemory - startMemory;

        await pool.query('INSERT INTO rsa_enc (enc_text, public_key, cipher_type,
private_key, key_size, enc_time, memory_used) VALUES (?, ?, ?, ?, ?, ?, ?)',
        [text, publicKey, cipherType, null, keySize, encryptionTime, memoryUsed]);

        res.json({ encryptedText, encryptionTime, memoryUsed });
    } catch (error) {
        logStream.write(`[${new Date().toISOString()}] Error in /encrypt-rsa:
${error.stack}\n`);
        console.error('Error in /encrypt-rsa:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
});

app.post('/decrypt-rsa', async (req, res) => {
    try {
        const { text, privateKey, keySize, cipherType } = req.body;
        const rsaInstance = new rsaAddon.RSAWrapper();

        const startMemory = process.memoryUsage().heapUsed / 1024 / 1024;
        const startTime = performance.now();

        const decryptedText = rsaInstance.decrypt(privateKey, text, cipherType);
```

```
        const endTime = performance.now();
        const endMemory = process.memoryUsage().heapUsed / 1024 / 1024;

        const decryptionTime = (endTime - startTime) / 1000;
        const memoryUsed = endMemory - startMemory;

        await pool.query('INSERT INTO rsa_dec (encrypted_text, private_key, result,
cipher_type, key_size, decryption_time, memory_used) VALUES (?, ?, ?, ?, ?, ?, ?)',
        [text, privateKey, decryptedText, cipherType, keySize, decryptionTime,
memoryUsed]);

        res.json({ decryptedText, decryptionTime, memoryUsed });
    } catch (error) {
        logStream.write(`[${new Date().toISOString()}] Error in /decrypt-rsa:
${error.stack}\n`);
        console.error('Error in /decrypt-rsa:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
});

app.get('/results/aes', async (req, res) => {
    try {
        const [rows] = await pool.query('SELECT * FROM aes_enc');
        res.json(rows);
    } catch (error) {
        logStream.write(`[${new Date().toISOString()}] Error in /results/aes:
${error.stack}\n`);
        console.error('Error in /results/aes:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
});

app.get('/results/rsa', async (req, res) => {
    try {
        const [rows] = await pool.query('SELECT * FROM rsa_enc');
        res.json(rows);
    } catch (error) {
        logStream.write(`[${new Date().toISOString()}] Error in /results/rsa:
${error.stack}\n`);
        console.error('Error in /results/rsa:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
});

const PORT = process.env.PORT || 3001;

app.listen(PORT, () => {
```

```
    console.log(`Server is running on port ${PORT}`);
});

process.on('uncaughtException', (err) => {
    logStream.write(`[${new Date().toISOString()}] Uncaught Exception: ${err.stack}\n`);
    console.error('Uncaught Exception:', err);
});

process.on('unhandledRejection', (reason, promise) => {
    logStream.write(`[${new Date().toISOString()}] Unhandled Rejection at: ${promise},
reason: ${reason.stack || reason}\n`);
    console.error('Unhandled Rejection at:', promise, 'reason:', reason);
});
```