Київський національний торговельно-економічний університет
Кафедра програмної інженерії та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

Розробка web-додатку «En-code» кодування та декодування текстової інформації

Студента 4 курсу, 7 групи,
напрям підготовки
6.050103
«Програмна інженерія»
спеціалізації
«Інженерія програмного
забезпечення»

Бородая Микити
Андрійовича

_____
підпис студента

Науковий керівник
кандидат технічних наук,
доцент

Цензура Микола
Олександрович

_____
підпис керівника

Гарант освітньої програми
кандидат технічних наук,
доцент

Цензура Микола
Олександрович

_____
підпис керівника

КИЇВ – 2019

ЗМІСТ

ВСТУП

Захист інформації в глобальній мережі Інтернет має важливу цінність. Пояснюється це, тим що, користувачі вказують особисті дані, яка слідом передаються на сервер. Передача інформації між браузером користувача та сервером не завжди є надійною і це може привести до витоку цієї інформації. Щоб уникнути такого результату використовують розширення протоколу HTTPS, який у свою чергу застосовує криптографічні протоколи SSL і TLS.

На даний момент за статистичними результатами досліджень, більша частина веб-сайтів використовує протокол HTTPS. Криптографічні технології постійно розвиваються, щоб надати максимальний захист інформації.

Протокол SSL використовує асиметричний алгоритм шифрування RSA, який буде розглянутий і проаналізований в даному проекті. Також, щоб надати повне уявлення про криптографічні алгоритми шифрування, будуть розглянуті симетричні алгоритми та головні переваги використання над асиметричними алгоритмами.

Актуальність. Застосування криптосистем в глобальній мережі Інтернет є актуальним рішенням захисту даних. Головним засобом є кодування даних за допомогою алгоритмів кодування.

Об’єктом дослідження є методи та засоби захисту інформації в глобальній мережі Інтернет.

Предметом дослідження є розробка веб-додатку кодування та декодування текстової інформації.

Метою випускного кваліфікаційного проекту є аналіз криптографічних алгоритмів кодування та розробка веб-додатку, який надає можливості кодування та декодування текстової інформації.

Завдання дослідження:

- Визначити вимоги та рекомендації до виконання проекту;
- Проаналізувати криптографічні засоби захисту даних;
- Розглянути симетричні алгоритми та режими шифрування;
- Розглянути асиметричний алгоритм RSA;
- Спроектувати та розробити серверне забезпечення;
- Спроектувати таблиці бази даних;
- Спроектувати та розробити клієнтську частину.

РОЗДІЛ 1. ВИМОГИ ТА РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ ПРОЕКТУ

1.1 Опис предметної області

Веб-додаток передбачає пряму взаємодію з користувачем, з метою отримання від них даних і видачі результату. Веб-сайт у свою чергу виступає як інформаційний ресурс і орієнтується більше на віддачу контенту.

Веб-додатки використовуються багатьма комерційними організаціями для створення внутрішніх систем. Крім цього веб-додатки можуть використовуватися некомерційними організаціями для надання простих і складних систем для користувачів глобальної мережі Інтернет. Наприклад, веб-додатки Google Docs або Codesandbox.

Веб-додаток реалізовує динамічний зміст, це означає, що сторінки не потребують перезавантаження, таким чином набагато покращується процес використування .

1.2 Технічне завдання до розробки веб-додатку

1. Загальні вимоги

Необхідно створити простий та зрозумілий за інтерфейсом веб-додаток, який виконує функції шифрування та дешифрування текстової інформації застосовуючи асиметричні та симетричні алгоритми шифрування.

1.1 Вимоги до інструментів розробки

1.1.1. Інструменти для розробки серверної частини

- Програмна платформа Node.js та фреймворк Epress;
- Криптографічна бібліотека crypto (OpenSSL).

1.1.2. . Інструменти для розробки клієнтської частини

- мова розмітки HTML версії 5;
- таблиця стилів CSS версії 3;
- бібліотека для розробки інтерфейсів React;
- webpack – інструмент для складання модульних файлів JavaScript;
- babel – компілятор сучасного JavaScript-коду для підтримки старих браузерів.

2. Вимоги до розробки серверного забезпечення.

2.1. Вимоги до алгоритмів шифрування.

Серверне забезпечення має реалізовувати функціонал наступних алгоритмів шифрування: RSA, AES, Cast5, Camellia, Blowfish, DES та Triple DES.

2.2. Вимоги до реалізації функцій шифрування та дешифрування.

- вектор ініціалізації повинен знаходитися спочатку зашифрованого тексту;
- застосування режимів шифрування;
- довжина ключа асиметричного алгоритму RSA повинна бути не менше 2048 бітів;
- застосування криптографічної бібліотеки OpenSSL;

2.3. Вимоги до захисту обміну даних.

Для організації обміну даних між сервером та браузером, повинен використовуватися протокол HTTP та його розширення HTTPS для безпечного обміну даних.

3. Вимоги до розробки клієнтської частини.

3.1. Вимоги до оформлення сторінок.

Кожна сторінка, яка виконує головні функції веб-додатку повинна мати дві форми, перша для заповнення даних, друга для отримання результату функції.

3.1.1. Вимоги до змісту сторінок шифрування.

Форма заповнення має містити наступні поля:

- поле текстової інформації;
- поля алгоритму та режиму шифрування;
- поле ключа шифрування;
- поля початкового та остаточного кодування тексту.

Форма для отримання результату має містити поле зашифрованого тексту та ключа шифрування. Окрім форми має бути присутня кнопка для завантаження JSON файлу з результатом шифрування.

3.1.2. Вимоги до змісту сторінок дешифрування.

Форма заповнення має містити наступні поля:

- поле зашифрованого тексту;
- поля алгоритму та режиму шифрування;
- поле ключа шифрування;
- поля початкового та остаточного кодування тексту.

Також має бути реалізовано функціонал для завантаження JSON файлу, дані якого мають заповнюватися автоматично до форми заповнення.

3.2. Вимоги до оформлення інтерфейсу.

Візуалізація повинна забезпечувати зручний для користувача інтерфейс, який відповідає наступним вимогам:

- наявність локалізованого (україномовного) інтерфейсу користувача;
- меню навігації.

1.3 Аналіз засобів розробки

У глобальній мережі Інтернет існують стандартні засоби для розробки веб-сайтів, одними з цих засобів є мова розмітки HTML і таблиця стилів CSS.

Як мова розмітки HTML, так і таблиця стилів CSS мають різні версії, які поступово розвиваються разом з глобальною мережею Інтернет. Актуальні версії на даний момент, це HTML5 та CSS3. Відрізняються ці версії від попередників своєю простотою і гнучкістю.

Мова розмітки HTML5 тепер має семантичні теги, які спрощують спілкування між веб-сайтом і пошуковою системою. Також в порівнянні з попередньою версією HTML, розробка веб-сайтів не потребує реалізації такої технології, як Flash, яка у свою чергу погано впливала на продуктивність веб-сайтів.

Таблиця стилів третьої версії (CSS3) також відзначилася від попередників. Найбільшою відмінністю є можливість поділу стилів на модулі, що робить розробку набагато ефективнішою. Також були залучені медіа-запити (media queries), які надають можливість змінювати стилі при певному розмірі дисплея пристрою.

Очевидно, що при розробці нового веб-сайту використовуються новітні версії вищевказаних засобів розробки. У будь-якій ситуації

використання новітніх технологій є ефективним способом вирішення будь-яких завдань у сфері розробки в глобальній мережі Інтернет.

Найголовнішим засобом для створення інтерактивних елементів на веб-сайті є всіма улюблена мова програмування JavaScript, заснована і в даний момент підтримується на основі специфікації ECMAScript. JavaScript розширює можливості, які дозволяють створювати веб-сайти з необмеженим функціоналом. Але з огляду на те, що код виконується на стороні клієнта, то все ж існують деякі обмеження, що б не завантажити повністю браузер, який у свою чергу може видати помилку про перевищення доступної пам'яті і слідом припинити роботу.

До певного часу, а точніше до 2015 року, у сфері розробки веб-сайтів реалізація JavaScript-коду найчастіше створювалася за допомогою бібліотеки jQuery. Головною причиною використання даної бібліотеки обґрунтовувалося залученим функціональним стилем програмування, використання якого було набагато простішим за імперативного стилю. Але з приходом нової версії ECMAScript 6 (ES-2015) було повністю реалізовано можливість використання функціонального стилю програмування без використання додаткових засобів розробки.

Також бібліотека jQuery надає готові рішення деяких проблем, які часто зустрічаються при розробці веб-сайтів, наприклад: анімації, створення AJAX-запитів і спрощена навігація по DOM-дереву.

На сьогодні використання jQuery є застарілим засобом розробки. У разі розробки веб-сайту з мінімальним кодом рекомендується використовувати JavaScript без застосування бібліотек для маніпулювання елементами DOM-дерева. В іншому випадку існують новітні фреймворки або бібліотеки за допомогою яких веб-розробка стає ефективнішою.

Одна з таких бібліотек – React, використовується для розробки простих і складних інтерфейсів, впроваджує віртуальне DOM-дерево (Virtual DOM), яке порівнює елементи з поточним DOM-деревом і якщо є відмінності між елементами, відповідно оновлює їх. Таким чином зменшуючи навантаження на браузер користувача.

NodeJS – JavaScript платформа заснована на основі рушію V8. Часто використовується веб-розробниками у зв'язку з простотою і вже знайомими принципами мови програмування JavaScript. Має переваги, такі як запуск сервера на різних операційних системах, велику підтримку з боку спільноти.

Найчастіше в розробці серверних програмних рішень у зв'язці з NodeJS використовується фреймворк Express. Переваги використання даного фреймворку, це легка і зрозуміла організація структури коду, можливість обробки шляхів (routing). Таким чином серверне програмне забезпечення виглядає як API-сервіс, до якого виконуються запити, обробляються і повертають результат до браузера користувача.

PostgreSQL – надійна об'єктно-реляційна система управління базами даних, має величезний вибір між типами даних і функціями. Дає можливість використовувати JSON-дані в таблицях, створювати двомірні масиви та нові типи даних.

1.4 Висновки до розділу

Розглянуто веб-додаток кодування та декодування текстової інформації, як предметну область.

Визначено технічне завдання до розробки веб-додатку, в якому описуються загальні вимоги щодо функціоналу та вимоги до розробки клієнтської та серверної частини.

Проаналізовані та вибрані наступні засоби розробки:

- PostgreSQL гнучка реляційна СКБД, яка застосовує найбільшу кількість функцій для проектування бази даних використовуючи різні підходи проектування, що є оптимальним варіантом для цього проекту;
- платформа NodeJS та фреймворк Express, простий та зрозумілий набір засобів для розробки продуктивних та швидких серверних рішень. Має впроваджену реалізацію криптографічної бібліотеки OpenSSL;
- бібліотека React для розробки інтерфейсів;
- стандартний набір інструментів такий як, мова розмітки HTML5 та таблиця стилів CSS3.

РОЗДІЛ 2. МЕТОДИ ТА ЗАСОБИ ЗАХИСТУ ІНФОРМАЦІЇ

2.1 Криптографічний захист інформації

Криптографія – наука, що вивчає методи приховування інформації, а точніше забезпечення конфіденційності і цілісності даних, а також аутентифікація, контроль і цілісність співзалежних учасників.

Перші записи криптографії були моноалфавітні, являли собою текст змінений способом підстановки інших знаків.

До 1975 року криптографія використовувала шифрувальний метод з секретним ключем, за допомогою якого можна було отримати доступ до розшифровки даних. Пізніше були розроблені методи криптографії з відкритим ключем, які у свою чергу можуть передаватися через відкриті канали зв'язку і використовуватися для перевірки даних.

Криптографія є наукою, яка була заснована за допомогою математики та інформатики. Криптоаналіз дуже схожий до криптографії. Обидві науки тісно пов'язані між собою, однак в останньому випадку вивчаються способи розшифровки прихованої інформації.

Для приховування інформації в криптографії використовується шифрування. Шифрування – це перетворення інформації в форму, яка є не читабельною без ключа шифрування.

Шифрування інформаційних даних відбувається за допомогою зворотніх перетворень, кожне з яких описується ключем і порядком, що визначає черговість їх застосування.

Ключ – є важливим елементом в криптографічного захисту інформації, застосовується як певна послідовність символів для реалізації шифрувальних алгоритмів. Кожне перетворення визначається ключем, який забезпечує безпеку інформації, приклад на рис.2.1. (Процес шифрування з ключем).

Рис.2.1. Процес шифрування з ключем

Види криптографії:

- Симетричне шифрування – надійний та швидкий метод шифрування, використовує один і той же секретний ключ для шифрування і дешифрування інформації.
- Асиметричне шифрування – найнадійніший метод шифрування інформації при обміні даних між користувачами мережі. Використовується відкритий і секретний ключі, які в свою чергу повинні бути пов'язані між собою математичною залежністю. Шифрування інформації проводиться за допомогою відкритого

ключа, в зворотному порядку для дешифрування – секретний ключ.

- Електронний цифровий підпис – надає можливість шифрувати повідомлення за допомогою секретного ключа, в той час як відкритий ключ використовується для розшифровки. Той факт, що повідомлення шифрується секретним ключем власника, служить підтвердженням, що повідомлення належить надходить від справжнього власника секретного ключа.
- Хешування – метод відповідає за перетворення інформації в байти заданого зразка. Хеш-функція виконує перетворення даних, реалізується алгоритмами симетричного шифрування за допомогою зв'язування блоків. Результатом хеш-функції є хеш-код. Хеш-код має унікальну послідовність символів.

Для надання ефективного захисту даних за допомогою шифрування, варто виділити наступні загальноприйняті вимоги:

- зашифровану інформацію можливо прочитати тільки при наявності ключа;
- число операцій, необхідних для розшифровки інформації шляхом перебору ключів повинно виходити за межі можливостей сучасних комп'ютерів;
- знання алгоритму шифрування не повинно впливати на надійність захисту;
- внесення змін до ключа повинні призводити до значної зміни виду зашифрованої інформації, навіть при використанні одного і того ж ключа.

Крім вибору відповідної криптографічної системи шифрування, важливою проблемою залишається – управління ключами. Не в

залежності від складності та надійності криптографічної системи, вона заснована на використанні ключів. Якщо не забезпечено досить надійне управління ключами, то система стає вразливою до злому, що призведе до витоку даних. Управління ключами охоплює генерацію, зберігання і розподіл ключів.

- При генерації ключів не варто використовувати невипадкові ключі з метою легкості їх запам'ятовування. Генерація ключів повинна використовувати високий ступінь випадковості на основі «натуральних» випадкових процесів.
- Зберігання секретних ключів повинно здійснюватися на носіях, які не можуть бути злічені або скопійовані, а також для організації безпеки ключової бази кожен ключ повинен зберігатися в зашифрованому вигляді. Ключі, які зашифровують ключову інформацію називаються майстер-ключами.
- Розподіл ключів це найнебезпечніший етап життєвого циклу ключів. Головною метою при розподілі секретних ключів – це запобігти потраплянню ключів до сторонніх осіб. Для відкритих ключів головне забезпечити автентичність та цілісність.

2.2 Аналіз симетричного шифрування

Головною перевагою симетричного шифрування це здатність обробляти величезну кількість даних за короткий проміжок часу. Тому симетричні алгоритми часто використовуються для шифрування великих масивів даних. Симетричне шифрування, як засіб захисту даних, може виявитися вельми витратним процесом через складність передачі секретного ключа.

Основні способи симетричного шифрування:

- перестановка – символи початкового тексту переставляються відповідно заданих правил;
- підстановка – символи відкритого тексту замінюються символами того ж або іншого алфавіту в залежності від ключа шифрування;
- аналітичне перетворення – початковий текст перетворюється за заданим аналітичним правилом. Наприклад, гамування - перетворює початковий текст за допомогою деякої псевдовипадкової послідовності, яка генерується на основі ключа;
- комбіноване перетворення – застосовується до кожного блоку шифрувального тексту, являє собою послідовність головних методів перетворення.

Кожен симетричний алгоритм шифрування має симетричний шифр. Симетричний шифр – це шифр в якому використовується один і той же ключ для шифрування і дешифрування даних. Симетричні шифри діляться на блокові й потокові.

Блокові шифри – обробляють дані блоками з фіксованою довжиною (зазвичай 64, 128 біт).

Потокові шифри – шифр виконує шифрування над кожним бітом або байтом відкритого тексту використовуючи гамування.

Найбільш важливою характеристикою блокового шифру є довжина ключа. Надійність шифру безпосередньо залежить від довжини ключа, чим довше ключ, тим надійніше шифр. Якщо довжина ключа n біт, то кількість можливих ключів 2n. Для знаходження правильного ключа необхідно перебрати 2n-1 ключів.

Блокові шифри: AES, ГОСТ 28147-89, DES, 3DES, Camellia, RC2, RC5, Blowfish, Twofish, NUSH, IDEA, CAST.

Потокові шифри: RC4, SEAL, WAKE.

На даний момент найбільш надійний симетричний алгоритм шифрування – це AES (Advanced Encryption Standard) має 128 бітний розмір блоку, довжина ключа може бути 128, 192 або 256 біт. Прийнятий як стандарт шифрування за урядом США. Заснований на алгоритмі Rijndael. AES має досить таки простий математичний опис, що звичайно ж стурбувало багатьох фахівців в області захисту даних. За словами фахівців їх турбувала проста алгебраїчна структура алгоритму, жоден інший на той час блоковий шифр не мав такого простого алгебраїчного уявлення.

DES (Data Encryption Standard) – перший симетричний шифр з використанням ключа довжиною 56 біт. Розмір блоку дорівнює 64 бітам. DES схвалено урядом США і залучено для передачі даних через мережу Інтернет. В даний час прямим розвитком DES є алгоритм Triple DES (3DES), який реалізовує шифрування і дешифрування шляхом триразового виконання алгоритму DES.

RC2 – блоковий нестабільний 64 бітний шифр, що допускає довжину ключа від 8 до 1024 біт. Як і більшість алгоритмів, є мережею Фейстеля. Використовувати даний шифр не рекомендується, у зв'язку з тим що алгоритм в повному обсязі проаналізовано і може мати серйозні уразливості безпеки які не були виявлені раніше.

RC4 – єдина версія алгоритму з реалізацією потокового шифру, довжина ключа може становити від 40 до 2048 біт. Широко використовується в різних системах захисту інформації (SSL, TLS). Алгоритм RC4 будується на основі генератора псевдовипадкових бітів, на вході записується ключа, а на виході зчитуються псевдовипадкові біти.

RC5 – блоковий шифр мережі Фейстеля дозволяє варіювати розмір блоку між 32, 64 і 128 бітами. Довжина ключа варіюється від 0 до 2040 бітів.

RC6 – блоковий шифр мережі Фейстеля розроблений на основі RC5, були внесені різні зміни, що б допустити алгоритм до AES конкурсу. А саме фіксований 128 бітний розмір блоку і підтримку довжини ключів в 128, 192 і 256 біт.

2.3 Аналіз режимів шифрування

Кожен блоковий шифр має кілька режимів шифрування. Режим шифрування використовується для перетворення послідовності блоків відкритих даних в послідовність блоків зашифрованих даних. Для шифрування одного блоку можуть використовуватися дані іншого блоку.

На сьогодні актуальними режимами шифрування є: ECB, CBC, CFB, OFB та CTR.

ECB (Electronic Codebook) – режим електронної кодової книги або режим простої заміни. При використанні даного режиму кожен блок відкритого тексту замінюється шифротекстом. Головною перевагою даного режиму це швидкість обробки блоків, і можливість паралельного шифрування. Недоліком режиму є те, що результат шифрування однакових блоків одним і тим же ключем збігаються.

Режим ECB варто використовувати для безпечної передачі окремих значень (наприклад, ключа). Ще його зручно застосовувати, коли потрібен випадковий доступ до бази даних.

CBC (Cipher Block Chaining) – режим зчеплення блоків шифротексту з використанням механізму зворотного зв'язку. Складає

кожен блок відкритого тексту (крім першого) побитно з результатом попереднього шифрування за допомогою оператора XOR.

У режимі CBC останній блок шифротексту на виході залежить від ключа і вектора ініціалізації, та від кожного біту відкритого тексту. Отже, останній блок шифротекста можна використовувати як ідентифікатор, який приховує інформацію про вміст повідомлення. MAC (Message Authentication Code) – ідентифікатор, часто використовується для аутентифікації повідомлень і відправників.

CFB (Cipher Feedback) – режим зворотного зв'язку по шифротексту, дуже схожий на режим «CBC», трансформує блоковий шифр в потоковий шифр, що само синхронізується. Якщо хоч найменшу частину шифротексту втрачено, шифр видаватиме неправильний відкритий текст доки не буде досягнуто стану, який був під час шифрування.

Режим CFB рекомендується використовувати для передачі по каналах, які є зашумленими, оскільки в даному режимі не накопичуються помилки.

OFB (Output Feedback) – режим зворотного зв'язку по виходу, перетворює блоковий шифр в синхронний шифр потоку. Щоб отримати шифротекст, даний режим генерує ключові блоки, які є результатом складання з блоками відкритого тексту. Обернення біту в шифротексті виробляє обернений біт у відкритому тексті в тому ж самому місці розташування.

CTR (Counter Mode) – режим лічильника, подібно режиму OFB, перетворює блоковий шифр в потоковий. Шифрування відкритого тексту проводиться за допомогою вектора ініціалізації та деякої константи, в результаті отриманий шифротекст і відкритий текст складається за

допомогою оператора XOR. Завдяки паралельному виконанню є дуже швидким режимом шифрування.

2.4 Асиметричне шифрування та алгоритм шифрування RSA

Головною проблемою симетричних криптографічних систем є розподілення ключів. Для того що б реалізувати конфіденційний обмін інформацією між двома адресатами, ключ повинен бути згенерований одним з них, а потім якимось чином знову ж у конфіденційному порядку переданий іншому. Для вирішення цієї проблеми існують криптосистеми з відкритим ключем. Суть цих систем полягає в тому, що кожному адресату генерується два ключі, відкритий і секретний, пов'язані між собою математичною залежністю. Відкритий ключ може бути доступний кожному, хто бажає передати повідомлення адресату. Секретний ключ у свою чергу зберігається в таємниці. Вихідний текст шифрується відкритим ключем і передається. Дешифрування даного повідомлення можливо тільки з використанням секретного ключа, який відомий тільки адресату, рис.2.2. (Життєвий цикл асиметричного шифрування).

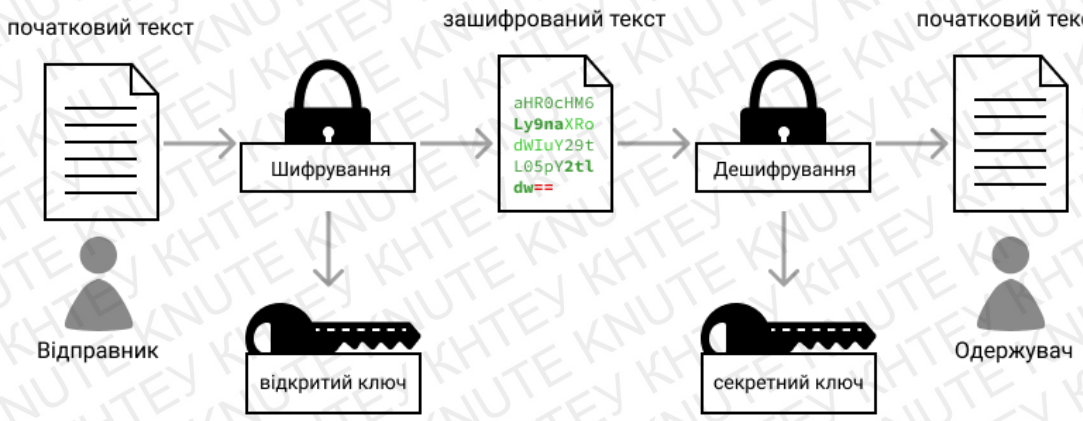

Рис.2.2 Життєвий цикл асиметричного шифрування

RSA – поширений криптографічний алгоритм з відкритим ключем. Виконує шифрування даних за допомогою обчислювальної складності задачі факторизації великих цілих чисел.

Протоколи, такі як SSH, MIME, SSL/TLS, використовують алгоритм RSA для шифрування даних. Також використовується в браузерах для встановлення безпечного з'єднання до незахищеної мережі або валідації цифрового підпису.

Алгоритм RSA працює наступним чином:

1. Генерується два великих простих числа p та q;
2. Обчислюється добуток за формулою (2.1);

$$n = p \times q \tag{2.1}$$

3. Обчислюється функція Ейлера $f(n)$
4. Вибирається відкритий ключ – e, як число яке задовольняє умові за формулою (2.2) з результатом функції Ейлера;

$$0 < e < n \tag{2.2}$$

5. Обчислюється закритий ключ – d, як зворотне число до e по модулю $f(n)$, зі співвідношенням формули (2.3);

$$d \times e \equiv 1 \tag{2.3}$$

6. Публікується відкритий ключ (e, n) в спеціальному сховищі.

Стійкість даного алгоритму залежить від довжини ключа. В даний час лабораторія RSA рекомендує використовувати ключі розміром не менше 2048 бітів.

Кпиптографічна система PGP – дозволяє користувачам анонімно обмінюватися інформацією в електронному вигляді. Має багато програмних реалізацій завдяки стандарту OpenPGP (RFC 4880).

В процесі обміну інформації виконуються операції шифрування і цифрового підпису повідомлень, файлів та іншої інформації.

Шифрування працює за тим же принципом використовуючи два ключі. Усього доступно три типи ключів:

- RSA;
- RSA Legacy;
- GnuPG.

При процесі шифрування програма PGP виконує стиснення тексту для ускладнення криптоаналізу. На самому початку генерується число - сесійний ключ, який в підсумку шифрується відкритим ключем. Для дешифрування повідомлень використовується закритий ключ, в процесі дешифрування з якого спочатку витягується сесійний ключ, а потім вже дешифрує повідомлення.

Програма PGP є гібридною криптосистемою, оскільки використовує симетричне шифрування і реалізовує функціонал відкритого ключа. Таким чином надає досить таки швидкий процес шифрування і зручне керування ключами.

2.5 Висновки до розділу

Розглянуто наступні засоби криптографічного захисту даних:

- симетричні та асиметричні алгоритми шифрування;

- цифровий підпис;
- хешування.

Розглянуто процес шифрування за допомогою симетричних алгоритмів використовуючи секретний ключ та вектор ініціалізації. Проаналізувавши алгоритми шифрування, рекомендується застосовувати симетричний алгоритм шифрування AES, що є таким, який прийнятий урядом США, як стандарт шифрування.

Ґрунтуючись на результатах аналізу режимів шифрування, було вирішено, що варто використовувати:

- режим зчеплення блоків шифротексту (CBC);
- та режим лічильника (CTR), який є найефективнішим на даний час.

Проаналізовано асиметричний алгоритм шифрування RSA, який використовує відкритий та секретний ключі. Розглянуто математичний процес застосування алгоритму шифрування. Щодо рекомендацій використання алгоритму RSA, можна визначити, що розмір ключа повинен складати не менше ніж 2048 бітів.

Проаналізовано криптографічну систему PGP, як систему для обміну інформації серед користувачів. Розглянуто процес виконання програми та переваги використування.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Проектування бази даних

В даному проекті таблиці БД потрібні тільки для зчитування, оскільки записувати будь-які дані при шифруванні інформації користувача на базу даних є порушенням конфіденційності.

Для моделювання таблиць БД використовується програмне забезпечення pgModeler - PostgreSQL Database Modeler (демо-версія).

За умовами розробки були спроектовані дві таблиці бази даних для зберігання списку алгоритмів та режимів шифрування, рис.3.1. (Логічна модель БД). Дві таблиці мають зв'язок багато-до-баготьох, таким чином утворючи третю таблицю, рис.3.2. (Фізична модель БД).

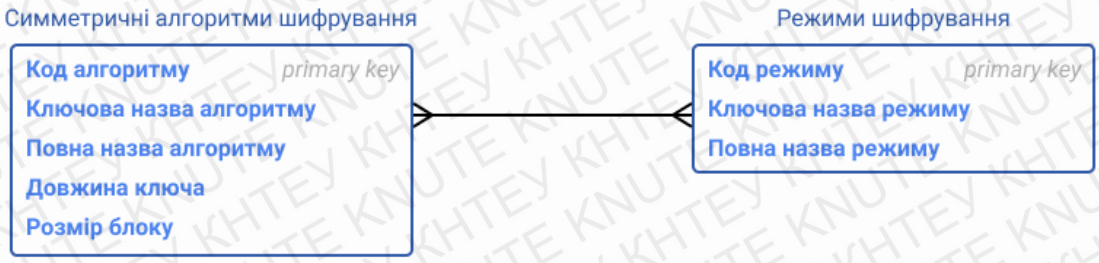

Рис.3.1. Логічна модель БД

Таблиця симетричних алгоритмів шифрування, яка містить наступні поля:

- «id» – тип даних serial, містить унікальний номер запису в таблиці;
- «name» – тип даних varchar, містить ключову назву алгоритму, названу раніше в бібліотеці OpenSSL;

- «title» – тип даних varchar, також містить назву алгоритму, але в зрозумілій формі для застосування до інтерфейсу;
- «keysize» – тип даних smallint, містить довжину ключа алгоритму шифрування;
- «blocksize» – тип даних smallint, містить розмір блоку алгоритму шифрування.

Таблиця режимів шифрування, має зв'язок багато-до-багатьох з таблицею «Симетричні алгоритми шифрування»:

- «id» – тип даних serial;
- «name» – тип даних varchar, містить назву режиму шифрування;
- «title» – тип даних varchar, містить повну назву режиму шифрування.

Таблиця симетричних алгоритмів шифрування з доступними режимами шифрування, яка є згенерованої таблицею зв'язку багато-до-багатьох таблиць «Симетричні алгоритми шифрування» та «Режими шифрування»:

- «algorithm_id» – тип даних integer, є зовнішнім ключем таблиці «Симетричних алгоритмів шифрування»;
- «mode_id» – тип даних integer, є зовнішнім ключем таблиці «Режимів шифрування».

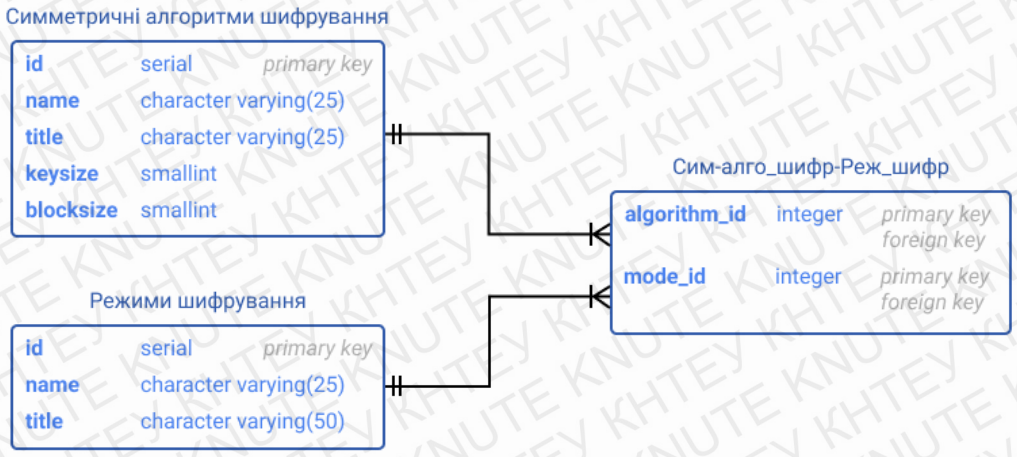

Рис.3.2. Фізична модель БД

3.2 Розробка серверної частини

Розробка серверної частини складається з проектування архітектури й написання програмного коду. Серверне забезпечення відгукується на запити від браузера, приймає, обробляє і повертає дані за допомогою архітектурного стилю REST.

Для розробки серверного забезпечення використовується архітектура Model-Service-Controller-Router, рис.3.1 (Структура серверного забезпечення), яка надає такі переваги, як розподіл логіки та ефективне розширення функціоналу.

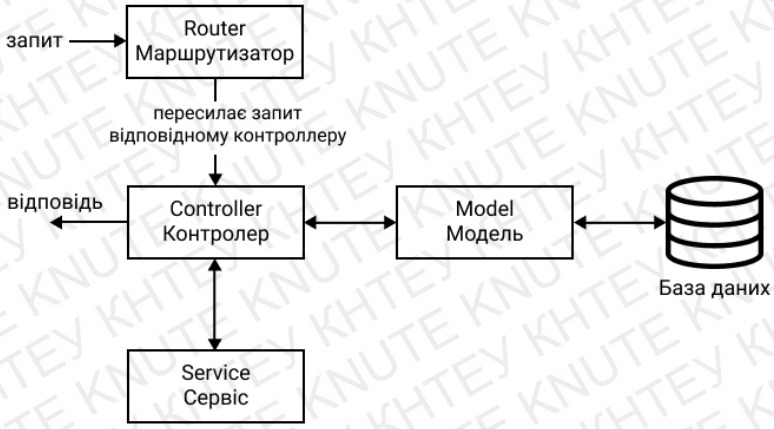

Рис.3.3. Структура серверного забезпечення

- маршрутизатор (Router) обробляє запити та викликає відповідну функцію контролер (Controller);
- контролер – функція для отримання потрібних даних із сервісів та моделей, після чого відправляється результат до браузера;
- сервіс – функція, яка виконує деякі логічні операції, такі як шифрування і дешифрування;
- модель виконує запити до певних таблиць бази даних.

На основі архітектури серверного забезпечення була створена структура директорій, рис.3.4. (Структура директорій серверного забезпечення), за допомогою якої простіше орієнтуватися під час розробки.

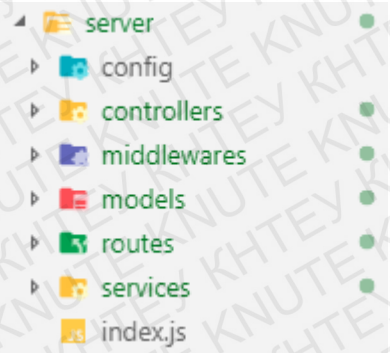

Рис.3.4. Структура директорій серверного забезпечення

Розробка починається з ініціалізації пакетного менеджера npm і системи контролю версій git. За допомогою пакетного менеджера встановлюються всі потрібні засоби розробки.

Головний файл ініціалізації index.js серверного забезпечення складається з налаштувань сервера, підключення бібліотек і модулів (див. додаток А).

Ґрунтуючись на вимогах до проекту створені наступні шляхи маршрутизатору (див. додаток Б):

- /encrypt – симетричне шифрування;

- /decrypt – симетричне дешифрування;
- /rsa-encrypt – асиметричне шифрування RSA;
- /rsa-decrypt – асиметричне дешифрування RSA;
- /get-data – отримання доступних алгоритмів шифрування і кодування.

Як було зазначено раніше кожен маршрутизатор повинен містити контролер, який у свою чергу залучає сервіси та моделі. Тому, створено наступні три функції контролера:

- SymmetricController – обробляє запити для симетричного шифрування та дешифрування;
- RSAController – обробляє запити для асиметричного шифрування та дешифрування;
- DataController – обробляє запити на отримання даних з функцій моделей.

Контролер у свою чергу викликає сервіс, який виконує деякі логічні операції. Для симетричного шифрування створено сервіс SymmetricService, рис.3.5. (Код функцій encrypt та decrypt сервісу symmetricService), а для асиметричного RSAService на рис.3.6. (Код функцій encrypt та decrypt сервісу RSAService).

```
const encrypt = ({
  password,
  encodingFrom = 'utf8',
  encodingTo = 'base64',
}) => {
  const privateKey = getKey(password);
  const iVector = getIV(ALGORITHM.IV_LEN);

  const cipher = crypto
    .createCipheriv(ALGORITHM.CIPHER, privateKey, iVector);
  let crypted = cipher.update(text, encodingFrom);
  crypted = Buffer.concat([iVector, crypted, cipher.final()]);

  return {
    text: crypted.toString(encodingTo),
    key: privateKey.toString(encodingTo),
    algorithm,
  };
};
```

```
const decrypt = ({ privateKey, decodingFrom, decodingTo }) => {
  const bufferedText = Buffer.from(text, decodingFrom);
  const cipherText = bufferedText.slice(ALGORITHM.IV_LEN);
  const iVector = bufferedText.slice(0, ALGORITHM.IV_LEN);
  const key = Buffer.from(privateKey, decodingFrom);

  const decipher = crypto
    .createDecipheriv(ALGORITHM.CIPHER, key, iVector);
  let decrypted = decipher.update(cipherText);
  decrypted = Buffer.concat([decrypted, decipher.final()]);

  return {
    text: decrypted.toString(decodingTo),
  };
};

return {
  encrypt,
  decrypt,
};
```

Рис.3.5. Код функцій encrypt та decrypt сервісу symmetricService

На самому початку виконання функції encrypt оголошується змінна ALGORITHM, яка містить інформацію про алгоритм, а саме: повна назва з режимом шифрування, довжина ключа, вектора ініціалізації й солі.

Після чого генерується вектор ініціалізації та ключ за допомогою випадкових байтів.

Наступним етапом є створення шифру використовуючи ключ і вектор ініціалізації за допомогою методу createCipheriv (бібліотека crypto). Використовуючи створений шифр виконується функція шифрування, яка приймає початковий текст і початковий алгоритм кодування. Завершується виконання шифрування об'єднанням вектора ініціалізації й зашифрованого тексту.

Функція дешифрування мають схожу структуру виконання, головною відмінністю є відділення вектора ініціалізації з шифротексту і виконання методу createDecipheriv.

```
31  const encrypt = ({
32    publicKey = null,
33    encodingFrom = 'utf8',
34    encodingTo = 'base64',
35  }) => {
36    const keys = generateKeys(publicKey);
37
38    const toEncrypt = Buffer.from(text, encodingFrom);
39    const encrypted = crypto
40      .publicEncrypt(keys.publicKey, toEncrypt)
41      .toString(encodingTo);
42
43    return {
44      text: encrypted,
45      publicKey: keys.publicKey,
46      privateKey: keys.privateKey,
47    };
48  };
```

```
49  const decrypt = ({
50    privateKey,
51    decodingFrom = 'base64',
52    decodingTo = 'utf8',
53  }) => {
54    const toDecrypt = Buffer.from(text, decodingFrom);
55    const decrypted = crypto
56      .privateDecrypt(privateKey, toDecrypt)
57      .toString(decodingTo);
58
59    return {
60      text: decrypted,
61    };
62  };
63  return {
64    encrypt,
65    decrypt,
66  };
67  }
```

Рис.3.6. Код функцій encrypt та decrypt сервісу RSAService

Функція асиметричного шифрування RSA-алгоритму, починає виконання з генерації пари ключів довжиною 2048 бітів, зберігаючи їх у форматі pem. Потім створюється змінна toEncrypt, до якої присвоюється початковий текст у форматі буфера (Buffer). За допомогою методу

publicEncrypt виконується шифрування з раніше згенерованим відкритим ключем.

Функція дешифрування потребує секретний ключ шифрування, який передається аргументом до методу privateDecrypt.

Як було зазначено, моделі використовуються для створення запитів до бази даних. Оскільки все ще потрібно зчитувати з бази даних, це дві таблиці зі звязком багато-до-багатьох, які містять список доступних симетричних алгоритмів та режими шифрування, було створено модель SymmetricModel. Дана модель містить функцію, яка створює запит на отримання даних з двух таблиць, рис.3.7. (Код моделі AlgorithmModel).

```
import { query } from './index';

const getAlgorithmList = async () => {
  const sql = `select algo.id, algo.name, algo.title, algo.keysize,
    algo.blocksize, json_agg(json_build_object(
      'id', modes.id,
      'name', modes.name,
      'title', modes.title
    )) as modes
    from encrypt_algorithms as algo
    inner join algorithm_mode on algo.id = algorithm_mode.algorithm_id
    inner join modes on algorithm_mode.mode_id = modes.id
    group by algo.id, algo.name, algo.title, algo.keysize, algo.blocksize
    order by algo.name`;
  const result = await query(sql);
  return result.rows;
};

const Algorithm = {
  getAlgorithmList,
};

export default Algorithm;
```

Рис.3.7. Код моделі AlgorithmModel

Функція query є допоміжною функцією для створення запиту використовуючи клієнт pg на рис.3.8. (Код допоміжної функції query).

```
import { Pool } from 'pg';
import { dbConfig } from '../config/config';
const pool = new Pool({
  user: dbConfig.user,
  host: 'localhost',
  database: 'encode',
  password: dbConfig.password,
  port: 5432,
});
export const query = async sql => {
  const client = await pool.connect();
  let result;
  try {
    result = await client.query(sql);
  } catch (error) {
    throw error;
  }
  client.release();
  return result;
};
```

Рис.3.8. Код допоміжної функції query

3.3 Розробка клієнтської частини

Розробка починається з ініціалізації бібліотеки React та підключення її до головної сторінки index.html (див. додаток В). Таким чином розробка сторінок та стилів застосовується безпосередньо в файлах JavaScript.

Ґрунтуючись на розробленому серверному забезпеченні були створені наступні сторінки:

- сторінки симетричного шифрування і дешифрування;
- сторінки RSA-шифрування і дешифрування;
- сторінка з описом веб-додатку.

Для розробки клієнтської частини використовуючи бібліотеку React була спроектована архітектура (рис).

Для поділу логіки та інтерфейсу використовується методика поділу компонентів на:

- компоненти-уявлення – які містять тільки структуру та дизайн компонента, а саме JSX-структуру (HTML-in-JS) і CSS-стилі;

- компоненти-контейнери – містять в собі компоненти-уявлення і будь-які логічні взаємодії з даними, наприклад, створення запитів, обробка отриманих даних з сервера та інше.

Сторінки шифрування і дешифрування містять компоненти-контейнери, які містять форму заповнення і форму результату, рис.3.9. (Сторінка симетричного шифрування).

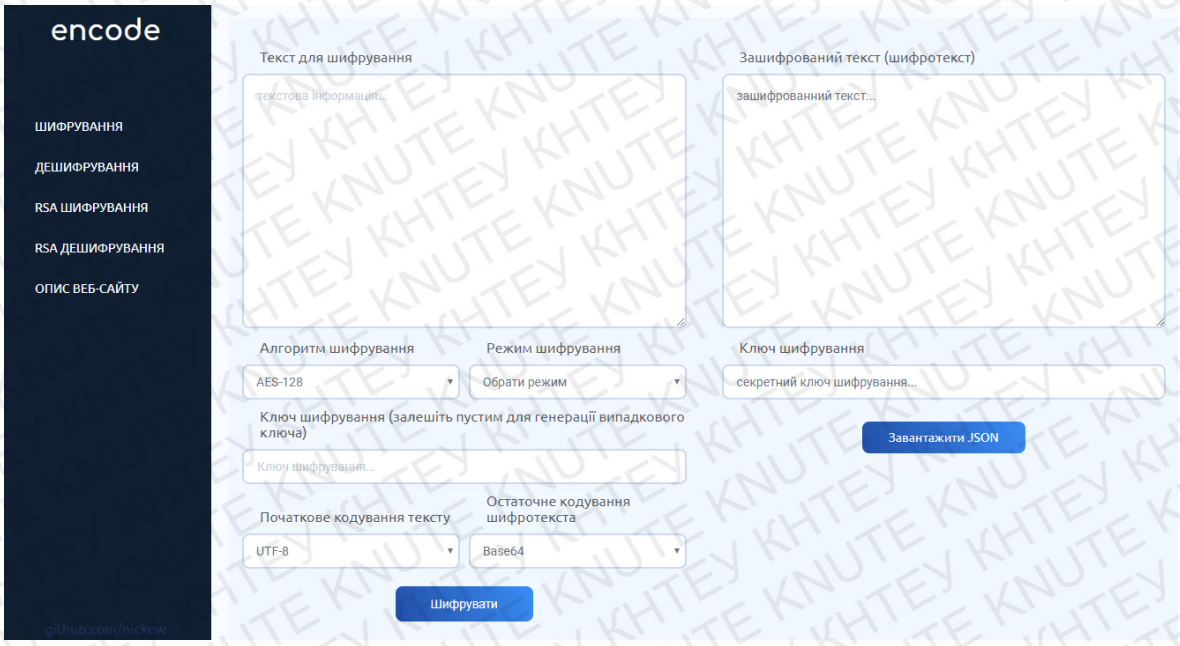

Рис.3.9. Сторінка симетричного шифрування

Після посилання форми, викликається допоміжна функція для створення запиту за допомогою технології fetch. В результаті отримані дані зберігаються в глобальному стані веб-додатку. Таким чином, форма результату при оновленні глобального стану, отримує дані й заповнює ними форму для відображення результату.

Також надана можливість зберегти зашифровані дані в форматі JSON, щоб пізніше можна було дешифрувати інформацію завантаживши JSON файл, рис.3.10. (Сторінка симетричного дешифрування).

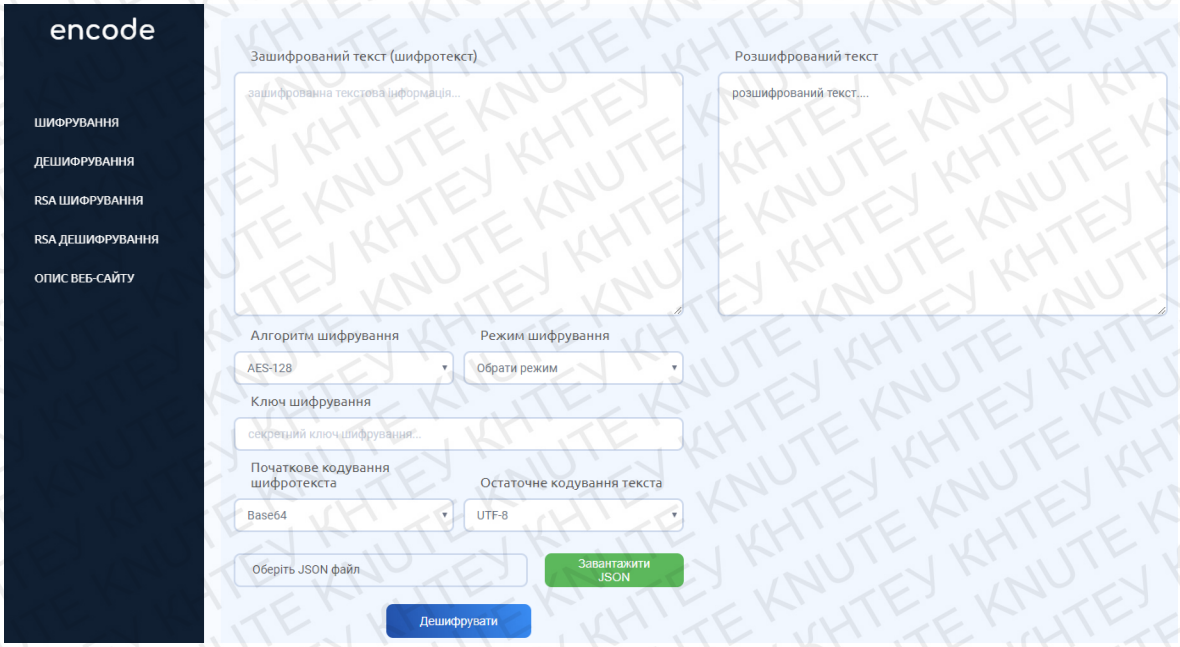

Рис.3.10. Сторінка симетричного дешифрування

Сторінки RSA шифрування і дешифрування створені за тим же принципом, що і сторінки для симетричного шифрування.

Перехід між сторінками залучений за допомогою бібліотеки react-router, яка надає можливість переходу між сторінками оновлюючи тільки ті, компоненти які цього потребують. Таким чином компонент відповідає за ліву панель не оновлюється при переході між сторінками.

3.4 Висновки до розділу

В даному розділі було спроектовано і розроблено клієнтську і серверну частину веб-додатку і структуру бази даних за допомогою сучасних засобів розробки, таких як: React, NodeJS, Express і PostgreSQL.

Браузер і сервер спілкуються за принципом архітектурного стилю REST. Браузер створює запит, сервер обробляє цей запит за допомогою обробників шляхів, які у свою чергу викликають функції-сервіси або функції-моделі в залежності від запиту, після чого повертають результат

у форматі JSON назад в браузер користувача, і вже на стороні клієнта ці дані обробляються і додаються в глобальний стан веб-додатку.

Таблиці бази даних спроектовані за стандартними принципами реляційної бази даних. Створені таблиці використовуються в основному тільки для читання інформації алгоритмів шифрування і кодування.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

В ході виконання випускної кваліфікаційної роботи було розглянуто криптографічні засоби захисту інформації в глобальній мережі Інтернет. Було проаналізовано симетричні алгоритми шифруванн, режими шифрування, асиметричний алгоритм RSA та криптографічна система PGP.

Спроектовано та розроблено структуру веб-додатку дотримуючись принципів простого і зрозумілого інтерфейсу. Клієнтську частину розроблено, як односторінковий додаток за допомогою бібліотеки для розробки інтерфейсів React.

Серверне забезпечення розроблено за архітектурою Model-Service-Controller-Router, за допомогою якої поділяється логіка на чотири окремі рівня, які не залежать один від одного.

За допомогою бібліотеки crypto, яка є реалізацією бібліотеки OpenSSL, розроблено функціонал шифрування та дешифрування текстової інформації. Наступні симетричні алгоритми шифрування були включені до вибору: AES, DES, RC2, Camellia, Cast5, Seed, Blowfish, та алгоритм асиметричного шифрування – RSA.

У подальшому розвитку проекту планується додати більше алгоритмів шифрування, розробити мобільну версію застосовуючи React Native та застосувати методику серверного рендерингу (Server Side Rendering) за допомогою бібліотеки next.js.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Видання іноземною мовою:

1. Simpson K. You Don't Know JS: Async & Performance / Kyle Simpson. – O'Reilly Media, Inc, 2015 – 247 p.
2. Paar C. Understanding Cryptography / Christof Paar, Jan Pelzl. – Springer, 2011 – 327 p.
3. Banks. A. Learning React. Functional Web Development with React and Redux / Alex Banks, Eve Porcello. – O'Reilly Media, 2017 – 350 p.

Підручники:

4. Панасенко С.П. Алгоритмы шифрования. Специальный справочник / С.П. Панасенко. – СПб.: БХВ-Петербург, 2009. – 576 с.

Електронні ресурси:

5. Документація бібліотеки для розробки інтерфейсів React [Електронний ресурс]. – [Режим доступу]: https://reactjs.org/docs.
6. Документація бібліотеки crypto [Електронний ресурс]. – [Режим доступу]: https://nodejs.org/api/crypto.html.
7. Документація фреймворку Express [Електронний ресурс]. – [Режим доступу]: https://expressjs.com/ru/api.html.

ДОДАТКИ

Ініціалізація серверу ДОДАТОК А

Код файлу ініціалізації сервера index.js

```
import express from 'express';
import cookieParser from 'cookie-parser';
import { resolve } from 'path';
import cors from 'cors';
import bodyParser from 'body-parser';
import setup from './middlewares/setupMiddleware';
import { router as routes } from './routes';
const app = express();
const port = 3000;
app.use(bodyParser.json());
app.use(
  bodyParser.urlencoded({ extended: true })
);
app.use(cookieParser('secret_word'));
app.use(cors({ origin: 'http://localhost:3000', credentials: true }));
app.use(routes);
setup(app, {
  outputPath: resolve(process.cwd(), 'build'),
  publicPath: '/',
});
app.listen(port, err => {
  if (err) {
    console.error(err);
  } else {
    console.log(`server running on localhost:${port}`);
  }
});
```

ДОДАТОК Б

Код маршрутизатора (router) apiRoute:

```
import express from 'express';
import SymmetricController from '../controllers/SymmetricController';
import RSAController from '../controllers/RSAController';
import DataController from '../controllers/DataController';

const router = express.Router();

const ENCRYPT_URL = '/encrypt';
const DECRYPT_URL = '/decrypt';
const RSA_ENCRYPT_URL = '/rsa-encrypt';
const RSA_DECRYPT_URL = '/rsa-decrypt';
const ALGO_ENCO_LIST_URL = '/get-data';

router.post(ENCRYPT_URL, SymmetricController.encrypt);
router.post(DECRYPT_URL, SymmetricController.decrypt);
router.post(RSA_ENCRYPT_URL, RSAController.encrypt);
router.post(RSA_DECRYPT_URL, RSAController.decrypt);
router.get(ALGO_ENCO_LIST_URL, DataController.getData);

export { router as ApiRoute };
```

ДОДАТОК В

HTML-розмітка index.html:

```
<!DOCTYPE html>
<html lang="ua">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta http-equiv="X-UA-Compatible" content="ie=edge" />
  <title>Encode - шифрування та дешифрування текстової інформації</title>
  <link href=https://fonts.googleapis.com/css?family=Roboto:400,500,700 rel="stylesheet" />
 </head>
 <body>
  <div id="app"></div>
 </body>
</html>
```

Код файлу ініціалізації клієнтської частини index.jsx:

```
import '@babel/polyfill';
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter as Router } from 'react-router-dom';
import App from './containers/App';
import GlobalStore from './store/GlobalStore';
ReactDOM.render(<GlobalStore>
  <Router>
   <App />
  </Router>
 </GlobalStore>,
 MOUNT_NODE);
```