МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА
Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці

ОСВІТНЬО ПРОФЕСІЙНА ПРОГРАМА «КОМП'ЮТЕРНІ НАУКИ»
галузь знань 12 «Інформаційні технології»
спеціальність 122 «Комп'ютерні науки»

Форма навчання: денна

КВАЛІФІКАЦІЙНИЙ БАКАЛАВРСЬКИЙ ПРОЄКТ

на тему

«ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ ПЕРЕДАЧІ ЗАШИФРОВАНИХ ПОВІДОМЛЕНЬ»

здобувача Урденка Дениса Олександровича _____

Науковий керівник:
д. е. н., професор
_____ Мозгаллі О.П.

Робота допущена до захисту перед екзаменаційною комісією з атестації здобувачів вищої освіти
завідувач кафедри:
к.е.н., доцент.
_____ Тішков Б.О.

Київ 2024

Міністерство освіти і науки України
Київський національний економічний університет імені Вадима Гетьмана
Навчально-науковий інститут «Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА «КОМП'ЮТЕРНІ НАУКИ»
галузь знань 12 «Інформаційні технології»
спеціальність 122 «Комп'ютерні науки»

ПОГОДЖЕНО:
Керівник проектної групи(гарант)
освітньо-професійної програми
_____Іванченко Г.Ф.
"____" _____2024 р.

ЗАТВЕРДЖУЮ:
Завідувач кафедри
_____Тішков Б.О.
"____" _____2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувача вищої освіти ***Урденка Дениса Олександровича***

очної (денної) форми навчання

на підготовку кваліфікаційного бакалаврського проєкту
на тему: *«Інформаційна система для передачі зашифрованих повідомлень»*

Тему затверджено наказом ректора Університету від «11» березня 2024 р. № 529-ст.

Кваліфікаційний бакалаврський проєкт виконується на матеріалах
відкритих інтернет-джерел та інших публікацій

План кваліфікаційного бакалаврського проєкту

Розділ І Характеристика та аналіз процесів ІТ-підтримки передавання зашифрованих повідомлень

Розділ ІІ Розробка вимог і моделювання інформаційної системи для передавання зашифрованих повідомлень

Розділ ІІІ Проектування та реалізація компонентів системи для передавання зашифрованих повідомлень

Об'єкт дослідження інформаційна система для передачі зашифрованих повідомлень

Предмет дослідження інформаційні системи та методи шифрування даних

Мета кваліфікаційного бакалаврського проєкту розробити інформаційну систему для передачі зашифрованих повідомлень

Конкретні завдання, які студент повинен виконати для досягнення поставленої мети:

У розділі I *Дослідити принципи роботи та характеристики симетричних та асиметричних алгоритмів шифрування, порівняти різні методи шифрування з точки зору їх стійкості, швидкості та складності реалізації. Обрати оптимальний метод шифрування для розроблення інформаційної системи.*

У розділі II *Визначити та описати вимоги до інформаційної системи для передавання зашифрованих повідомлень. Проаналізувати та навести специфікацію вимог до інформаційної системи для передавання зашифрованих повідомлень. Розробити постановку задачі, архітектуру інформаційної системи для передачі зашифрованих повідомлень. Визначити функціональні модулі системи, такі як модуль шифрування/дешифрування, модуль аутентифікації, модуль керування ключами тощо. Розробити схеми взаємодії між модулями системи.*

У розділі III *Здійснити проектування та реалізувати компоненти інформаційної системи для передавання зашифрованих повідомлень. Вибрати мову програмування та інструменти розробки для реалізації системи. Розробити програмне забезпечення для кожного модуля системи. Навести опис інформаційного та технічного забезпечення для реалізації інформаційної системи. Протестувати систему щодо наявності помилок та уразливостей.*

Завдання підготував
науковий керівник _____ **Мозгаллі Ольга Петрівна**

“11” березня 2024 р.

Завдання одержав
студент _____ **Урденко Денис Олександрович**

“11” березня 2024 р.

Відгук
про кваліфікаційний бакалаврський проєкт
здобувача навчально-наукового інституту
«Інститут інформаційних технологій в економіці»
освітньо-професійної програми
«Комп'ютерні науки»

Урденка Дениса Олександровича

на тему
«Інформаційна система для передачі зашифрованих повідомлень»

1. Актуальність теми: У сучасному світі інформаційні системи (ІС) та дані постійно піддаються атакам із боку зловмисників, тому захист інформації стає все більш важливим завданням, що робить тему розробки безпечних систем передачі даних своєчасним. В свою чергу, поява нових алгоритмів шифрування та методів криптографії робить можливим створення більш стійких та ефективних систем передачі зашифрованих повідомлень, а потенційно широке застосування ІС для передачі зашифрованих повідомлень в різних соціально-економічних сферах, зумовлює високу актуальність обраної теми випускного бакалаврського проєкту.

2. Позитивні риси кваліфікаційного бакалаврського проєкту: В роботі було системно досліджено предметну область, включаючи принципи шифрування, архітектуру систем, методи захисту від атак та економічні аспекти. Автором було застосовано актуальні методи аналізу та синтезу інформації, а також програмного забезпечення для розроблення та тестування інформаційної системи для передавання зашифрованих повідомлень. Розроблена інформаційна система для передавання зашифрованих повідомлень може мати практичне застосування у різних сферах діяльності.

3. Наявність самостійних розробок автора: Кваліфікаційний бакалаврський проєкт виконаний автором самостійно із застосуванням сучасного інструментарію. Автор самостійно дослідив методи шифрування та розробив архітектуру інформаційної системи; також ним було самостійно обрано програмні та технічні засоби й розроблено програмне забезпечення для реалізації інформаційної системи для передавання зашифрованих повідомлень. Крім того, автором було проведено первинне тестування ІС та усунено виявлені помилки.

4. Цінність теоретичних висновків та практичних рекомендацій: Теоретичні висновки роботи можуть бути використані для розробки нових методів шифрування та інформаційних систем. Практичні рекомендації можуть бути використані для впровадження розробленої інформаційної системи передавання зашифрованих повідомлень на підприємствах та в організаціях, що потребують застосування такого роду інформаційних систем.

5. Наявність недоліків: Представлене проектне рішення потребує подальшого вдосконалення й розвитку для використання у промисловій експлуатації. Також перед впровадженням в промислову експлуатації необхідно буде провести тестування розробленої ІС на проникнення, щоб оцінити її стійкість до можливих кібератак.

6. Загальна оцінка кваліфікаційного бакалаврського проєкту та його допущення до захисту перед ЕК: Загалом, представлений кваліфікаційний бакалаврський проєкт відповідає встановленим вимогам методичних вказівок щодо структури, обсягу та змісту, та рекомендується до захисту з **високою оцінкою**, а Урденку Денису Олександровичу може бути присвоєний освітньо-кваліфікаційний рівень «бакалавр» галузі знань 12 «Інформаційні технології» за спеціальністю 122 «Комп'ютерні науки».

Науковий керівник

_____ д. е. н., Мозгаллі О.П.

(підпис)

"10" червня 2024 р.

Рецензія
на кваліфікаційний бакалаврський проєкт
Урденка Дениса Олександровича

тема
«Інформаційна система для передачі зашифрованих провідомлень»

Актуальність теми кваліфікаційного бакалаврського проєкту і доцільність її розроблення визначаються зростаючим обсягом цифрової комунікації та необхідністю забезпечення конфіденційності та безпеки обміну інформацією. Автор розуміє, що в сучасному світі, де пересилання даних відбувається через різноманітні канали зв'язку, важливо мати ефективні та надійні механізми шифрування для захисту конфіденційної інформації від несанкціонованого доступу. Висвітлена потреба дозволяє чітко зрозуміти актуальність та значення для безпечної комунікації.

Якість проведеного дослідження в бакалаврському проєкті є високою, що дозволило усвідомити різноманітні аспекти системи, включаючи її архітектуру, алгоритми шифрування, механізм аутентифікації. Аналіз існуючих інформаційних систем проведений в роботі дав розуміння сильних та слабких сторін, що дозволило логічно вивести автором функціональних та нефункціональних вимог до системи та вибору оптимальних архітектурних рішень.

Позитивні риси кваліфікаційного бакалаврського проєкту виявляються в здатності глибоко проаналізувати та систематизувати інформацію щодо інформаційних систем передачі даних. Автор демонструє високий рівень розуміння предметної області передачі повідомлень та вміння ефективно застосовувати теоретичні знання у практичних цілях.

Зауваження до кваліфікаційного бакалаврського проєкту є незначними, зокрема наявні стилістичні помилки, недостатньо опрацьовано підходи українських вчених щодо розроблення систем передавання зашифрованих повідомлень, що свідчить про високу якість виконаної роботи. Розроблена автором серверна частина системи є готовою для тестування, запуску, реалізації запропонованих подальших вдосконалення системи. Це відкриває шлях для подальшого розвитку проекту та можливості його впровадження у практичне використання, сприяючи вирішенню реальних завдань

Практична значимість висновків і рекомендацій даного кваліфікаційного бакалаврського проєкту полягає в важливому внеску у розвиток інформаційних систем для передачі конфіденційної інформації. Професіоналізм, використання актуальних архітектурних та програмних рішень свідчать про високий рівень підготовки автора. Вважаю, що робота виконана на високому професійному рівні, має теоретичну та практичну значущість й заслуговує на оцінку «**Відмінно**».

Доцент кафедри інформаційних технологій і
програмування факультету математики,
інформатики та фізики Український державний
університет імені Михайла Драгоманова _____ Умрик М.А.

АНОТАЦІЯ

кваліфікаційного бакалаврського проєкту студента 4 курсу
Навчально-наукового інституту «Інститут інформаційних технологій в економіці»
Урденка Дениса Олександровича, виконаної на тему
«Інформаційна система для передачі зашифрованих повідомлень»
Київ: кафедра інформаційних технологій в економіці, 2024 р.

Кваліфікаційний бакалаврський проєкт присвячений актуальній проблемі передавання зашифрованих повідомлень, яку пропонується вирішити і удосконалювати за рахунок сучасник інформаційних технологій.

Кваліфікаційний бакалаврський проєкт складається з трьох розділів, що логічно пов’язані між собою.

В першому розділі проводиться характеристика предметної області, що охоплює веб-сайти для передачі зашифрованих повідомлень. Аналізуються існуючі платформи, їхні переваги та недоліки, розроблюється порівняльна таблиця. Розглядаються потреби користувачів у безпеці та конфіденційності при обміні інформацією в мережі Інтернет. Визначаються бізнес цілі.

Другий розділ є проектним і присвячений проектуванню системи, розробленню її архітектури, виконанню постановки та розробленню алгоритму розв’язання задач, які необхідно вирішити для успішної реалізації бакалаврського проєкту.

Третій розділ – конструктивний. Тут наведено інформаційне, технічне програмне та організаційне забезпечення для системи передавання зашифрованих повідомлень. Висвітлені питання організації інформаційного забезпечення: розроблена структура інформаційного забезпечення, описані та наведені в додатках форми вхідних та вихідних документів. Обґрунтований комплекс технічних засобів, а також програмне забезпечення, що використовується для створення системи передавання інформаційних повідомлень. Додатково було створено прототип веб-сайту.

Висновки містять рекомендації щодо доцільності розроблення та впровадження системи передавання зашифрованих повідомлень через мережу Інтернет.

РЕФЕРАТ

Кваліфікаційний бакалаврський проєкт містить 79 сторінки, 3 таблиці, 30 рисунків, список літератури з 27 найменувань, 4 додатків.

ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ ПЕРЕДАЧІ ЗАШИФРОВАНИХ ПОВІДОМЛЕНЬ

Ключові слова: система, вимоги, аналіз, шифрування, розробка.

Предметом дослідження є інформаційні системи та методи шифрування даних.

Об’єктом дослідження виступає інформаційна система для передачі зашифрованих повідомлень.

Мета кваліфікаційного бакалаврського проєкту полягає у розробиці інформаційної системи для передачі зашифрованих повідомлень.

Завданням кваліфікаційного бакалаврського проєкту є дослідження предметної області, моделювання та розробка інформаційної системи.

Апаратні та програмні засоби, що використовувалися при проектуванні:

- Umlet;
- Figma;
- MongoDB;
- Redis;
- Docker;
- VSCodium;
- Мова програмування Go 1.22.1;

Результати досягнуті в процесі роботи – спроектована інформаційна система для передачі зашифрованих повідомлень.

Одержані результати можуть бути використані в якості самостійної розробки або удосконалені, інтегровані в іншу систему в майбутньому.

Рік виконання випускного бакалаврського проєкту – 2024.

Рік захисту бакалаврського проєкту – 2024.

ЗМІСТ

ВСТУП

У сучасному інформаційному суспільстві, де технології розвиваються і швидко змінюються, розробка та впровадження ефективних інформаційних систем стає невід'ємною складовою успішного функціонування організацій та підприємств. Бакалаварський проєкт, присвячений розробленню інформаційної системи, є актуальним завданням, що дозволяє не лише досліджувати сучасні підходи до проектування інформаційних систем, але й створювати конкретні рішення для вирішення завдань у визначеній предметній області.

Об’єктом дослідження є інформаційна система для передачі зашифрованих повідомлень. Ця система виступає важливим елементом в забезпеченні конфіденційності, цілісності та доступності інформації під час її передачі через інтернет-мережу.

Предметом дослідження є інформаційні системи та методи шифрування даних, які використовуються для захисту конфіденційності та цілісності інформації, що передається через мережу. Дослідження цих систем та методів спрямоване на розуміння їхньої структури, принципів роботи та ефективності в захисті інформації в різних умовах експлуатації.

Метою цієї кваліфікаційної бакалаврської роботи є розробка інформаційної системи для передачі зашифрованих повідомлень, що передбачає проєктування та розробку програмного рішення, яке забезпечить ефективне шифрування та дешифрування даних, а також безпечну передачу цих даних через мережу.

У цьому бакалаврському проєкті було визначено характеристику задачі та здійснено аналіз предметної галузі, включаючи опис предметної області та об’єкта дослідження, аналіз існуючих інформаційних систем та розробку концепції інформаційної системи. Саме перший розділ дозволив визначити напрямки для подальшої розробки.

В другому визначаються вимоги та обґрунтування методології проектування та постановку задачі, функціональну модель задачі,

характеристику самої задачі, а також вивчення вихідної та вхідної інформації, а також алгоритму функціонування системи.

Третій розділ фокусується на розробці рішень для інформаційної системи за видами забезпечення, включаючи інформаційне, організаційне, програмне та технічне забезпечення. Це дозволило визначити конкретні компоненти та інструменти, які будуть використовуватися для успішної реалізації інформаційної системи. Також було розглянуто на практиктиці роботу серверної частини системи.

У цьому контексті, кваліфікаційний проєкт виступає як важливий етап у розробці та впровадженні інформаційних технологій, що відповідають вимогам сучасного бізнесу та сприяють оптимізації роботи організацій.

РОЗДІЛ 1. ХАРАКТЕРИСТИКА ТА АНАЛІЗ ПРОЦЕСІВ ІТ-ПІДТРИМКИ ПЕРЕДАВАННЯ ЗАШИФРОВАНИХ ПОВІДОМЛЕНЬ

1.1. Характеристика предметної області та об’єкта дослідження

З урахуванням зростаючої кількості цифрових комунікацій, захист конфіденційної інформації стає критично важливим завданням для користувачів та підприємств. Основною метою цієї предметної області є створення ефективних та безпечних засобів для обміну інформацією, забезпечуючи конфіденційність, цілісність та доступність даних.

Основні завдання які вирішуються в цій галузі це розробка веб-сервісів та додатків з високим рівнем безпеки для передачі зашифрованих повідомлень та розробка інтуїтивно зрозумілих інтерфейсів для зручного користування та обміну зашифрованими повідомленнями.

Потенційні сфери застосування:

- Особисті комунікації. Захист приватних розмов та обміну особистою інформацією. Наприклад, передача паролів.

- Корпоративні комуникації. Забезпечення безпечного обміну інформацією в межах корпорацій та між партнерами.

- Сектор охорони здоров'я. Захист конфіденційних медичних даних та обмін ними між фахівцями.

Ця предметна область є актуальною і постійно еволюціонує відповідно до вимог сучасного цифрового світу, спрямованого на забезпечення безпеки та конфіденційності у всіх сферах комунікацій.

1.2. Аналіз літературних джерел та практичного досвіду застосування інформаційних систем і технологій для передавання зашифрованих повідомлень.

Згідно даних дослідження 2021 року «Безпека в інтернеті» [1] Київського міжнародного інституту соціології, 23% українців старші за 15 років стикалися зі зламом акаунтів в соціальних мережах або месенджерах. В даному досліджені невідомо які саме методи використовувалися для взламу акаунтів, та це і не важливо бо в кінцевому результаті конфіденційні дані

користувачів потрапили до рук хакерів. Для захисту конфіденційної інформації можна було використовувати месенджери з асиметричними методами шифрування, такі як Threema чи Signal, які не зберігають переписки на своїх серверах а повідомлення шифруються на пристроях користувачів.

Однак месенджери з асиметричним шифруванням не набули достатньої популярності в Україні. Для захисту конфіденційної інформації через незахищені месенджери можна використовувати веб сервіси які дозволяють завантажувати уривки тексту та захищати їх паролем, що дозволяє створити додатковий прошарок захисту даних які небезпечно просто передавати через месенджер і незалежать від реалізації протоколів спілкування месенджерів. За допомогою цих веб сервісів можна ділитися гіперпосиланням на конфіденційну інформацію розміщену та встановити обмеження такі як пароль, одноразовий перегляд, час видалення, щоб захистити конфіденційні дані такі як паролі чи дані банківських карток від зовнішнього втручання і не зберігати дані в чаті месенджеру чи на поштовій скриньці.

Розглянемо практичні приклади схожих системи. Існуючих рішень на ринку не так багато, тож я виділив 3 інформаційні системи: Pastebin [2], One Time Secret [3], Privnote [4]. Порівняння було зроблено у вигляді таблиці (Таблиця 1.1 Порівняльна таблиця).

Таблиця 1.1 Порівняльна таблиця сервісів

Можливості	Pastebin	One Time Secret	Privnote
Задання паролю	Так	Так	Так
Таймер для тимчасових повідомлень	Так	Так	Ні
Email нотифікація	Ні	Ні	Так
Збереження повідомлень в акаунті	Так	Ні	Ні
Видалення після прочитання	Так	Ні	Ні
Підсвітка синтаксису	Так	Ні	Ні
Додавання тегів	Так	Ні	Ні

Джерело: складено автором за матеріалами [2 — 4]

Всі ці сервіси захищають вхідну інформацію від користувача паролем. Найбільше функціоналу є в сервісу Pastebin хоча він не дозволяє одразу відправити на email посилання на повідомлення. Ці платформи є закритим вихідним кодом, тобто невідомо чи шифруються повідомлення на стороні сервера чи ні.

Було визначено основні бізнес цілі:

- Залучення та утримання користувачів. Створення привабливої платформи, яка привертає нових користувачів та забезпечує їх лояльність.

- Збільшення популярності і репутації. Розширення впливу та визнання на ринку, що сприяє залученню нових користувачів та бізнес партнерів.

- Отримання прибутку. Розробка стратегій для отримання прибутку через преміум-послуги, рекламу або через співпрацю з підприємствами.

РОЗДІЛ 2. РОЗРОБКА ВИМОГ І МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ПЕРЕДАВАННЯ ЗАШИФРОВАНИХ ПОВІДОМЛЕНЬ

2.1. Аналіз і специфікація вимог до інформаційної системи для передавання зашифрованих повідомлень.

Після аналізу існуючих практичних рішень було створено функціональні вимоги:

- користувач повинен мати можливість створювати, редагувати, переглядати та видаляти власні повідомлення;

- повідомлення має шифруватися на стороні серверу якщо користувач вказав опцію шифрування;

- в результаті створення повідомлення користувач повинен отримати посилання на щойно створене повідомлення;

- користувачі повинні мати можливість створювати облікові записи та входити в систему.

- не авторизований користувач сайту повинен мати можливість переглянути публічне повідомлення іншого користувача використовуючи посилання на повідомлення і вводу паролю від повідомлення;

Визначимо нефункціональні вимоги:

- час обробки шифрування та дешифрування повідомлення не повинен перевищувати 1 секунду;

- затримка при збереженні та отриманні повідомлень не повинна перевищувати 1 секунду;

- повідомлення з опцією шифрування повинні шифруватися за допомогою криптографічного алгоритму AES-256;

- система повинна підтримувати горизонтальне масштабування для обробки збільшеної кількості запитів;

- користувацький інтерфейс повинен бути простим та інтуїтивно зрозумілим;

- компоненти системи повинні розгортатися за допомогою контейнеризації;

- система повинна бути розроблена за допомогою мікросервісної архітектури.

Для інформаційної системи було обрано мікросервісну архітектуру, розглянемо детальніше що це за архітектура. Мікросервісна архітектура - це архітектурний підхід до розробки програмного забезпечення, при якому додаток розбивається на невеликі автономні сервіси, які називаються мікросервісами. Кожен мікросервіс виконує конкретну функцію та може бути розгорнутий та горизонтально масштабований незалежно від інших сервісів.

Мікросервісну архітектуру, згідно [21-22], може бути визначено наступними характеристиками:

1. Незалежність. Мікросервіси можуть бути розроблені, випробувані, розгорнуті та масштабовані незалежно один від одного. Це дозволяє командам працювати над окремими сервісами без впливу на інші частини системи.

2. Розділення відповідальностей. Кожен мікросервіс відповідає за конкретну функціональність або бізнес-процес. Це сприяє легшій обслуговуваності та розумінню системи.

3. Масштабування. Мікросервіси можуть бути замінені або оновлені незалежно, що полегшує розгортання нових версій та горизонтальне масштабування лише необхідних компонентів.

4. Спільна комунікація через API. Мікросервіси спілкуються між собою за допомогою API (інтерфейсів програмування застосунків), що дозволяє їм взаємодіяти та обмінюватися даними в не залежності від мови програмування.

5. Технологічне різноманіття. Кожен мікросервіс може бути написаний різними мовами програмування та використовувати різні технології, що дозволяє використовувати найкращі інструменти для конкретного завдання.

6. Оркестрація та координація. Для управління взаємодією між мікросервісами часто використовується механізм оркестрації, наприклад, Kubernetes або Docker Swarm. Це також дозволяє автоматично масштабувати систему горизонтально.

Мікросервісна архітектура дозволяє компаніям покращити гнучкість, швидкість розробки та можливість масштабування, але також вимагає

ефективного управління та координації між сервісами. Саме тому вона ідеально підходить для розробки данної системи.

Для розробки інформаційної системи було обрано мову програмування Go. Go [6] або Golang - це мова програмування, розроблена в компанії Google, яка появилася у 2007 році і стала відомою своєю простотою, ефективністю та зручністю для розробки великих та надійних програм. Ось основні причини вибору мови, за якими було обрано мову програмування Go: для програмної реалізації інформаційної системи передавання зашифрованих повідомлень.

По-перше синтаксис мови Go схожий на синтаксис C, що полегшує перехід для тих, хто вже має досвід з інших мов програмування, таких як C, C++, або Java. Також вона включає в себе деякі сучасні концепції, щоб полегшити розробку, наприклад, використання горутин та каналів.

По-друге, Go - компільована мова програмування. Компілятор Go генерує нативний код для конкретної операційної системи та архітектури, що робить виконання програм досить швидким.

По-третє. Go підтримує кілька парадигм програмування, включаючи процедурне та об'єктно-орієнтоване програмування, але вона також включає сучасні елементи, такі як конкурентність та паралельність.

По-четверте, в мову вбудована підтримка конкурентності, що означає, що розробники можуть легко створювати паралельні процеси та використовувати горутини для управління багатьма завданнями одночасно. Це робить Go ефективною для розробки високопродуктивних та масштабованих додатків.

Більш того, ця мова має розширену та ефективну стандартну бібліотеку, що включає в себе різноманітні інструменти для розробників, такі як робота з мережами, криптографія, робота з конфігураціями тощо.

Також слід відмітити, що Go надає інструменти для написання юніт тестів, що сприяє розробці надійних та добре тестованих програм.

Саме тому мова програмування Go добре підходить для створення надійних та ефективних високонавантажених систем, а також для розробки веб-додатків та інших типів програм, , через що її було обрано для програмної реалізації інформаційної системи для передачі зашифрованих повідомлень.

Для зберігання даних користувачів було обрано базу даних MongoDB [7]. MongoDB - це NoSQL документоорієнтована система управління базами даних, яка зберігає дані у форматі BSON, що є бінарним представлення JSON-подібних об'єктів. Розроблена компанією MongoDB Inc., вона стала популярною завдяки своїй гнучкості, швидкості та здатності працювати з великими обсягами даних та динамічно змінюваними схемами.

Основні причини вибору MongoDB для реалізації інформаційної системи для передачі зашифрованих повідомлень є наступними.

Перш за все, дані зберігаються у вигляді документів, які є BSON-подібними об'єктами. Кожен документ містить ключі та значення, схожі на пари ключ-значення у JSON. Це дозволяє мати гнучку схему, що дозволяє зберігати документи з різними полями та структурами.

По-друге, MongoDB використовує мову запитів схожу на JSON, що полегшує роботу з базою даних. Запити можуть бути виконані з використанням методів MongoDB.

По-третє, MongoDB підтримує реплікацію.

По-четверте, Підтримує горизонтальне масштабування, що дозволяє розподіляти дані на кілька серверів для забезпечення високої продуктивності та можливості обробки великого обсягу даних.

Більш того, MongoDB підтримує індексацію та операції текстового пошуку, що робить його відмінним вибором для додатків, де текстовий пошук грає ключову роль.

Саме через це MongoDB знаходить застосування у багатьох сферах, включаючи веб-розробку, аналітику, реєстрацію журналів, системи управління контентом та інші області, де важливо працювати з гнучкою та швидкою системою зберігання даних. І саме тому MongoDB було обрано для реалізації інформаційної системи для передачі зашифрованих повідомлень.

Додатково було обрано кешовану базу даних Redis [8]. Ця база даних дозволяє зменшити затримку доступу до даних, збільшити пропускну здатність, полегшити наватаженнянавантаження на основну базу даних. ОсновнаОсновною її функціяфункцією в даній системі буде зберганнязберігання тимчасових даних, таких як набільшнайбільш

запитувальні дані з MongoDB чи коди підтвердження реєстрації. Основні причини вибору бази даних для роботи системи були наступними:

- дані Redis зберігаються в оперативній пам'яті сервера, а не на окремих дисках, що значно зменшує час відгуку та збільшує швидкість передачі даних;

- в Redis можна зберігати рядки, списки, хеш-таблиці та інші структури даних;

- Redis має механізм реплікації;

- Redis використовує простий механізи доступу до даних за допомогою ключа.

2.2. Постановка задачі

Розробка сайту для передачі зашифрованих повідомлень виходить із сучасних вимог до забезпечення конфіденційності та безпеки інформаційних обмінів [14]. У сучасному світі, де цифрові комунікації займають ключове положення в бізнесі та особистій сфері, виникає необхідність в створенні ефективних інструментів для захисту важливої інформації від несанкціонованого доступу. Задача полягає в розробці ефективної та зручної системи передачі повідомлень з використанням надійних алгоритмів шифрування.

Для розуміння вхідної і вихідної інформації системи була створено Таблиця 2.1 2.2 а також інформаційну модель задачі на Рисунку 2.1.

Таблиця 2.1 Вхідна інформація

Назва вхідного повідомлення	Ідентифікатор	Форма подання	Термін і частота надходження	Джерело
Дані користувача	*USR*	Екранна форма	По мірі заключення	Користувач

Запит на розшифрування повідомлення	REQ	Екранна форма	Щоденно, автоматично	Користувач
Лог-файл системних подій	LOG	Файл	Щотижня автоматично	Система

Джерело: складено автором

Таблиця 2.2 Вихідна інформація

Назва вихідного повідомлення	Ідентифікатор	Форма подання і вимоги до неї	Періодичність видання	Термін видання і допустимий час затримки	Користувач інформації
1	2	3	4	5	6
Повідомлення	MSG	Файл	За запитом користувача	Затримка в 1 хвилину	Автор та призначений отримувач
Звіт про події системи	EVT	Файл	Щотижня	Щоп’ятниці о 23:00	Адміністратор системи

Джерело: складено автором

Рисунок 2.1 - Інформаційна модель задачі

Джерело: складено автором

Основні економічні переваги є наступними [13]:

1. Покращення конкурентоспроможності. Запровадження ефективної системи шифрування і передачі повідомлень робить продукт більш конкурентоспроможним на ринку інформаційних технологій, забезпечуючи конкурентні переваги у сфері безпеки та приватності.

2. Залучення корпоративних клієнтів. Врахування високих стандартів безпеки може робити цей продукт привабливим для корпоративних клієнтів, які цінують конфіденційність у бізнес-спілкуванні.

3. Збільшення довіри користувачів. Забезпечення захищеної платформи для обміну повідомленнями допомагає підвищити рівень довіри користувачів до сервісу, що може призвести до збільшення активності та лояльності.

4. Можливість розвитку додаткових сервісів. Існує можливість розширення функціоналу та надання додаткових платних сервісів, таких як розширена система шифрування, аудит безпеки, тощо, що може призвести до додаткового економічного зростання.

З метою вчасного впровадження розробленого веб-сайту на ринку, встановлені чіткі терміни розробки системи. Загальний графік робіт передбачає завершення розробки та тестування протягом 3 місяців, після чого розпочнеться етап пілотного впровадження та коригування можливих недоліків.

У рамках вирішення задачі визначені ключові посадові особи, відповідальні за різні аспекти системи наведені на рисунку 2.2.

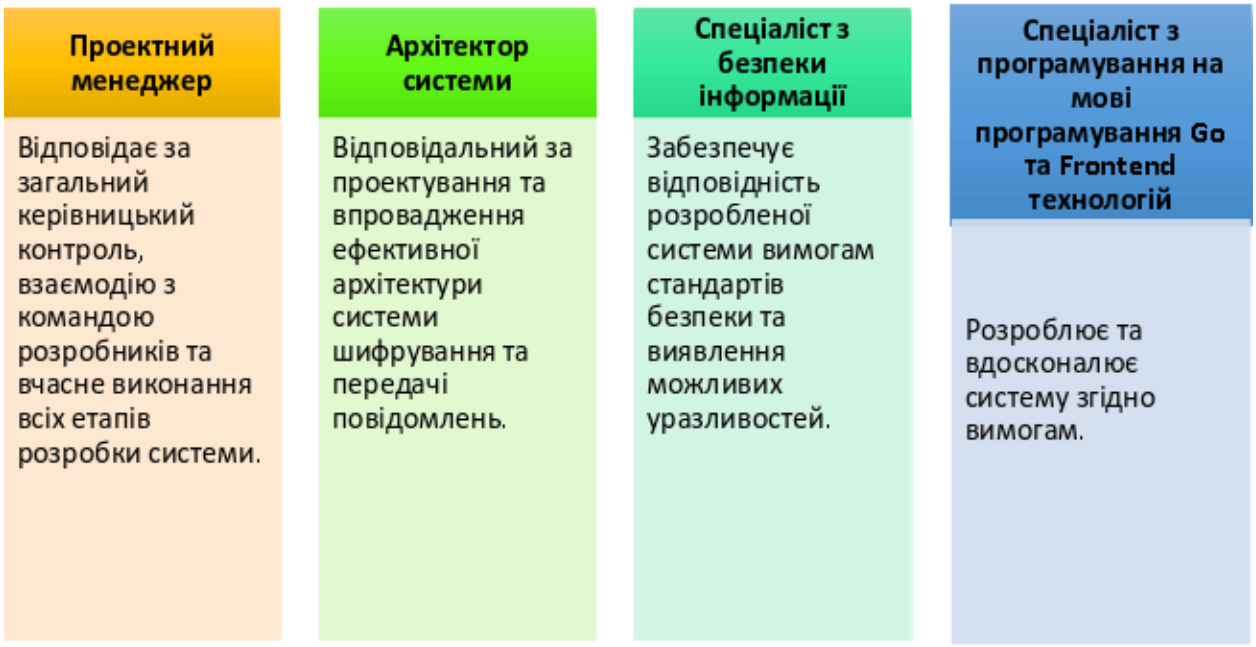

Рисунок 2.2 - Ключові посадові особи по роботі з інформаційною системою для передачі зашифрованих повідомлень

Джерело: складено автором

Основним результатом вирішення поставленої задачі є функціональний та безпечний веб-сайт для передачі зашифрованих повідомлень. Крім того, вимагається:

- Зручний інтерфейс користувача. Створення інтуїтивно зрозумілого та зручного інтерфейсу для користувачів.

- Висока швидкість передачі повідомлень. Забезпечення ефективної та швидкої передачі зашифрованих повідомлень між користувачами.

Ці вимоги спрямовані на створення продукту, який буде відповідати сучасним стандартам безпеки та високим вимогам користувачів.

2.3. Моделювання інформаційної системи для передавання зашифрованих повідомлень.

Для моделювання системи використовується методологія Unified Modeling Language (UML) [4] для створення діаграм, що дозволить уявити функціональність та взаємодію користувача з системою. Використання UML дозволяє не лише візуалізувати процеси та взаємозв'язки, але й забезпечити базовий рівень документації, необхідної для подальшої розробки та імплементації інформаційної системи

Для розуміння сценаріїв використання системи була створена діаграма прецендентів яка зорбражена на Рисунку 2.3

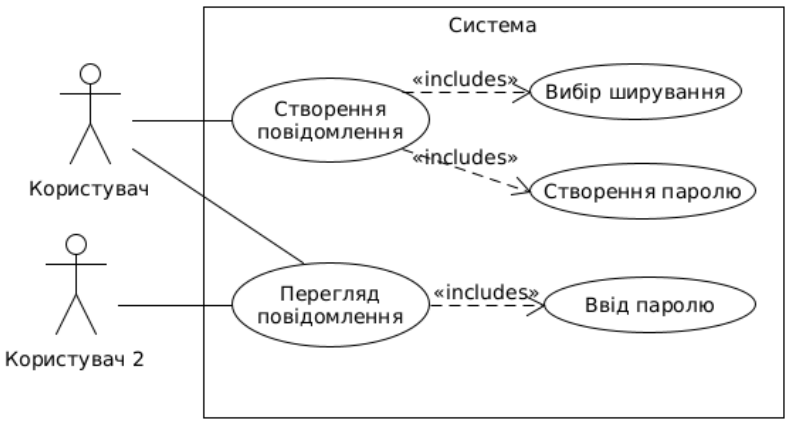

Рисунок 2.3 - Діаграма прецендентів

Джерело: складено автором

В системі є користувачі вони можуть читати створювати повідомлення та читати повідомлення за посиланню, якщо знають пароль. Вибір методу шифрування та паролю буде мати вплив на те як саме повідомлення шифруються. Система має бути максимально автономною і зберігати конфіденційність, тож окрім користувачів із системою ніхто не повинен бути зв’язаний.

Для розуміння взаємодії об’єктів було створено діаграми послідовностей (Рисунок 2.4 та Рисунок 2.5).

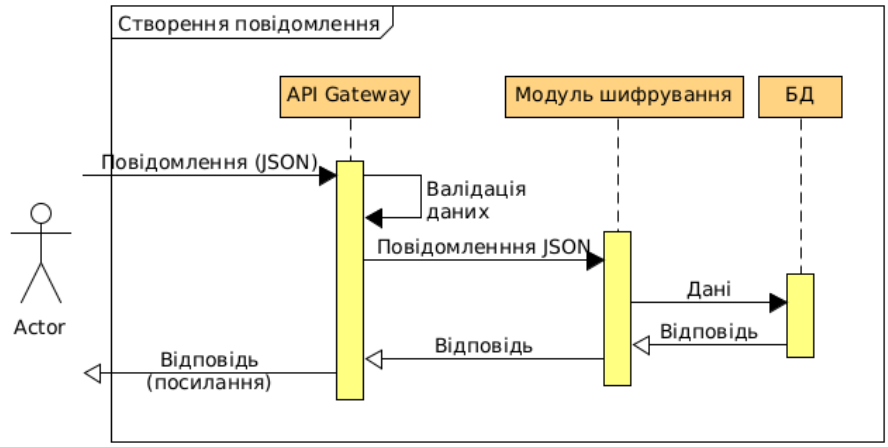

Рисунок 2.4 - Діаграма послідовностей

Джерело: складено автором

Для того щоб створити повідомлення користувач через Frontend відправляє повідомлення до API Gateway який валідує запит. Після чого повідомлення у форматі JSON з іншими метаданими відправляється до модуля шифгрування де вони записуються в базу даних. Модуль шифрування створюю відповідь для користувача в залежності від того чи вдалося записати дані в базу даних. Якщо все пройшло нормально то користувач отримує посилання на повідомлення.

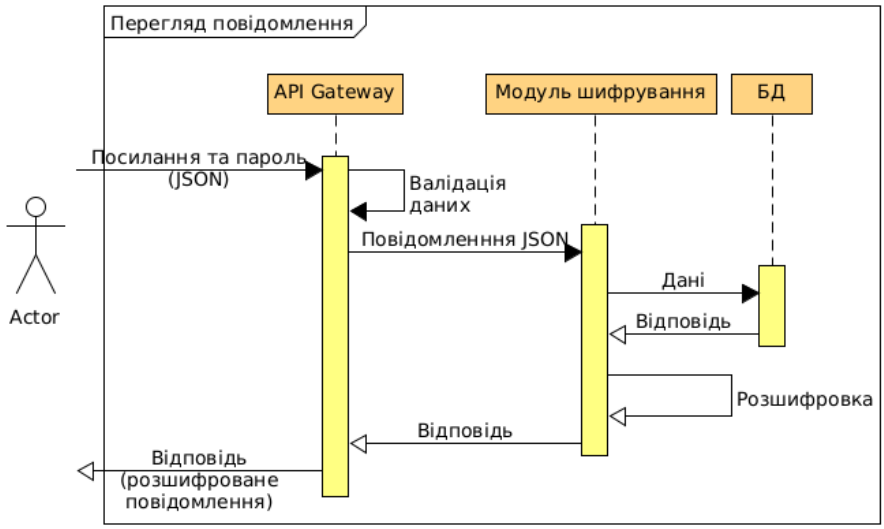

Рисунок 2.5 - Діаграма послідовностей

Джерело: складено автором

Для того щоб переглянути повідомлення коритувач переходить за посиланням та вводить пароль. Frontend відправляє до API Gateway дані у форматі JSON та валідує їх. Далі модуль шифрування читає по айді

повідомлення з бази даних та розшифровує текст. Якщо розшифрування пройшло успішно то користувач отримує розшифроване повідомлення.

Таким чином, в цьому розділі було визначено ролі користувачів, їхні функціональні можливості та очікування від системи, сформульовано перелік функцій, які повинна мати система, з урахуванням потреб користувачів, описано такі характеристики системи, як продуктивність, надійність, безпека та зручність використання. Також було створено модель системи, яка описує її структуру, зв'язки між її компонентами та потоки даних, й розроблено діаграми потоків даних та інші моделі, які о’писують, як система буде працювати.

РОЗДІЛ 3. РОЗРОБКА РІШЕНЬ ДЛЯ ІС ЗА ВИДАМИ ЗАБЕЗПЕЧЕННЯ

3.1. Інформаційне забезпечення

Створимо інфологічну модель (Рисунок 3.1).

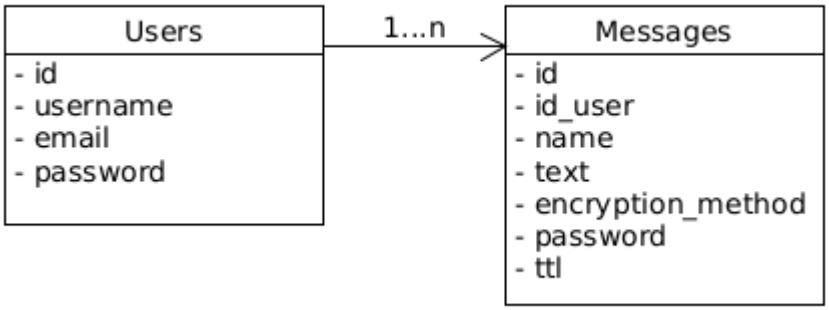

Рисунок 3.1 - Інфологічна модель

Джерело: складено автором

Інфологічна модель складається з двох сутностей це користувачі (Users) та повідомлення (Messages). Для індетифікації користувачі було обрано такі атрибути як ім’я користувача, адреса електронної поштової скриньки, пароль та айді. Користувачі можуть створювати безліч повідомлень, їх будемо індетитифікувати за власним айді та айді користувача який його створив, ім’ям повідомлення, текстом, методом шифрування, паролем та часом життя цього повідомлення.

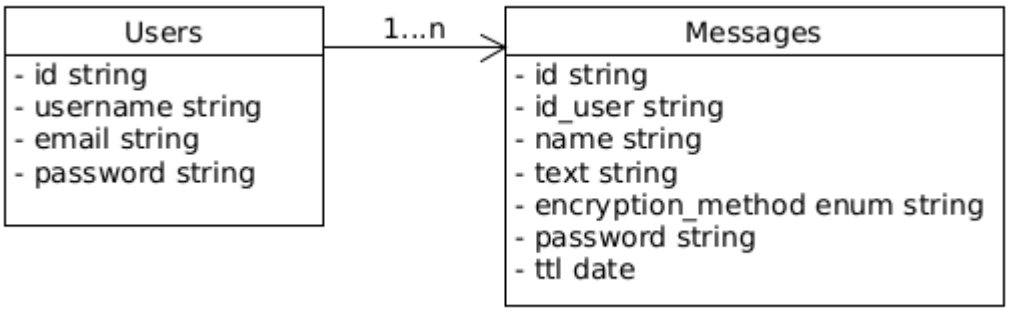

Рисунок 3.2 - Даталогічна модель

Джерело: складено автором

Даталогічна модель, яку зображено на Рисунку 3.2 повністю повторює інфологічну модель. Тут слід пояснити типи даних атрибутів сутностей. В користувачів всі атрибути мають тип строк. В повідомелені теж усе строки

окрім ttl, він повинен мати тип данних date, що дозволить в будь який момент змінити відображення дати.

Створимо прототипи екранних форм. Основна увага користувача буде зосереджена на головній сторінці з повідомленнями (Рисунок 3.3). Додатково було створено більш наглядний прототип вигляду вебсайту через додаток Figma (Рисунок 3.4).

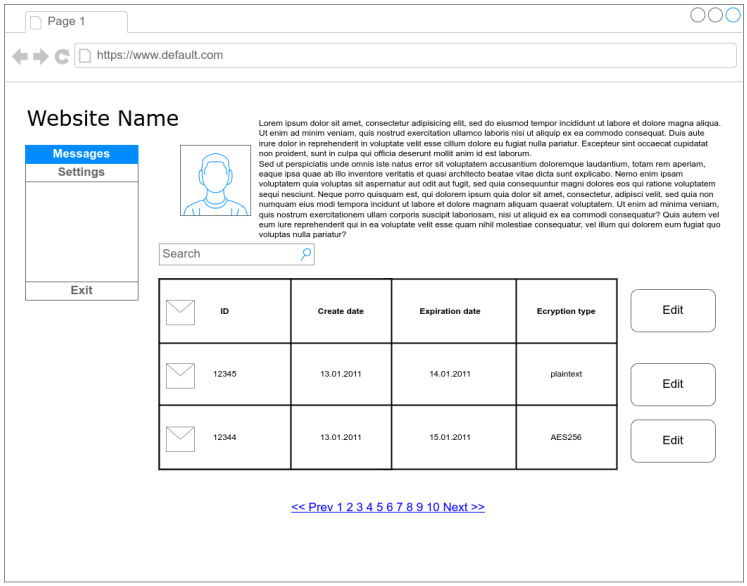

Рисунок 3.3 - Прототип головної сторінки

Джерело: складено автором

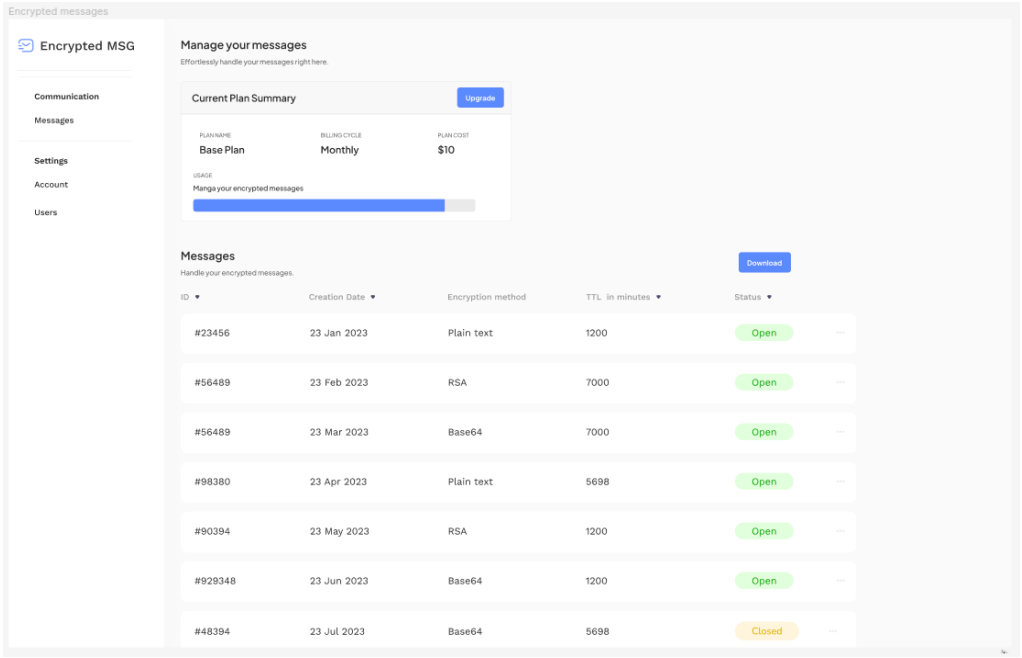

Рисунок 3.4 - Прототип головної сторінки у Figma

Джерело: складено автором

Сайт повинен дотримуватися мінімалістичного стилю щоб користувачів нічого не відволікало. Головна сторінка (Рисунок 3.4) повинна складатися з

таблиці повідомлень користувача де вказано айді повідомлення, дату створення, дату закінчення дії, тип шифрування, та статус. Інші навігаційні елементи розміщуються збоку.

Рисунок 3.5 - Прототип форми створення повідомлення

Джерело: складено автором

Форма створення повідомлення (Рисунок 3.5) теж повинна бути мінімалістична і складається лише з назви повідомлення, тексту повідомлення, дати закінчення дії та паролю.

3.2. Технічне забезпечення

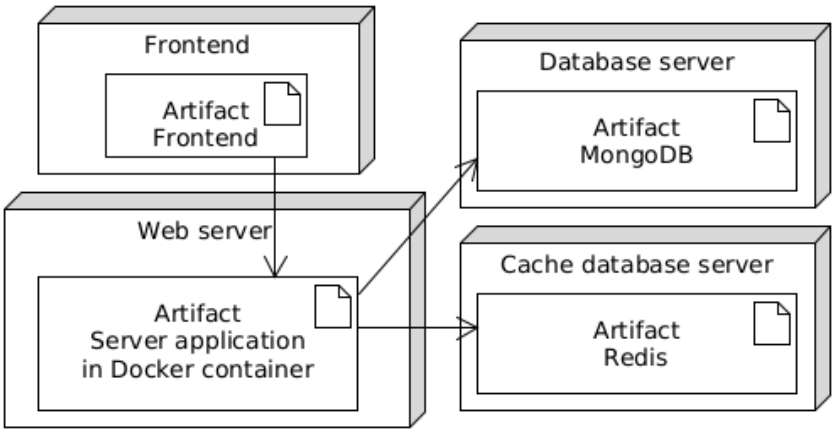

Рисунок 3.6 - Діаграма розгортання

Джерело: складено автором

Для розгортання системи зображеної на рисунку 3.6 знадобиться Docker. Кожен компонент системи, а це Frontend, веб сервер та база даних, будуть запускатися в ізольованих контейнерах через docker compose. Протокол зв’язку між Frontend та Web server був обраний HTTP. Web server з MongoDB та Redis спілкуються вже своїм протоколом.

Для налаштування Web server використовується конфіг файл з синтаксисом YAML. В ньому вказується базовий шлях для звернення до серверу, порт, ключ для шифрування токенів автентифікації користувачів, інформація для підключення до бази даних MongoDB та Redis, ключ для шифрування повідомлень у випадку вибору шифрування «internal».

Алгоритм шифрування повідомлень в програмі має декілька варіантів залежно від типу шифрування, визначеного в тілі JSON в HTTP запиті. Перш за все, є режим «plaintext», при якому повідомлення не шифрується, а просто зберігається в базі даних як текст. Це найпростіший режим, де дані залишаються відкритими і доступними для читання. Цей варіант користувач може використовувати щоб зберігати свої дані які є не чутливими або вже зашифровані дані самим користувачем.

Далі, режим «password» дозволяє захистити повідомлення паролем. Дані які передаються на сервер не шифруються, але для їх прочитання необхідно ввести правильний пароль. Це забезпечує додатковий рівень безпеки, оскільки доступ до даних має лише той, хто знає потрібний пароль.

У режимі «internal» повідомлення шифрується за допомогою AES-256 шифрування з використанням ключа, що зберігається на сервері. Це забезпечує захист даних від несанкціонованого доступу, оскільки ключ не відомий зовнішнім користувачам. Цей режим можна використовувати якщо потребується подвійне шифрування для вже зашифрованих користувачем повідомлення.

Режим «aes» передбачає шифрування повідомлення за допомогою AES-256 шифрування з ключем, який вказується користувачем в полі пароля у JSON у HTTP запиті. Це надає користувачам можливість використовувати власні ключі для шифрування їх даних, забезпечуючи додатковий рівень контролю над безпекою. Варто зазначити що ключ для розшифрування повідомлення користувача не зберігається на сервері.

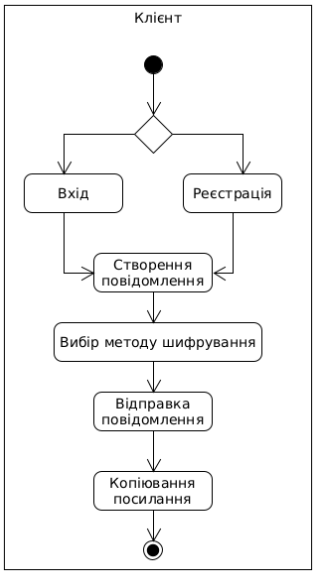

Рисунок 3.7 - Алгоритм дій користувача

Джерело: складено автором

Алгоритм дії користувача (Рисунок 3.7) дуже простий: користувач реєструється або входить в існуючий акаунт, далі він переходить до створення повідомлення, обирає метод шифрування. Після цих дій він отримує посилання на своє повідомлення.

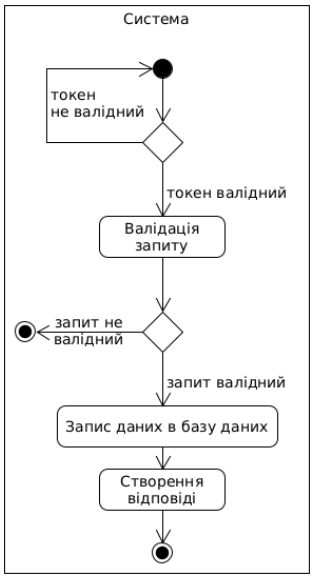

Рисунок 3.8 - Алгоритм дій системи

Джерело: складено автором

Алгоритм дії системи коли вона отримала запит на створення повідомлення, представлено на Рисунку 3.8. Для початку сисема валідує токен та запит і у випадку якщо все валідне зашифровує та створює потрібний JSON

документ в базі даних. Коли повідомлення створене, то система повертає відповідь у вигляді посилання на створене повідомлення.

3.3. Програмне забезпечення

Розробимо діаграму компонентів (Рисунок 3.9).

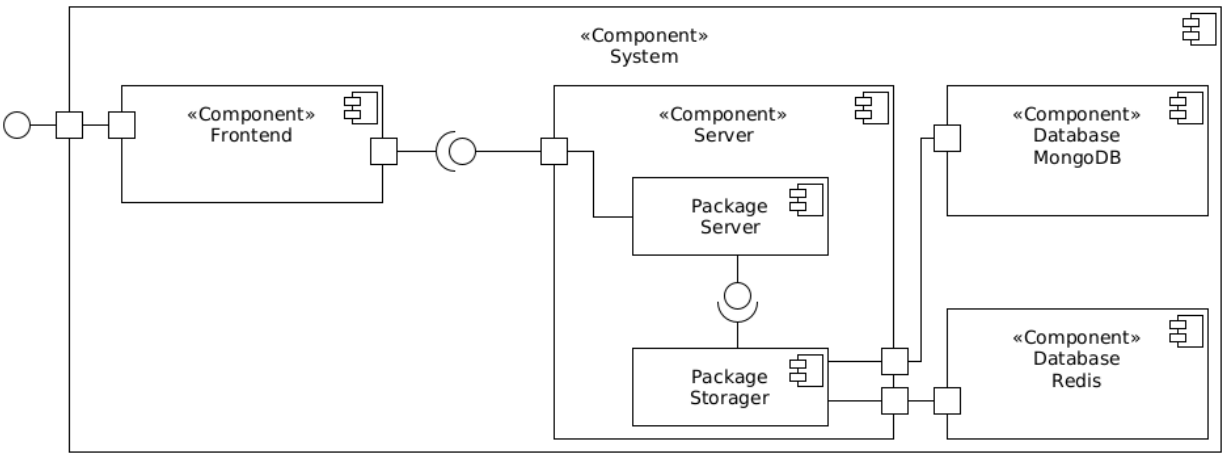

Рисунок 3.9 - Діаграма компонентів

Джерело: складено автором

Уся система складається з Frontend частини, серверу та баз даних. Сервер представляє собою розроблену програму на мові програмування Go. Сама програма також має містити в собі пакет storager який дозволяє не прив’язуватися до реалізації бази даних а містить в собі інтерфейси для реалізації, які повинні бути в головному пакеті Server, цей пакет треба самостійно описати в приєднати до коду програми на Go. Frontend частина майже повністю залежить від реалізації Server частини.

З урахування сучасних вимог розгортання систем, усі ці компоненти повинні працювати в Docker контейнері. Це необхідно для забезпечення максимальної гнучкості та масштабованості системи. Крім того, використання Kubernetes для керування цими контейнерами дозволяє автоматизувати процеси масштабування, управління ресурсами і відновлення під час виникнення помилок. Такий підхід забезпечить надійність, ефективність та зручність адміністрування всіх компонентів системи.

3.4. Результати реалізації інформаційної системи для передавання зашифрованих повідомлень

Різні варіанти шифрування дозволяють програмі адаптуватися до різних вимог щодо безпеки та конфіденційності даних, забезпечуючи гнучкість та захист від різних загроз. Для більш конкретного розуміння роботи системи, розглянемо більш детально що як саме система авторизує та захищає дані користувачів. Для того щоб користувач міг використовувати систему для створення повідомлень йому треба зареєструватися. Під час реєстрації користувач відправляє в тілі HTTP запиту свій логін і пароль. Система перевірить базу даних на наявність даного логіну в базі даних MongoDB, у випадку якщо логін ще не зайнятий то створюється ключ значення в базі даних Redis, після чого користувач отримує відповідь від сервера в тілі відповіді є посилання на яке повинен перейти користувач щоб підтвердити реєстрацію, і тоді сервер створює обліковий запис користувача в колекції документів MongoDB. Для подальшої роботи системи користувач відправляє запит на авторизацію, після чого користувачу видається JWT [12] токен який потім буде використовуватися в заголовках кожного запиту до системи.

Щоб створити повідомлення користувач відправляє запит до системи, де вказує потрібну інформацію для успішного створення. Наприклад користувач створює повідомлення з типом шифрування «aes», тоді система отримає запит, перевірить валідність JWT токену, якщо не валідний то система поверне помилку, якщо токен валідний то система перевірить валідність запиту, візьме пароль з запиту і зашифрує ним повідомлення через AES-256 шифрування. Система збереже зашифроване повідомлення в форматі Base64 в колекції MongoDB. Схожий алгоритм працює з «internal» шифруванням, тільки ключ для шифрування повідомлення буде використовуватися той який зазначений на сервері а не від користувача. Слід зазначити що користувач може ввести ключ пароль менший за 256 біт, тому сервер спочатку хешує пароль за допомогою SHA-256 [11] для отримання ключа розміром 256 біт. Тож, навіть якщо користувач вводить пароль менший за 256 біт, сервер у будь якому випадку створить ключ фіксованого розміру.

Розглянемо детальніше алгоритм шифрування. AES (Advanced Encryption Standard) — це симетричний блочний алгоритм шифрування [9].

Цей алгоритм використовує один і той самий ключ для шифрування і для дешифрування. Розберемо кожен етап цього алгоритму.

Спочатку дані розбиваються на блоки по 128 біт і ключ теж на 128 біт, далі над кожним блоком відбувається ряд дій які називають раунд. Дані в блоці за допомогою операції XOR комбінуються з ключем. Далі відбувається підстановка, дані в блоці проходять через послідовність підстановок байтів за допомогою S-box (Substitution Box), де кожен байт замінюється на інший згідно із таблицею замін. Після чого відбувається зміщення строк вліво і перемішування стовпчиків. Кількість раундів залежить від розміру ключа, для 192 бітного ключа буде 10 раундів, для 192 бітного ключа буде 12 раундів, для 256 бітного ключа 14 раундів. В останньому раунді не відбувається змішування стовпців і це ніяк не позначається на захисті алгоритму. Для розшифровки даних використовуються зворотні дії з використанням ключа до зашифрованих даних.

Розроблена система виконує мінімальні функції які потрібні їй для функціонування, тобто реєстрацію користувачів, авторизацію, CRUD операції над повідомленням в режимі plaintext, password, internal, aes. Коди готового бекенд серверу та додаткових файлів для його запуску описані в Додатку А.

Розглянемо запуск і роботу бекенд серверу. Для запуску усіх компонентів включаючи програму і бази даних використовується Docker і конфігураційний файл у форматі YAML. Для конфігурації серверу треба створити файл «config.yaml», вміст кофігураційного файлу є в Додатку Б. Для запуску системи треба виконати команду «docker-compose up -d --build deepenc» (Рисунок 3.10) в середині робочої директорії програмних кодів де є файл «docker-compose.yaml», вміст якого є у Додатку В.

```
deepenc $ docker compose up -d --build deepenc                                    main
WARN[0000] /home/kibe/go/src/deepenc/docker-compose.yaml: `version` is obsolete
[+] Building 1.3s (13/13) FINISHED                                  docker:default
 => [deepenc internal] load build definition from Dockerfile                 0.0s
 => => transferring dockerfile: 440B                                         0.0s
 => [deepenc internal] load metadata for docker.io/library/golang:1.22.1-alpine3.19  1.1s
 => [deepenc internal] load .dockerignore                                    0.0s
 => => transferring context: 2B                                              0.0s
 => [deepenc builder 1/6] FROM docker.io/library/golang:1.22.1-alpine3.19@sha256:046  0.0s
 => [deepenc internal] load build context                                    0.1s
 => => transferring context: 47.32kB                                         0.0s
 => CACHED [deepenc builder 2/6] WORKDIR /app                                0.0s
 => CACHED [deepenc builder 3/6] COPY . .                                    0.0s
 => CACHED [deepenc builder 4/6] RUN go mod tidy                             0.0s
 => CACHED [deepenc builder 5/6] RUN go build -o ./deepenc main.go           0.0s
 => CACHED [deepenc builder 6/6] RUN touch config.yaml                       0.0s
 => CACHED [deepenc stage-1 1/2] COPY --from=builder /app/deepenc /bin/deepenc  0.0s
 => CACHED [deepenc stage-1 2/2] COPY --from=builder /app/config.yaml /bin/config.ya  0.0s
 => [deepenc] exporting to image                                             0.0s
 => => exporting layers                                                      0.0s
 => => writing image sha256:3553031af8d1232430b585bb68ca4b1b93b643c17fb00b471b955545  0.0s
 => => naming to docker.io/library/deepenc-deepenc                           0.0s
[+] Running 5/5
 ✔ Network deepenc_default            Created                                0.2s
 ✔ Container deepenc-mongo-1          Started                                0.6s
 ✔ Container deepenc-redis-1          Started                                0.6s
 ✔ Container deepenc-mongo-express-1  Started                                0.8s
 ✔ Container deepenc                  Started                                1.0s
deepenc $                                                                    main
```

Рисунок 3.10 - Результат виконання команди запуску контейнерів Docker

Джерело: складено автором

Перевіримо чи запустився головний сервер командою «docker compose logs -f deepenc» (Рисунок 3.11).

```
deepenc $ docker compose logs -f deepenc
WARN[0000] /home/kibe/go/src/deepenc/docker-compose.yaml:
deepenc  | ⇨ http server started on [::]:1234
```

Рисунок 3.11 - Результат виконання команди виводу логів

Джерело: складено автором

Тепер коли вже все працює, можна відправляти HTTP запити до серверу. Для демонстрації роботи надалі запити до серверу будуть відправлятися за допомогою утиліти curl та відформатуємо відповідь від серверу за допомогою утиліти jq. Відправимо запит на реєстрацію користувача (Рисунок 3.12).

```
deepenc $ curl -s --request POST \                                           main
  --url http://localhost:1234/api/signup \
  --header 'Content-Type: application/json' \
  --data '{
        "username": "testusername",
        "password": "testpassword"
}' | jq

{
  "message": "confirm your username by route - /api/verify/746573747573726e616d65da39a3ee
5e6b4b0d3255bfef95601890afd80709"
}
```

Рисунок 3.12 - Результат виконання запиту реєстрації на сервері

Джерело: складено автором

У відповідь отримуємо шлях по якому треба звернутися до серверу щоб підтвердити реєстрацію. Відправимо запит за заданим шляхом до серверу (Рисунок 3.13)

```
deepenc $ curl --request GET \                                                    main
  --url http://localhost:1234/api/verify/74657374757365726e616d65da39a3ee5e6b4b0d3255bfef95
601890afd80709
```

Рисунок 3.13 - Результат виконання команди підтвердження реєстрації

Джерело: складено автором

Утиліта не виводить в термінал код відповідь від серверу, тож перевіримо чи дійсно створився користувач в базі даних MongoDB, для цього треба перейти в браузер за адресою localhost:8081, тоді відкриється графічний інтерфейс Mongo Express (Рисунок 3.14).

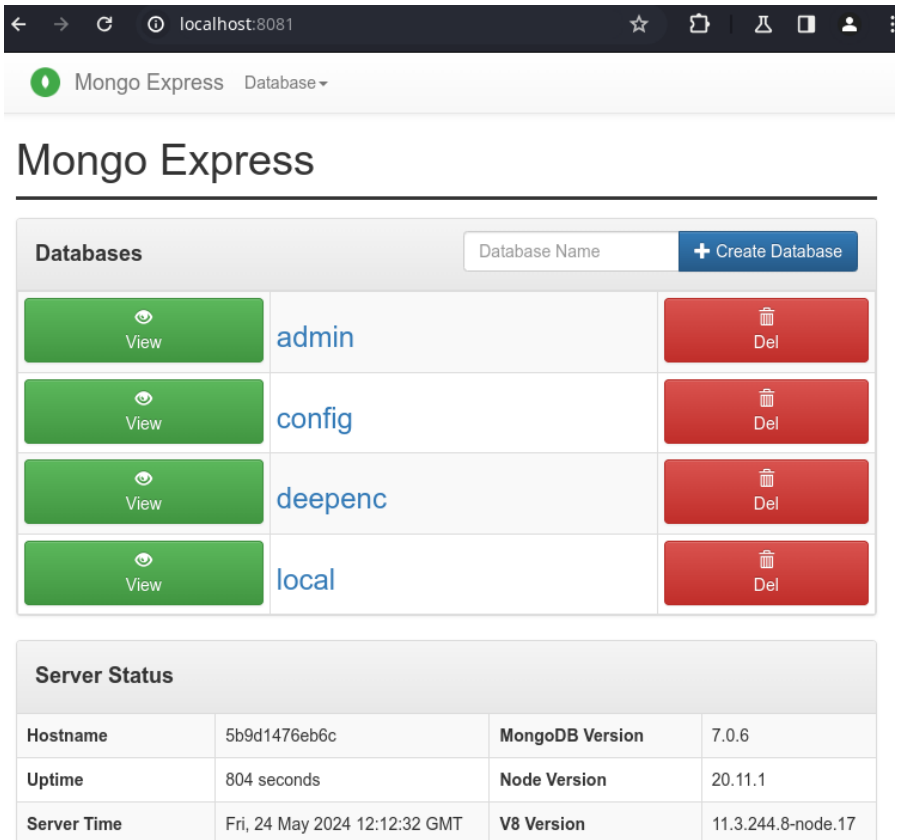

Рисунок 3.14 - Інтерфейс Mongo Express до бази даних MongoDB

Джерело: складено автором

Зайдемо в базу даних deepenc в Mongo Express та виберемо колекцію Users, там буде єдиний документ що буде сигналом того що користувач створений (Рисунок 3.15).

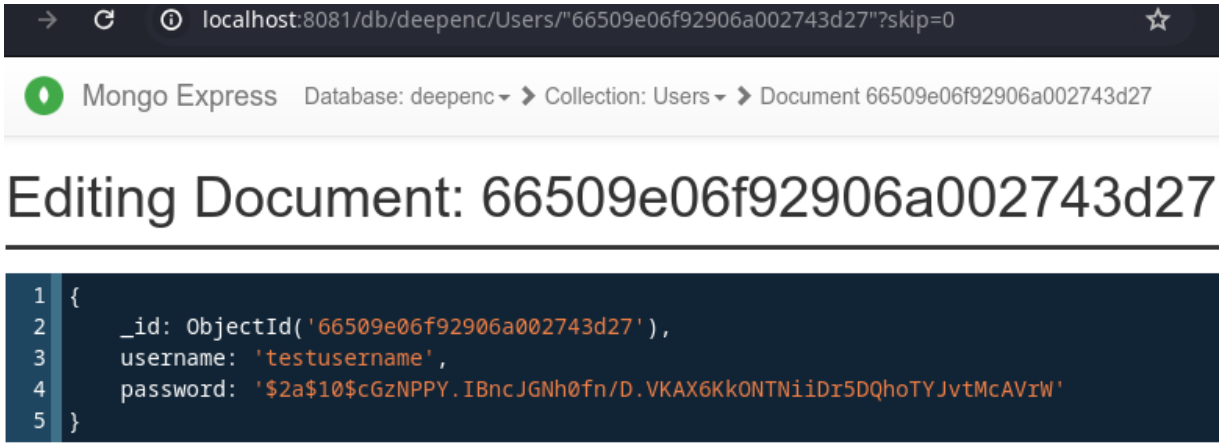

Рисунок 3.15 - Запис користувача в колекції MongoDB

Джерело: складено автором

Варто зазначити що в полі password дані не зберігаються у відкритому вигляді, а хешуються алгоритмом bcrypt. Наступним кроком буде проходження авторизації та отримання від серверу JWT токену. Відправимо на сервер запит з даними які вводилися при реєстрації (Рисунок 3.16).

```
deepenc $ curl -s --request POST \                                    main
  --url http://localhost:1234/api/signin \
  --header 'Content-Type: application/json' \
  --data '{
        "username": "testusername",
        "password": "testpassword"
}' | jq

{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiI2NjUwOWUwNmY5MjkwNmEwMDI3NDNkMj
cifQ.-NyTB5hfSE99VwNufL8oioOpXkuV-WzZ0Klujapg9kk"
}
```

Рисунок 3.16 - Результат виконання запиту на авторизації

Джерело: складено автором

У відповідь від серверу отримуємо токен який буде використовуватися для запитів які створюються повідомлення. Відправимо запити на створення повідомлення з різним encoding_type які опрацьовуються системою (Рисунок 3.17 3.18 3.19 3.20). Дані запити є правильними і повернуть HTTP код відповіді 201, тому утиліта більше нічого не виводить.

```
deepenc $ curl --request POST \                                                    main
  --url http://localhost:1234/api/messages \
  --header 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiI2NjUwOWUw
NmY5MjkwNmEwMDI3NDNkMjcifQ.-NyTB5hfSE99VwNufL8oioOpXkuV-WzZ0Klujapg9kk' \
  --header 'Content-Type: application/json' \
  --data '{
    "content": "Hello World! This is test message",
    "is_private": false,
    "encoding_type": "plaintext",
    "password": "",
    "only_owner_view": false,
    "is_anon": false,
    "is_one_time": false
}'
```

Рисунок 3.17 - Результат виконання запиту на створення повідомлення в режимі plaintext

Джерело: складено автором

```
deepenc $ curl --request POST \                                                    main
  --url http://localhost:1234/api/messages \
  --header 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiI2NjUwOWUw
NmY5MjkwNmEwMDI3NDNkMjcifQ.-NyTB5hfSE99VwNufL8oioOpXkuV-WzZ0Klujapg9kk' \
  --header 'Content-Type: application/json' \
  --data '{
    "content": "Hello World! This is test message 2",
    "is_private": false,
    "encoding_type": "password",
    "password": "testpassword",
    "only_owner_view": false,
    "is_anon": false,
    "is_one_time": false
}'
```

Рисунок 3.18 - Результат виконання запиту на створення повідомлення в режимі password

Джерело: складено автором

```
deepenc $ curl --request POST \                                                    main
  --url http://localhost:1234/api/messages \
  --header 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiI2NjUwOWUw
NmY5MjkwNmEwMDI3NDNkMjcifQ.-NyTB5hfSE99VwNufL8oioOpXkuV-WzZ0Klujapg9kk' \
  --header 'Content-Type: application/json' \
  --data '{
    "content": "Hello World! This is test message 3",
    "is_private": false,
    "encoding_type": "aes",
    "password": "testpassword",
    "only_owner_view": false,
    "is_anon": false,
    "is_one_time": false
}'
```

Рисунок 3.19 - Результат виконання запиту на створення повідомлення в режимі aes

Джерело: складено автором

```
deepenc $ curl --request POST \                                                    main
  --url http://localhost:1234/api/messages \
  --header 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiI2NjUwOWUw
NmY5MjkwNmEwMDI3NDNkMjcifQ.-NyTB5hfSE99VwNufL8oioOpXkuV-WzZ0Klujapg9kk' \
  --header 'Content-Type: application/json' \
  --data '{
    "content": "Hello World! This is test message 4",
    "is_private": false,
    "encoding_type": "internal",
    "password": "",
    "only_owner_view": false,
    "is_anon": false,
    "is_one_time": false
}'
```

Рисунок 3.20 - Результат виконання запиту на створення повідомлення в режимі internal

Джерело: складено автором

Подивимося власний список створених користувачем повідомлень. Для цього відправимо відповідний HTTP запит на сервер (Рисунок 3.21 та Рисунок 3.22).

```
deepenc $ curl -s --request GET \                                                  main
  --url http://localhost:1234/api/messages \
  --header 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiI2NjUwOWU
wNmY5MjkwNmEwMDI3NDNkMjcifQ.-NyTB5hfSE99VwNufL8oioOpXkuV-WzZ0Klujapg9kk' | jq
[
  {
    "id": "66509ecaf92906a002743d28",
    "owner_id": "66509e06f92906a002743d27",
    "content": "Hello World! This is test message",
    "is_private": false,
    "encoding_type": "plaintext",
    "only_owner_view": false,
    "is_anon": false,
    "is_one_time": false
  },
  {
    "id": "66509edef92906a002743d29",
    "owner_id": "66509e06f92906a002743d27",
    "content": "Hello World! This is test message 2",
    "is_private": true,
    "encoding_type": "password",
    "password": "$2a$10$3a8HItX75FCz337PlBjpTuZ.RLOfHdSDeEm.2ZMMfyy7Za8874Sjm",
    "only_owner_view": false,
    "is_anon": false,
    "is_one_time": false
  },
```

Рисунок 3.21 - Результат виконання запиту на вивід створених користувачем повідомлень

Джерело: складено автором

```
    {
      "id": "66509ef5f92906a002743d2a",
      "owner_id": "66509e06f92906a002743d27",
      "content": "iLFDMdK0LuT8HXHL3ONu3fvMhIsGojnoLw/csWDmvc2hSRa3r6nEMO4F8WnAlkXgLE61fq3F+G
eaINvGO1z3",
      "is_private": true,
      "encoding_type": "internal",
      "only_owner_view": false,
      "is_anon": false,
      "is_one_time": false
    },
    {
      "id": "66509f18f92906a002743d2b",
      "owner_id": "66509e06f92906a002743d27",
      "content": "CDucAYzBcIsZcE1BScrEhGIZyfif3+QfRLR2IBcNqldIqGDJa9VED/ijH5QtHzBx7U4kPqzj4c
SdtoMCsUb0",
      "is_private": true,
      "encoding_type": "aes",
      "only_owner_view": false,
      "is_anon": false,
      "is_one_time": false
    }
  ]
```


Рисунок 3.22 Продовження списку створених користувачем повідомлень

Джерело: складено автором

Розглянемо результати створення повідомлень. Вміст повідомлення (поле content) з id «66509ef5f92906a002743d2a» є зашифровані дані у форматі base64 [10], ключем встановленим через конфігураційний файл на сервері. Вміст повідомлення «66509f18f92906a002743d2b» також є зашифрованим, тільки для шифрування використовувався ключ від користувача.

Повідомлення з id «66509ecaf92906a002743d28» це не зашифроване повідомлення і воно є публічним, це означає що можна переглянути це повідомлення без наявного акаунту в системі і без паролю. Відправимо відповідний запит на сервер щоб переконатися що повідомлення доступне без реєстрації, без токену JWT (Рисунок 3.23).

```
deepenc $ curl -s --request GET \
  --url http://localhost:1234/api/messages/public/66509ecaf92906a002743d28 | jq
{
  "id": "66509ecaf92906a002743d28",
  "owner_id": "66509e06f92906a002743d27",
  "content": "Hello World! This is test message",
  "is_private": false,
  "encoding_type": "plaintext",
  "only_owner_view": false,
  "is_anon": false,
  "is_one_time": false
}
```

Рисунок 3.23 - Результат виконання команди виведення повідомлення

Джерело: складено автором

Повідомлення з id «66509edef92906a002743d29» має тип захисту «password» що робить його недоступним без паролю і не є публічним. Переконаємося в цьому, відправимо запит як показано на Рисунку 3.24.

Рисунок 3.24 - Результат виконання запиту на виведення повідомлення яке не є публічним

Джерело: складено автором

Утиліта curl нічого не вивела, це означає що сервер нічого не повернув і повідомлення недоступне без паролю. Тепер відправимо запит з паролем щоб отримати повідомлення (Рисунок 3.25).

```
deepenc $ curl -s --request POST \
  --url http://localhost:1234/api/messages/66509edef92906a002743d29 \
  --header 'Content-Type: application/json' \
  --data '{
        "password": "testpassword"
}' | jq
{
  "content": "Hello World! This is test message 2",
  "is_private": true,
  "encoding_type": "password",
  "password": "$2a$10$3a8HItX75FCz337PlBjpTuZ.RLOfHdSDeEm.2ZMMfyy7Za8874Sjm",
  "only_owner_view": false,
  "is_anon": false,
  "is_one_time": false
}
```

Рисунок 3.25 - Результат виконання команди на отримання приватного повідомлення захищеним паролем

Джерело: складено автором

Як побачимо з успішної відповіді від серверу повідомлення видно. Також слід зазначити що поле «password» є захешовним алгоритмом bcrypt значенням, тобто пароль у відкритому вигляді на сервері не зберігається.

Переконатися в тому чи дійсно повідомлення шифруються а паролі хешуються можна подивившись колекцію документів «Messages» в MongoDB через інтерфейс Mongo Express (Рисунок 3.26).

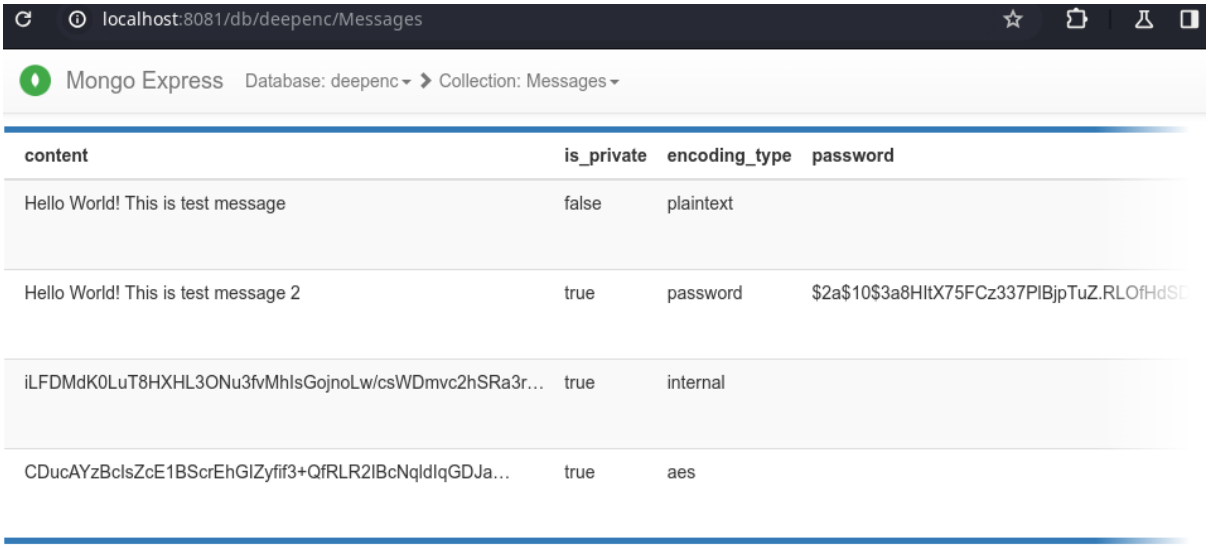

Рисунок 3.26 - Список створених повідомлень в MongoDB

Джерело: складено автором

Наступні кроки покращення кодової бази зводиться до створення Frontend частини системи, додавання нових методів шифрування, створення механізму шифрування файлів, створення функціоналу онлайн менеджеру паролів, розширення інфраструктури системи, додавання рекламних банерів на вебсайт, створення доступу REST API системи для інтеграції в інші сервіси.

ВИСНОВКИ

Даний бакалаврський проєкт є результатом аналізу та дослідження важливої предметної галузі, пов'язаної з розробкою та впровадженням інформаційних систем. В сучасному світі, де інформація стає все більш цінною, питання її захисту набуває все більшої актуальності. Передача даних через Інтернет або інші канали зв'язку може бути небезпечною, якщо вони не захищені від перехоплення та розшифровки. Характеристика обраної області дослідження та вивчення існуючих інформаційних систем надали розуміння та визначили ключові вимоги до розроблюваної системи.

Аналіз існуючих систем, аналіз джерел, вимог, постановка задачі, та розгляд вихідної та вхідної інформації сприяли чіткому визначенню стратегії та цілей системи. Важливим етапом стало розроблення алгоритмів функціонування системи та архітектурні рішення, що визначило ключові елементи та принципи функціонування системи, зокрема забезпечення конфіденційності та цілісності даних, стійкість до перехоплення та розшифровки, простота використання, ефективність роботи.

Для досягнення поставленої мети кваліфікаційного бакалаврського проєкту було досліджено принципи та алгоритми криптографії, розроблено інформаційну систему для передачі зашифрованих повідомлень, створено оригінальне програмне забезпечення для шифрування та дешифрування даних, здійснено тестування та оцінювання працездатності розробленої інформаційної системи.

Розробка рішень для інформаційної системи в різних аспектах забезпечення, таких як інформаційне, організаційне, програмне та технічне, дозволила визначити конкретні компоненти та інструменти для вдалої реалізації системи. Також в бакалаврському проєкті розроблено функціонуючу серверну частину інформаційної системи, яку можна використовувати для подальшого вдосконалення, розширення та використання.

В данному бакалаврському проєкті виокремлюється як важлива ланка вдосконалення та оптимізації інформаційної системи в області передачі зашифрованої інформації. Розроблені концепції та рішення спрямовані на

поліпшення ефективності та безпеки обробки інформації, що на практиці визначає успішність функціонування сучасних інформаційних систем.

Отже, в ході кваліфікаційного бакалаврського проєкту було розроблено інформаційну систему для передачі зашифрованих повідомлень, яка відповідає всім поставленим вимогам і задачам дослідження. Розроблена система може бути використана для захисту конфіденційних даних у різних сферах діяльності, зокрема електронній комерції, банківській справі, державному управлінні, освітній діяльності, медичній сфері, особистому спілкуванні та інших.

В подальшому планується продовжити дослідження в обраному напрями й дослідити можливості використання більш стійких криптографічних алгоритмів, розробити методи захисту інформаційної системи від кібератак, здійснити дослідження щодо можливості інтеграції розробленої системи з іншими інформаційними системами.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Безпека в інтернеті, 2021. – 90 с. – (Київський міжнародний інститут соціології).

2. Pastebin [Електронний ресурс] – Режим доступу до ресурсу: https://pastebin.com/.

3. Onetimesecret [Електронний ресурс] – Режим доступу до ресурсу: https://onetimesecret.com/.

4. Privnote [Електронний ресурс] – Режим доступу до ресурсу: https://privnote.com/#.

5. The Unified Modeling Language [Електронний ресурс] – Режим доступу до ресурсу: https://www.uml-diagrams.org/.

6. Golang Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://go.dev/doc/.

7. MongoDB Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/docs/manual/.

8. Redis Documentation [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://redis.io/docs/latest/.

9. Advanced Encryption Standard (AES) – Гейтерсбург: Federal Information Processing Standards Publication, 2001. – 38 с.

10. Josefsson S. The Base16, Base32, and Base64 Data Encodings [Електронний ресурс] / Simon Josefsson. – 2006. – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc4648.

11. Hansen T. US Secure Hash Algorithms (SHA and HMAC-SHA) [Електронний ресурс] / T. Hansen, D. Eastlake. – 2006. – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc4634.

12. Jones M. JSON Web Token (JWT) [Електронний ресурс] / M. Jones, J. Bradley, N. Sakimura. – 2015. – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc7519.

13. Furnell S. E-Mail Security A Pocket Guide / S. Furnell, P. Dowland., 2010. – (IT Governance Publishing).

14. Stamp M. Information Security: Principles and Practice / Mark Stamp., 2011. – (Wiley).

15. Menezes A. Handbook of Applied Cryptography / A. Menezes, P. van Oorschot, S. Vanstone., 2001. – 816 с. – (CRC Press).

16. Groot J. What Is Data Encryption? (Definition, Best Practices & More) [Електронний ресурс] / Juliana Groot // Digital Guardian. – 2023. – Режим доступу до ресурсу: https://www.digitalguardian.com/blog/what-data-encryption.

17. JEDI: Many-to-Many End-to-End Encryption and Key Delegation for IoT / S.Kumar, Y. Hu, M. Andersen, R. Ada Popa., 2019. – 19 с. – (University of California, Berkeley).

18. Richards M. Fundamentals of Software Architecture: An Engineering Approach / M. Richards, N. Ford., 2020. – 419 с. – (O'Reilly Media).

19. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems / Martin Kleppmann., 2017. – 611 с. – (O'Reilly Media).

20. DevOps. Посібник / П.Дебуа, Д. Вілліс, Д. Кім, Д. Хамбл., 2023. – 384 с.

21. Newman S. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith / Sam Newman., 2019. – 270 с. – (O'Reilly Media).

22. Schenker G. Getting Started with Containerization: Reduce the operational burden on your system by automating and managing your containers / G. Schenker, H. Saito., 2019. – (Packt Publishing).

23. Мартін Р. Чиста архітектура / Роберт Мартін., 2019. – 368 с.

24. Donovan A. The Go Programming Language / A. Donovan, B. Kernighan., 2015. – 400 с. – (Addison-Wesley Professional).

25. Bodner J. Learning Go: An Idiomatic Approach to Real-World Go Programming / Jon Bodner.. – 300 с. – (O’Reilly Media).

26. Woodbeck A. Network Programming with Go: Code Secure and Reliable Network Services from Scratch / Adam Woodbeck.. – 392 с. – (No Starch Press).

27. Steele T. Black Hat Go: Go Programming For Hackers and Pentesters / T. Steele, C. Patten, D. Kottmann., 2020. – 368 с. – (No Starch Press).

ДОДАТКИ

Додаток А

Повний вихідний код:

```
// main.go
package main

import (
	"context"
	"flag"
	"log"
	"net/http"
	"os"
	"os/signal"
	"time"

	"github.com/arimatakao/deepenc/cmd/config"
	"github.com/arimatakao/deepenc/server"
)

var pathToConfig *string = flag.String("config", "./config.yaml", "path to config
yaml file")

func init() {
	flag.Parse()
}

func main() {
	err := config.LoadConfig(*pathToConfig)
	if err != nil {
		log.Fatal(err)
	}

	srv := new(server.Server)
	err = srv.Init()
	if err != nil {
		log.Fatal(err)
	}

	ctx, stop := signal.NotifyContext(context.Background(), os.Interrupt)
	defer stop()

	go func() {
		if err := srv.Run(); err != nil && err != http.ErrServerClosed {
			log.Fatal("Error occurred while running server: ", err.Error())
		} else {
			log.Println("Shutdown server")
		}
```

```
	}()

	<-ctx.Done()
	ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
	defer cancel()
	if err := srv.Shutdown(ctx); err != nil {
		log.Fatal("Server shutdown with error: ", err.Error())
	}

	log.Println("Shutdown is successful")
	os.Exit(0)
}

// cmd/config/config.go
package config

import (
	"errors"
	"os"
	"strconv"

	"gopkg.in/yaml.v3"
)

var (
	Port      string
	MongoURL  string
	RedisURL  string
	JWTSecret string
	AESInternalKey []byte
)

type cfg struct {
	Port      int  ``yaml:"port"``
	MongoDBURL string `yaml:"mongodb_url"`
	RedisURL   string `yaml:"redis_url"`
	JWTSecret  string `yaml:"jwt_secret"`
	AESInternalKey string `yaml:"aes_internal_key"`
}

func LoadConfig(pathToYaml string) error {
	data, err := os.ReadFile(pathToYaml)
	if err != nil {
		return err
	}

	c := new(cfg)
	if err = yaml.Unmarshal(data, c); err != nil {
```

```
		return err
	}

	if c.Port <= 0 {
		return errors.New("port value from config is not allowed")
	}

	if c.JWTSecret == "" {
		return errors.New("jwt_secret field from config is empty")
	}

	if len(c.AESInternalKey) < 8 {
		return errors.New("aes_internal_key field is shorter than 8 symbols in
config")
	}

	Port = strconv.Itoa(c.Port)
	MongoURL = c.MongoDBURL
	RedisURL = c.RedisURL
	JWTSecret = c.JWTSecret
	AESInternalKey = []byte(c.AESInternalKey)

	return nil
}

// utils/utils.go
package utils

import (
	"crypto/aes"
	"crypto/cipher"
	"crypto/rand"
	"crypto/sha256"
	"encoding/base64"
	"errors"
	"io"
)

const (
	EMPTY_SYMBOLS = "                "
)

func EncryptAES256(key []byte, plaintext string) (string, error) {
	if len(key) == 0 {
		return "", errors.New("key is empty")
	}
	if len(plaintext) == 0 {
		return "", errors.New("plaintext is empty")
```

```
	}

	keyHashed := sha256.Sum256(key)

	if len(plaintext) < 16 {
		plaintext += EMPTY_SYMBOLS[len(plaintext):]
	}

	bplaintext := []byte(plaintext)

	block, err := aes.NewCipher(keyHashed[:])
	if err != nil {
		return "", err
	}

	aesgcm, err := cipher.NewGCM(block)
	if err != nil {
		return "", nil
	}

	nonce := make([]byte, aesgcm.NonceSize())
	if _, err := io.ReadFull(rand.Reader, nonce); err != nil {
		return "", err
	}

	ciphertext := aesgcm.Seal(nonce, nonce, bplaintext, nil)

	ciphertextBase64 := base64.StdEncoding.EncodeToString(ciphertext)

	return ciphertextBase64, nil
}

func DecryptAES256(key []byte, base64ciphertext string) (string, error) {
	if len(key) == 0 {
		return "", errors.New("key is empty")
	}
	if len(base64ciphertext) == 0 {
		return "", errors.New("base64ciphertext is empty")
	}
	keyHashed := sha256.Sum256(key)

	ciphertext, err := base64.StdEncoding.DecodeString(base64ciphertext)
	if err != nil {
		return "", err
	}

	block, err := aes.NewCipher(keyHashed[:])
	if err != nil {
```

```
		return "", err
	}

	aesgcm, err := cipher.NewGCM(block)
	if err != nil {
		return "", err
	}

	nonceSize := aesgcm.NonceSize()
	if len(ciphertext) < nonceSize {
		return "", errors.New("nonceSize is bigger than encrypted text")
	}

	// split the nonce from the ciptertext
	nonce, ciphertext := ciphertext[:nonceSize], ciphertext[nonceSize:]

	plaintext, err := aesgcm.Open(nil, nonce, ciphertext, nil)
	if err != nil {
		return "", err
	}

	return string(plaintext), nil
}

// utils/utils_test.go
package utils

import (
	"testing"

	"github.com/stretchr/testify/assert"
)

type TestCaseEncryption struct {
	Name      string
	Key       []byte
	Plaintext string
	WithError bool
}

func TestEncryptAES256(t *testing.T) {
	key := "testkey"
	plaintext := "plaintext1234567"
	plaintextShort := "plaintext"

	cases := []TestCaseEncryption{
		{
			Name:      "key is empty",
```

```
                Key:       []byte{},
                Plaintext: plaintext,
                WithError: true,
            },
            {
                Name:      "plaintext is empty",
                Key:       []byte(key),
                Plaintext: "",
                WithError: true,
            },
            {
                Name:      "key is fine and plaintext is short",
                Key:       []byte("testkey"),
                Plaintext: plaintextShort,
                WithError: false,
            },
            {
                Name:      "key and plaintext is fine",
                Key:       []byte("testkey"),
                Plaintext: plaintext,
                WithError: false,
            },
        }

        for _, testCase := range cases {
            result, err := EncryptAES256(testCase.Key, testCase.Plaintext)

            if testCase.WithError {
                assert.Empty(t, result, testCase.Name)
                assert.NotNil(t, err, testCase.Name)
            } else {
                assert.NotEmpty(t, result, testCase.Name)
                assert.Nil(t, err, testCase.Name)
            }
        }
    }

    type TestCaseDecryption struct {
        Name            string
        Key             []byte
        CiphertextBase64 string
        ExpectedResult   string
        WithError       bool
    }

    func TestDecryptAES256(t *testing.T) {
        key := "testkey"
```

```
	cases := []TestCaseDecryption{
		{
			Name:            "key is empty",
			Key:             []byte{},
			CiphertextBase64: "123",
			ExpectedResult:   "",
			WithError:       true,
		},
		{
			Name:            "base64ciphertext is empty",
			Key:             []byte(key),
			CiphertextBase64: "",
			ExpectedResult:   "",
			WithError:       true,
		},
	}

	for _, testCase := range cases {
		result, err := DecryptAES256(testCase.Key,
testCase.CiphertextBase64)

		if testCase.WithError {
			assert.Empty(t, result, testCase.Name)
			assert.NotNil(t, err, testCase.Name)
		} else {
			assert.NotEmpty(t, result, testCase.Name)
			assert.Nil(t, err, testCase.Name)
		}
	}
}

// server/server.go
package server

import (
	"context"
	"net/http"

	"github.com/arimatakao/deepenc/cmd/config"
	"github.com/arimatakao/deepenc/server/database"
	echojwt "github.com/labstack/echo-jwt/v4"
	"github.com/labstack/echo/v4"
	"github.com/labstack/echo/v4/middleware"
	"github.com/labstack/gommon/log"
)

type Server struct {
	e      *echo.Echo
```

```
	db      database.Storager
	cachedb database.Cacher
}

func (s *Server) Init() error {
	s.e = echo.New()
	s.e.HideBanner = true

	s.e.Pre(middleware.RemoveTrailingSlash())
	s.e.Use(middleware.Logger())
	s.e.Logger.SetLevel(log.INFO)

	s.e.RouteNotFound("/*", func(c echo.Context) error {
		return c.NoContent(http.StatusNotFound)
	})

	basePath := s.e.Group("/api")

	// Public routes
	basePath.POST("/signup", s.SignUp)                    // Registration
	basePath.GET("/verify/:token", s.VerifySignUp)        // Verification
	basePath.POST("/signin", s.SignIn)                    // Login
	basePath.GET("/messages/public/:id", s.GetPublicMessage) // Get public
message by id
	basePath.POST("/messages/:id", s.GetPrivateMessage)   // Get private
message by id

	// JWT Auth routes
	messagePath := basePath.Group("/messages")
	messagePath.Use(echojwt.WithConfig(newJWTConfig(config.JWTSecret)))

	messagePath.GET("/public", s.GetPublicMessagesList) // Get list of public
messages with text
	messagePath.GET("", s.GetUserMessagesList)          // Get list of user id
messages
	messagePath.POST("", s.CreateMessage)               // Create message
	messagePath.PUT("/:id", s.UpdateMessage)            // Update message
	messagePath.DELETE("/:id", s.DeleteMessage)         // Delete message by
hand if ttl not set

	// Connect to DB
	db, err := database.NewMainDB(config.MongoURL)
	if err != nil {
		return err
	}
	s.db = db

	// Connect to CacheDB
```

```
	cachedb, err := database.NewCacheDB(config.RedisURL)
	if err != nil {
		return err
	}
	s.cachedb = cachedb

	return nil
}

func (s *Server) Run() error {
	return s.e.Start(":" + config.Port)
}

func (s *Server) Shutdown(ctx context.Context) error {
	if err := s.e.Shutdown(ctx); err != nil {
		return err
	}
	if err := s.db.Shutdown(ctx); err != nil {
		return err
	}
	if err := s.cachedb.Shutdown(ctx); err != nil {
		return err
	}
	return nil
}

// server/auth.go
package server

import (
	"errors"

	"github.com/golang-jwt/jwt/v5"
	echojwt "github.com/labstack/echo-jwt/v4"
	"github.com/labstack/echo/v4"
)

type jwtCustomClaims struct {
	jwt.RegisteredClaims
}

func newJWTConfig(secret string) echojwt.Config {
	cfg := echojwt.Config{
		NewClaimsFunc: func(c echo.Context) jwt.Claims {
			return new(jwtCustomClaims)
		},
		SigningKey: []byte(secret),
	}
```

```
	return cfg
}

func newJWT(userId, secret string) (string, error) {
	claims := &jwtCustomClaims{
		jwt.RegisteredClaims{
			ID: userId,
		},
	}

	token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
	signedToken, err := token.SignedString([]byte(secret))
	if err != nil {
		return "", err
	}
	return signedToken, nil
}

func getUserIdFromJWT(c echo.Context) (string, error) {
	user, ok := c.Get("user").(*jwt.Token)
	if !ok {
		return "", errors.New("can't convert value from context to
*jwt.Token")
	}
	claims, ok := user.Claims.(*jwtCustomClaims)
	if !ok {
		return "", errors.New("can't convert jwt token to custom claims")
	}
	return claims.ID, nil
}

// server/user.go
package server

import (
	"fmt"
	"net/http"

	"github.com/arimatakao/deepenc/cmd/config"
	"github.com/arimatakao/deepenc/server/database"
	"github.com/labstack/echo/v4"
	"go.mongodb.org/mongo-driver/mongo"
	"golang.org/x/crypto/bcrypt"
)

func EmptyHandler(c echo.Context) error {
	return c.String(http.StatusOK, "empty handler")
```

```
}

type systemMessage struct {
	Message string `json:"message"`
}

func resp(text string) *systemMessage {
	return &systemMessage{
		Message: text,
	}
}

func (s *Server) SignUp(c echo.Context) error {
	u := new(database.User)

	if err := c.Bind(u); err != nil {
		return c.String(http.StatusBadRequest, "")
	}

	if u.Username == "" || u.Password == "" {
		return c.String(http.StatusBadRequest, "")
	}

	_, err := s.db.GetUser(u.Username)
	if err != mongo.ErrNoDocuments {
		return c.JSON(http.StatusConflict, resp("user is already exist"))
	} else if err != nil && err != mongo.ErrNoDocuments {
		c.Logger().Error(err)
		return c.String(http.StatusInternalServerError, "")
	}

	if len(u.Password) < 8 && len(u.Password) > 33 {
		return c.JSON(http.StatusBadRequest,
			resp("password should contain more than 7 symbols"))
	}

	hashedPassword, err := bcrypt.GenerateFromPassword([]byte(u.Password),
bcrypt.DefaultCost)
	if err != nil {
		c.Logger().Error(err)
		return c.String(http.StatusInternalServerError, "")
	}

	token, err := s.cachedb.AddUser(u.Username, string(hashedPassword))
	if err != nil {
		c.Logger().Error(err)
		return c.String(http.StatusInternalServerError, "")
	}
```

```
	messageText := fmt.Sprintf("confirm your username by route -
/api/verify/%s", token)
	return c.JSON(http.StatusOK, resp(messageText))
}

func (s *Server) VerifySignUp(c echo.Context) error {
	confirmToken := c.Param("token")
	if confirmToken == "" {
		return c.String(http.StatusBadRequest, "")
	}

	u, err := s.cachedb.GetUser(confirmToken)
	if err != nil {
		c.Logger().Error(err)
		return c.String(http.StatusInternalServerError, "")
	}

	if err = s.db.AddUser(u); err != nil {
		c.Logger().Error(err)
		c.String(http.StatusInternalServerError, "")
	}

	return c.String(http.StatusCreated, "")
}

func (s *Server) SignIn(c echo.Context) error {
	u := new(database.User)

	if err := c.Bind(u); err != nil {
		return c.String(http.StatusBadRequest, "")
	}

	if u.Username == "" || u.Password == "" {
		return c.String(http.StatusBadRequest, "")
	}

	userDocument, err := s.db.GetUser(u.Username)
	if err == mongo.ErrNoDocuments {
		return c.JSON(http.StatusNotFound, "")
	} else if err != nil {
		c.Logger().Error(err)
		c.String(http.StatusInternalServerError, "")
	}

	err = bcrypt.CompareHashAndPassword([]byte(userDocument.Password),
[]byte(u.Password))
	if err != nil {
```

```
			return c.String(http.StatusBadRequest, "")
		}

		token, err := newJWT(userDocument.Id.Hex(), config.JWTSecret)
		if err != nil {
			c.Logger().Error(err)
			return c.String(http.StatusInternalServerError, "")
		}

		return c.JSON(http.StatusOK, map[string]string{
			"token": token,
		})
}

// server/messages.go
package server

import (
		"errors"
		"net/http"

		"github.com/arimatakao/deepenc/cmd/config"
		"github.com/arimatakao/deepenc/server/database"
		"github.com/arimatakao/deepenc/utils"
		"github.com/labstack/echo/v4"
		"go.mongodb.org/mongo-driver/mongo"
		"golang.org/x/crypto/bcrypt"
)

const (
		MIN_CONTENT_SIZE  = 16
		MIN_PASSWORD_SIZE = 8

		MAX_CONTENT_SIZE = 2000
)

type Message struct {
		Content       string `json:"content"`
		IsPrivate     bool   `json:"is_private"`
		EncodingType  string `json:"encoding_type"`
		Password      string `json:"password"`
		OnlyOwnerView bool   `json:"only_owner_view"`
		IsAnon        bool   `json:"is_anon"`
		IsOneTime     bool   `json:"is_one_time"`
}

func (m Message) toDatabaseFormat(userId string) *database.Message {
		return &database.Message{
```

```
                OwnerId:       userId,
                Content:       m.Content,
                IsPrivate:     m.IsPrivate,
                EncodingType:  m.EncodingType,
                Password:      m.Password,
                OnlyOwnerView: m.OnlyOwnerView,
                IsAnon:        m.IsAnon,
                IsOneTime:     m.IsOneTime,
        }
}

func (m Message) isValid() bool {
        if len(m.Content) > MAX_CONTENT_SIZE ||
                m.Content == "" {
                return false
        }

        switch m.EncodingType {
        case "plaintext":
                if m.Password != "" {
                        return false
                }
        case "password":
                if len(m.Password) < MIN_PASSWORD_SIZE {
                        return false
                }
        case "internal":
                if len(m.Content) < MIN_CONTENT_SIZE {
                        return false
                }
        case "aes":
                if len(m.Content) < MIN_CONTENT_SIZE {
                        return false
                }
                if len(m.Password) < MIN_PASSWORD_SIZE {
                        return false
                }
        default:
                return false
        }

        return true
}

func (m *Message) formatToEncodingType() error {
        switch m.EncodingType {
        case "plaintext":
                m.Password = ""
```

```
	case "password":
		hashedPassword, err :=
bcrypt.GenerateFromPassword([]byte(m.Password),
			bcrypt.DefaultCost)
		if err != nil {
			return err
		}
		m.Password = string(hashedPassword)
		m.IsPrivate = true
	case "internal":
		encrypted, err := utils.EncryptAES256(config.AESInternalKey,
m.Content)
		if err != nil {
			return err
		}
		m.Content = encrypted
		m.Password = ""
		m.IsPrivate = true
	case "aes":
		encrypted, err := utils.EncryptAES256([]byte(m.Password),
m.Content)
		if err != nil {
			return err
		}
		m.Content = encrypted
		m.Password = ""
		m.IsPrivate = true
	default:
		return errors.New("unknown encoding_type")
	}
	return nil
}

func toOutputFormat(dbmsg database.MessageOut) *Message {
	return &Message{
		Content:       dbmsg.Content,
		IsPrivate:     dbmsg.IsPrivate,
		EncodingType:  dbmsg.EncodingType,
		Password:      dbmsg.Password,
		OnlyOwnerView: dbmsg.OnlyOwnerView,
		IsAnon:        dbmsg.IsAnon,
		IsOneTime:     dbmsg.IsOneTime,
	}
}

type InputPassword struct {
	Password string `json:"password"`
}
```

```
func (s *Server) CreateMessage(c echo.Context) error {
	userId, err := getUserIdFromJWT(c)
	if err != nil {
		return c.String(http.StatusBadRequest, "")
	}

	msg := new(Message)
	if err := c.Bind(msg); err != nil {
		return c.String(http.StatusBadRequest, "")
	}

	if !msg.isValid() {
		return c.String(http.StatusBadRequest, "")
	}

	if err := msg.formatToEncodingType(); err != nil {
		c.Logger().Error(err)
		return c.String(http.StatusInternalServerError, "")
	}

	mFormat := msg.toDatabaseFormat(userId)

	resultId, err := s.db.AddMessage(mFormat)
	if err != nil {
		c.Logger().Error(err)
		return c.String(http.StatusInternalServerError, "")
	}

	c.Logger().Info("added new message: " + resultId)

	return c.String(http.StatusCreated, "")
}

func (s *Server) GetPublicMessage(c echo.Context) error {
	msgId := c.Param("id")
	if msgId == "" {
		return c.String(http.StatusBadRequest, "")
	}

	msg, err := s.db.GetMessage(msgId)
	if err == mongo.ErrNoDocuments {
		return c.String(http.StatusNotFound, "")
	}
	if err != nil {
		c.Logger().Error(err)
		return c.String(http.StatusInternalServerError, "")
	}
```

```
	if msg.IsPrivate ||
		msg.Password != "" ||
		msg.EncodingType != "plaintext" ||
		msg.OnlyOwnerView {
		return c.String(http.StatusNotFound, "")
	}

	if msg.IsAnon {
		msg.OwnerId = ""
	}

	if msg.IsOneTime {
		if err = s.db.DeleteMessage(msgId); err != nil {
			c.Logger().Error(err)
			return c.JSON(http.StatusNotFound, "")
		}
	}

	return c.JSON(http.StatusOK, msg)
}

func (s *Server) GetUserMessagesList(c echo.Context) error {
	userId, err := getUserIdFromJWT(c)
	if err != nil {
		return c.String(http.StatusBadRequest, "")
	}

	messages, err := s.db.GetUserMessages(userId)
	if err == mongo.ErrNoDocuments {
		return c.String(http.StatusNotFound, "")
	}
	if err != nil {
		c.Logger().Error(err)
		return c.String(http.StatusInternalServerError, "")
	}

	return c.JSON(http.StatusOK, messages)
}

func (s *Server) GetPublicMessagesList(c echo.Context) error {
	messages, err := s.db.GetLastPublicMessages(10)
	if err == mongo.ErrNoDocuments {
		return c.String(http.StatusNotFound, "")
	}
	if err != nil {
		c.Logger().Error(err)
		return c.String(http.StatusInternalServerError, "")
```

```
	}
	for _, v := range messages {
		if v.IsAnon {
			v.OwnerId = ""
		}
	}

	return c.JSON(http.StatusOK, messages)
}

func (s *Server) UpdateMessage(c echo.Context) error {
	userId, err := getUserIdFromJWT(c)
	if err != nil {
		return c.String(http.StatusBadRequest, "")
	}

	msgId := c.Param("id")
	if msgId == "" {
		return c.String(http.StatusBadRequest, "")
	}

	msg := new(Message)
	if err := c.Bind(msg); err != nil {
		return c.String(http.StatusBadRequest, "")
	}

	if !msg.isValid() {
		return c.String(http.StatusBadRequest, "")
	}

	_, err = s.db.GetMessage(msgId)
	if err == mongo.ErrNoDocuments {
		return c.String(http.StatusNotFound, "")
	}
	if err != nil {
		c.Logger().Error(err)
		return c.String(http.StatusInternalServerError, "")
	}

	if err = msg.formatToEncodingType(); err != nil {
		c.Logger().Error(err)
		return c.String(http.StatusInternalServerError, "")
	}

	mFormat := msg.toDatabaseFormat(userId)

	err = s.db.UpdateMessage(msgId, mFormat)
	if err != nil {
```

```
                c.Logger().Error(err)
                return c.String(http.StatusBadRequest, "")
        }

        return c.String(http.StatusNoContent, "")
}

func (s *Server) DeleteMessage(c echo.Context) error {
        userId, err := getUserIdFromJWT(c)
        if err != nil {
                return c.String(http.StatusBadRequest, "")
        }

        msgId := c.Param("id")
        if msgId == "" {
                return c.String(http.StatusBadRequest, "")
        }

        msg, err := s.db.GetMessage(msgId)
        if err == mongo.ErrNoDocuments {
                return c.String(http.StatusNotFound, "")
        }
        if err != nil {
                c.Logger().Error(err)
                return c.String(http.StatusInternalServerError, "")
        }

        if msg.OwnerId != userId {
                return c.String(http.StatusBadRequest, "")
        }

        err = s.db.DeleteMessage(msgId)
        if err != nil {
                c.Logger().Error(err)
                return c.String(http.StatusInternalServerError, "")
        }

        return c.String(http.StatusNoContent, "")
}

func (s *Server) GetPrivateMessage(c echo.Context) error {
        msgId := c.Param("id")
        if msgId == "" {
                return c.String(http.StatusBadRequest, "")
        }

        input := new(InputPassword)
        if err := c.Bind(input); err != nil {
```

```
		return c.String(http.StatusBadRequest, "")
	}

	msg, err := s.db.GetMessage(msgId)
	if err == mongo.ErrNoDocuments {
		return c.String(http.StatusNotFound, "")
	}
	if err != nil {
		c.Logger().Error(err)
		return c.String(http.StatusInternalServerError, "")
	}

	if !msg.IsPrivate || msg.OnlyOwnerView {
		return c.String(http.StatusNotFound, "")
	}

	switch msg.EncodingType {
	case "password":
		err = bcrypt.CompareHashAndPassword([]byte(msg.Password),
[]byte(input.Password))
		if err != nil {
			return c.String(http.StatusNotFound, "")
		}
	case "internal":
		decrypted, err := utils.DecryptAES256(config.AESInternalKey,
msg.Content)
		if err != nil {
			c.Logger().Error(err)
			return c.String(http.StatusInternalServerError, "")
		}
		msg.Content = decrypted
	case "aes":
		decrypted, err := utils.DecryptAES256([]byte(input.Password),
msg.Content)
		if err != nil {
			c.Logger().Warn(err)
			return c.String(http.StatusNotFound, "")
		}
		msg.Content = decrypted
		msg.Password = ""
	}

	msgResp := toOutputFormat(msg)

	if msgResp.IsAnon {
		msg.OwnerId = ""
	}
```

```
	if msgResp.IsOneTime {
		if err = s.db.DeleteMessage(msgId); err != nil {
			c.Logger().Error(err)
			return c.JSON(http.StatusNotFound, "")
		}
	}

	return c.JSON(http.StatusOK, msgResp)
}

// server/database/interfaces.go
package database

import (
	"context"

	"go.mongodb.org/mongo-driver/bson/primitive"
)

type cachedUser struct {
	Username       string `redis:"username"`
	HashedPassword string `redis:"password"`
}

type User struct {
	Username string `json:"username"`
	Password string `json:"password"`
}

type UserOut struct {
	Id       primitive.ObjectID `bson:"_id"`
	Username string             `json:"username"`
	Password string             `json:"password"`
}

type UsersDB interface {
	AddUser(u *User) error
	GetUser(username string) (UserOut, error)
}

type Cacher interface {
	AddUser(username string, hashedPassword string) (token string, err error)
	GetUser(token string) (*User, error)
	Shutdown(context.Context) error
}

type Message struct {
	OwnerId       string `json:"owner_id" bson:"owner_id"`
```

```
	Content	string `json:"content" bson:"content"`
	IsPrivate	bool	`json:"is_private" bson:"is_private"`
	EncodingType	string `json:"encoding_type" bson:"encoding_type"`
	Password	string `json:"password" bson:"password"`
	OnlyOwnerView bool	`json:"only_owner_view" bson:"only_owner_view"`
	IsAnon	bool	`json:"is_anon" bson:"is_anon"`
	IsOneTime	bool	`json:"is_one_time" bson:"is_one_time"`
}

type MessageOut struct {
	Id	primitive.ObjectID `json:"id" bson:"_id"`
	OwnerId	string	`json:"owner_id,omitempty" bson:"owner_id"`
	Content	string	`json:"content" bson:"content"`
	IsPrivate	bool	`json:"is_private" bson:"is_private"`
	EncodingType	string	`json:"encoding_type" bson:"encoding_type"`
	Password	string	`json:"password,omitempty" bson:"password"`
	OnlyOwnerView bool	`json:"only_owner_view"
bson:"only_owner_view"`
	IsAnon	bool	`json:"is_anon" bson:"is_anon"`
	IsOneTime	bool	`json:"is_one_time" bson:"is_one_time"`
}

type MessagesOut []MessageOut

type MessagesDB interface {
	AddMessage(m *Message) (id string, err error)
	GetMessage(id string) (MessageOut, error)
	GetLastPublicMessages(limit int) (MessagesOut, error)
	GetUserMessages(ownerId string) (MessagesOut, error)
	UpdateMessage(id string, m *Message) error
	DeleteMessage(id string) error
}

type Storager interface {
	UsersDB
	MessagesDB
	Shutdown(context.Context) error
}

// server/database/mongo.go
package database

import (
	"context"

	"go.mongodb.org/mongo-driver/bson/primitive"
)
```

```
type cachedUser struct {
	Username       string `redis:"username"`
	HashedPassword string `redis:"password"`
}

type User struct {
	Username string `json:"username"`
	Password string `json:"password"`
}

type UserOut struct {
	Id       primitive.ObjectID `bson:"_id"`
	Username string             `json:"username"`
	Password string             `json:"password"`
}

type UsersDB interface {
	AddUser(u *User) error
	GetUser(username string) (UserOut, error)
}

type Cacher interface {
	AddUser(username string, hashedPassword string) (token string, err error)
	GetUser(token string) (*User, error)
	Shutdown(context.Context) error
}

type Message struct {
	OwnerId      string `json:"owner_id" bson:"owner_id"`
	Content      string `json:"content" bson:"content"`
	IsPrivate    bool   `json:"is_private" bson:"is_private"`
	EncodingType  string `json:"encoding_type" bson:"encoding_type"`
	Password     string `json:"password" bson:"password"`
	OnlyOwnerView bool   `json:"only_owner_view" bson:"only_owner_view"`
	IsAnon       bool   `json:"is_anon" bson:"is_anon"`
	IsOneTime    bool   `json:"is_one_time" bson:"is_one_time"`
}

type MessageOut struct {
	Id           primitive.ObjectID `json:"id" bson:"_id"`
	OwnerId      string             `json:"owner_id,omitempty" bson:"owner_id"`
	Content      string             `json:"content" bson:"content"`
	IsPrivate    bool               `json:"is_private" bson:"is_private"`
	EncodingType  string             `json:"encoding_type" bson:"encoding_type"`
	Password     string             `json:"password,omitempty" bson:"password"`
	OnlyOwnerView bool               `json:"only_owner_view"
bson:"only_owner_view"`
	IsAnon       bool               `json:"is_anon" bson:"is_anon"`
```

```
	IsOneTime	bool		`json:"is_one_time" bson:"is_one_time"`
}

type MessagesOut []MessageOut

type MessagesDB interface {
	AddMessage(m *Message) (id string, err error)
	GetMessage(id string) (MessageOut, error)
	GetLastPublicMessages(limit int) (MessagesOut, error)
	GetUserMessages(ownerId string) (MessagesOut, error)
	UpdateMessage(id string, m *Message) error
	DeleteMessage(id string) error
}

type Storager interface {
	UsersDB
	MessagesDB
	Shutdown(context.Context) error
}

// server/database/redis.go
package database

import (
	"context"
	"crypto/sha1"
	"errors"
	"fmt"

	"github.com/redis/go-redis/v9"
)

type CacheDB struct {
	r *redis.Client
}

func NewCacheDB(connectionUrl string) (*CacheDB, error) {
	opt, err := redis.ParseURL(connectionUrl)
	if err != nil {
		return nil, err
	}

	r := redis.NewClient(opt)

	err = r.Ping(context.Background()).Err()
	if err != nil {
		return nil, err
	}
```

```
	return &CacheDB{
		r: r,
	}, nil
}

func (c CacheDB) Shutdown(context.Context) error {
	return c.r.Close()
}

func (c CacheDB) AddUser(username string, hashedPassword string) (string,
error) {
	hasher := sha1.New()
	hashedUsername := fmt.Sprintf("%x", hasher.Sum([]byte(username)))

	isExist, err := c.r.Exists(context.Background(), hashedUsername).Result()
	if err != nil {
		return "", err
	}

	if isExist == 1 {
		return "", errors.New("user is already exist in cache")
	}

	cUser := cachedUser{
		Username:       username,
		HashedPassword: hashedPassword,
	}
	err = c.r.HSet(context.Background(), hashedUsername, cUser).Err()
	if err != nil {
		return "", err
	}

	return hashedUsername, nil
}

func (c CacheDB) GetUser(token string) (*User, error) {
	result, err := c.r.HGetAll(context.Background(), token).Result()
	if err != nil {
		return nil, err
	}
	username, ok := result["username"]
	if !ok {
		return nil, errors.New("username field in hset not exist")
	}
	hashedPassword, ok := result["password"]
	if !ok {
		return nil, errors.New("password field in hset not exist")
```

```
	}

	err = c.r.Del(context.Background(), token).Err()
	if err != nil {
		return nil, err
	}

	return &User{
		Username: username,
		Password: hashedPassword,
	}, nil
}

// config-example.yaml
port: 1234
mongodb_url: "mongodb://root:example@mongo:27017"
redis_url: "redis://default:pass@redis:6379/0"
jwt_secret: "supersecretexample"
aes_internal_key: "aesinternalkey"

// Dockerfile
FROM golang:1.22.1-alpine3.19 as builder

ARG CGO_ENABLED=0
WORKDIR /app

COPY . .
RUN go mod tidy
RUN go build -o ./deepenc main.go
RUN touch config.yaml

FROM scratch
COPY --from=builder /app/deepenc /bin/deepenc
COPY --from=builder /app/config.yaml /bin/config.yaml


EXPOSE 1234

ENTRYPOINT ["/bin/deepenc"]

CMD ["-config ./config.yaml"]


// docker-compose-db-only.yaml
version: '3.9'

services:
```

```
  redis:
    image: redis:7.2.4-alpine
    restart: always
    hostname: redis
    ports:
      - "6379:6379"
    environment:
      - REDIS_USERNAME=default
      - REDIS_PASSWORD=pass
      - REDIS_PORT=6379
      - REDIS_DATABASES=0

  mongo:
    image: mongo:7.0.6
    restart: always
    ports:
      - 27017:27017
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: example
    # Persistence storage
    # volumes:
    #   - "./db/data:/db/data"

  mongo-express:
    image: mongo-express:1.0.2-20-alpine3.19
    restart: always
    ports:
      - 8081:8081
    environment:
      ME_CONFIG_MONGODB_ADMINUSERNAME: root
      ME_CONFIG_MONGODB_ADMINPASSWORD: example
      ME_CONFIG_MONGODB_URL: mongodb://root:example@mongo:27017/
    depends_on:
      - mongo

// Makefile
run:
        go run main.go -config ./config.yaml
build:
        mkdir bin
        go build -o ./bin/deepenc main.go
test:
        go test -v ./...
clear:
        rm -rf ./bin

// go.mod
```

```
module github.com/arimatakao/deepenc

go 1.22.1

require (
	github.com/golang-jwt/jwt/v5 v5.2.1
	github.com/labstack/echo-jwt/v4 v4.2.0
	github.com/labstack/echo/v4 v4.11.4
	github.com/labstack/gommon v0.4.2
	github.com/redis/go-redis/v9 v9.5.1
	github.com/stretchr/testify v1.9.0
	go.mongodb.org/mongo-driver v1.14.0
	golang.org/x/crypto v0.18.0
	gopkg.in/yaml.v3 v3.0.1
)

require (
	github.com/cespare/xxhash/v2 v2.2.0 // indirect
	github.com/davecgh/go-spew v1.1.1 // indirect
	github.com/dgryski/go-rendezvous v0.0.0-20200823014737-9f7001d12a5f //
indirect
	github.com/golang-jwt/jwt v3.2.2+incompatible // indirect
	github.com/golang/snappy v0.0.4 // indirect
	github.com/google/go-cmp v0.6.0 // indirect
	github.com/klauspost/compress v1.17.4 // indirect
	github.com/mattn/go-colorable v0.1.13 // indirect
	github.com/mattn/go-isatty v0.0.20 // indirect
	github.com/montanaflynn/stats v0.0.0-20171201202039-1bf9dbcd8cbe //
indirect
	github.com/pmezard/go-difflib v1.0.0 // indirect
	github.com/valyala/bytebufferpool v1.0.0 // indirect
	github.com/valyala/fasttemplate v1.2.2 // indirect
	github.com/xdg-go/pbkdf2 v1.0.0 // indirect
	github.com/xdg-go/scram v1.1.2 // indirect
	github.com/xdg-go/stringprep v1.0.4 // indirect
	github.com/youmark/pkcs8 v0.0.0-20181117223130-1be2e3e5546d //
indirect
	golang.org/x/net v0.20.0 // indirect
	golang.org/x/sync v0.1.0 // indirect
	golang.org/x/sys v0.16.0 // indirect
	golang.org/x/text v0.14.0 // indirect
	golang.org/x/time v0.5.0 // indirect
)

// go.sum
github.com/bsm/ginkgo/v2 v2.12.0
h1:Ny8MWAHyOepLGlLKYmXG4IEkioBysk6GpaRTLC8zwWs=
```

github.com/bsm/ginkgo/v2 v2.12.0/go.mod
h1:SwYbGRRDovPVboqFv0tPTcG1sN61LM1Z4ARdbAV9g4c=
github.com/bsm/gomega v1.27.10
h1:yeMWxP2pV2fG3FgAODIY8EiRE3dy0aeFYt4l7wh6yKA=
github.com/bsm/gomega v1.27.10/go.mod
h1:JyEr/xRbxbtgWNi8tIEVPUYZ5Dzef52k01W3YH0H+O0=
github.com/cespare/xxhash/v2 v2.2.0
h1:DC2CZ1Ep5Y4k3ZQ899DldepgrayRUGE6BBZ/cd9Cj44=
github.com/cespare/xxhash/v2 v2.2.0/go.mod
h1:VGX0DQ3Q6kWi7AoAeZDth3/j3BFtOZR5XLFGgcrjCOs=
github.com/davecgh/go-spew v1.1.1
h1:vj9j/u1bqnvCEfJOwUhtlOARqs3+rkHYY13jYWTU97c=
github.com/davecgh/go-spew v1.1.1/go.mod
h1:J7Y8YcW2NihsgmVo/mv3lAwl/skON4iLHjSsI+c5H38=
github.com/dgryski/go-rendezvous v0.0.0-20200823014737-9f7001d12a5f
h1:lO4WD4F/rVNCu3HqELle0jiPLLBs70cWOduZpkS1E78=
github.com/dgryski/go-rendezvous v0.0.0-20200823014737-9f7001d12a5f/go.mod
h1:cuUVRXasLTGF7a8hSLbxyZXjz+1KgoB3wDUb6vlszIc=
github.com/golang-jwt/jwt v3.2.2+incompatible
h1:IfV12K8xAKAnZqdXVzCZ+TOjboZ2keLg81eXfW3O+oY=
github.com/golang-jwt/jwt v3.2.2+incompatible/go.mod
h1:8pz2t5EyA70fFQQSrl6XZXzqecmYZeUEB8OUGHkxJ+I=
github.com/golang-jwt/jwt/v5 v5.2.1
h1:OuVbFODueb089Lh128TAcimifWaLhJwVflnrgM17wHk=
github.com/golang-jwt/jwt/v5 v5.2.1/go.mod
h1:pqrtFR0X4osieyHYxtmOUWsAWrfe1Q5UVIyoH402zdk=
github.com/golang/snappy v0.0.4
h1:yAGX7huGHXlcLOEtBnF4w7FQwA26wojNCwOYAEhLjQM=
github.com/golang/snappy v0.0.4/go.mod
h1:/XxbfmMg8lxefKM7IXC3fBNl/7bRcc72aCRzEWrmP2Q=
github.com/google/go-cmp v0.6.0
h1:ofyhxvXcZhMsU5ulbFiLKl/XBFqE1GSq7atu8tAmTRI=
github.com/google/go-cmp v0.6.0/go.mod
h1:17dUlkBOakJ0+DkrSSNjCkIjxS6bF9zb3elmeNGIjoY=
github.com/klauspost/compress v1.17.4
h1:Ej5ixsIri7BrIjBkRZLTo6ghwrEtHFk7ijlczPW4fZ4=
github.com/klauspost/compress v1.17.4/go.mod
h1:/dCuZOvVtNoHsyb+cuJD3itjs3NbnF6KH9zAO4BDxPM=
github.com/labstack/echo-jwt/v4 v4.2.0
h1:odSISV9JgcSCuhgQSV/6Io3i7nUmfM/QkBeR5GVJj5c=
github.com/labstack/echo-jwt/v4 v4.2.0/go.mod
h1:MA2RqdXdEn4/uEglx0HcUOgQSyBaTh5JcaHIan3biwU=
github.com/labstack/echo/v4 v4.11.4
h1:vDZmA+qNeh1pd/cCkEicDMrjtrnMGQ1QFI9gWN1zGq8=
github.com/labstack/echo/v4 v4.11.4/go.mod
h1:noh7EvLwqDsmh/X/HWKPUl1AjzJrhyptRyEbQJfxen8=
github.com/labstack/gommon v0.4.2
h1:F8qTUNXgG1+6WQmqoUWnz8WiEU60mXVVw0P4ht1WRA0=

github.com/labstack/gommon v0.4.2/go.mod
h1:QlUFxVM+SNXhDL/Z7YhocGIBYOiwB0mXm1+1bAPHPyU=
github.com/mattn/go-colorable v0.1.13
h1:fFA4WZxdEF4tXPZVKMLwD8oUnCTTo08duU7wxecdEvA=
github.com/mattn/go-colorable v0.1.13/go.mod
h1:7S9/ev0klgBDR4GtXTXX8a3vIGJpMovkB8vQcUbaXHg=
github.com/mattn/go-isatty v0.0.16/go.mod
h1:kYGgaQfpe5nmfYZH+SKPsOc2e4SrIfOl2e/yFXSvRLM=
github.com/mattn/go-isatty v0.0.20
h1:xfD0iDuEKnDkl03q4limB+vH+GxLEtL/jb4xVJSWWEY=
github.com/mattn/go-isatty v0.0.20/go.mod
h1:W+V8PltTTMOvKvAeJH7IuucS94S2C6jfK/D7dTCTo3Y=
github.com/montanaflynn/stats v0.0.0-20171201202039-1bf9dbcd8cbe
h1:iruDEfMl2E6fbMZ9s0scYfZQ84/6SPL6zC8ACM2oIL0=
github.com/montanaflynn/stats v0.0.0-20171201202039-1bf9dbcd8cbe/go.mod
h1:wL8QJuTMNUDYhXwkmfOly8iTdp5TEcJFWZD2D7SIkUc=
github.com/pmezard/go-difflib v1.0.0
h1:4DBwDE0NGyQoBHbLQYPwSUPoCMWR5BEzIk/f1lZbAQM=
github.com/pmezard/go-difflib v1.0.0/go.mod
h1:iKH77koFhYxTK1pcRnkKkqfTogsbg7gZNVY4sRDYZ/4=
github.com/redis/go-redis/v9 v9.5.1
h1:H1X4D3yHPaYrkL5X06Wh6xNVM/pX0Ft4RV0vMGvLBh8=
github.com/redis/go-redis/v9 v9.5.1/go.mod
h1:hdY0cQFCN4fnSYT6TkisLufl/4W5UIXyv0b/CLO2V2M=
github.com/stretchr/testify v1.9.0
h1:HtqpIVDClZ4nwg75+f6Lvsy/wHu+3BoSGCbBAcpTsTg=
github.com/stretchr/testify v1.9.0/go.mod
h1:r2ic/lqez/lEtzL7wO/rwa5dbSLXVDPFyf8C91i36aY=
github.com/valyala/bytebufferpool v1.0.0
h1:GqA5TC/0021Y/b9FG4Oi9Mr3q7XYx6KllzawFIhcdPw=
github.com/valyala/bytebufferpool v1.0.0/go.mod
h1:6bBcMArwyJ5K/AmCkWv1jt77kVWyCJ6HpOuEn7z0Csc=
github.com/valyala/fasttemplate v1.2.2
h1:lxLXG0uE3Qnshl9QyaK6XJxMXlQZELvChBOCmQD0Loo=
github.com/valyala/fasttemplate v1.2.2/go.mod
h1:KHLXt3tVN2HBp8eijSv/kGJopbvo7S+qRAEEKiv+SiQ=
github.com/xdg-go/pbkdf2 v1.0.0
h1:Su7DPu48wXMwC3bs7MCNG+z4FhcyEuz5dlvchbq0B0c=
github.com/xdg-go/pbkdf2 v1.0.0/go.mod
h1:jrpuAogTd400dnrH08LKmI/xc1MbPOebTwRqcT5RDeI=
github.com/xdg-go/scram v1.1.2
h1:FHX5I5B4i4hKRVRBCFRxq1iQRej7WO3hhBuJf+UUySY=
github.com/xdg-go/scram v1.1.2/go.mod
h1:RT/sEzTbU5y00aCK8UOx6R7YryM0iF1N2MOmC3kKLN4=
github.com/xdg-go/stringprep v1.0.4
h1:XLI/Ng3O1Atzq0oBs3TWm+5ZVgkq2aqdlvP9JtoZ6c8=
github.com/xdg-go/stringprep v1.0.4/go.mod
h1:mPGuuIYwz7CmR2bT9j4GbQqutWS1zV24gijq1dTyGkM=

github.com/youmark/pkcs8 v0.0.0-20181117223130-1be2e3e5546d
h1:splanxYIlg+5LfHAM6xpdFEAYOk8iySO56hMFq6uLyA=
github.com/youmark/pkcs8 v0.0.0-20181117223130-1be2e3e5546d/go.mod
h1:rHwXgn7JulP+udvsHwJoVG1YGAP6VLg4y9I5dyZdqmA=
github.com/yuin/goldmark v1.4.13/go.mod
h1:6yULJ656Px+3vBD8DxQVa3kxgyrAnzto9xy5taEt/CY=
go.mongodb.org/mongo-driver v1.14.0
h1:P98w8egYRjYe3XDjxhYJagTokP/H6HzlsnojRgZRd80=
go.mongodb.org/mongo-driver v1.14.0/go.mod
h1:Vzb0Mk/pa7e6cWw85R4F/endUC3u0U9jGcNU603k65c=
golang.org/x/crypto v0.0.0-20190308221718-c2843e01d9a2/go.mod
h1:djNgcEr1/C05ACkg1iLfiJU5Ep61QUkGW8qpdssI0+w=
golang.org/x/crypto v0.0.0-20210921155107-089bfa567519/go.mod
h1:GvvjBRRGRdwPK5ydBHafDWAxML/pGHZbMvKqRZ5+Abc=
golang.org/x/crypto v0.18.0
h1:PGVlW0xEltQnzFZ55hkuX5+KLyrMYhHld1YHO4AKcdc=
golang.org/x/crypto v0.18.0/go.mod
h1:R0j02AL6hcrfOiy9T4ZYp/rcWeMxM3L6QYxlOuEG1mg=
golang.org/x/mod v0.6.0-dev.0.20220419223038-86c51ed26bb4/go.mod
h1:jJ57K6gSWd91VN4djpZkiMVwK6gcyfeH4XE8wZrZaV4=
golang.org/x/net v0.0.0-20190620200207-3b0461eec859/go.mod
h1:z5CRVTTTmAJ677TzLLGU+0bjPO0LkuOLi4/5GtJWs/s=
golang.org/x/net v0.0.0-20210226172049-e18ecbb05110/go.mod
h1:m0MpNAwzfU5UDzcl9v0D8zg8gWTRqZa9RBIspLL5mdg=
golang.org/x/net v0.0.0-20220722155237-a158d28d115b/go.mod
h1:XRhObCWvk6IyKnWLug+ECip1KBveYUHfp+8e9klMJ9c=
golang.org/x/net v0.20.0
h1:aCL9BSgETF1k+blQaYUBx9hJ9LOGP3gAVemcZlf1Kpo=
golang.org/x/net v0.20.0/go.mod
h1:z8BVo6PvndSri0LbOE3hAn0apkU+1YvI6E70E9jsnvY=
golang.org/x/sync v0.0.0-20190423024810-112230192c58/go.mod
h1:RxMgew5VJxzue5/jJTE5uejpjVlOe/izrB70Jof72aM=
golang.org/x/sync v0.0.0-20220722155255-886fb9371eb4/go.mod
h1:RxMgew5VJxzue5/jJTE5uejpjVlOe/izrB70Jof72aM=
golang.org/x/sync v0.1.0
h1:wsuoTGHzEhffawBOhz5CYhcrV4IdKZbEyZjBMuTp12o=
golang.org/x/sync v0.1.0/go.mod
h1:RxMgew5VJxzue5/jJTE5uejpjVlOe/izrB70Jof72aM=
golang.org/x/sys v0.0.0-20190215142949-d0b11bdaac8a/go.mod
h1:STP8DvDyc/dI5b8T5hshtkjS+E42TnysNCUPdjciGhY=
golang.org/x/sys v0.0.0-20201119102817-f84b799fce68/go.mod
h1:h1NjWce9XRLGQEsW7wpKNCjG9DtNlClVuFLEZdDNbEs=
golang.org/x/sys v0.0.0-20210615035016-665e8c7367d1/go.mod
h1:oPkhp1MJrh7nUepCBck5+mAzfO9JrbApNNgaTdGDITg=
golang.org/x/sys v0.0.0-20220520151302-bc2c85ada10a/go.mod
h1:oPkhp1MJrh7nUepCBck5+mAzfO9JrbApNNgaTdGDITg=
golang.org/x/sys v0.0.0-20220722155257-8c9f86f7a55f/go.mod
h1:oPkhp1MJrh7nUepCBck5+mAzfO9JrbApNNgaTdGDITg=

golang.org/x/sys v0.0.0-20220811171246-fbc7d0a398ab/go.mod
h1:oPkhp1MJrh7nUepCBck5+mAzfO9JrbApNNgaTdGDITg=
golang.org/x/sys v0.6.0/go.mod
h1:oPkhp1MJrh7nUepCBck5+mAzfO9JrbApNNgaTdGDITg=
golang.org/x/sys v0.16.0
h1:xWw16ngr6ZMtmxDyKyIgsE93KNKz5HKmMa3b8ALHidU=
golang.org/x/sys v0.16.0/go.mod
h1:/VUhepiaJMQUp4+oa/7Zr1D23ma6VTLIYjOOTFZPUcA=
golang.org/x/term v0.0.0-20201126162022-7de9c90e9dd1/go.mod
h1:bj7SfCRtBDWHUb9snDiAeCFNEtKQo2Wmx5Cou7ajbmo=
golang.org/x/term v0.0.0-20210927222741-03fcf44c2211/go.mod
h1:jbD1KX2456YbFQfuXm/mYQcufACuNUgVhRMnK/tPxf8=
golang.org/x/text v0.3.0/go.mod
h1:NqM8EUOU14njkJ3fqMW+pc6Ldnwhi/IjpwHt7yyuwOQ=
golang.org/x/text v0.3.3/go.mod
h1:5Zoc/QRtKVWzQhOtBMvqHzDpF6irO9z98xDceosuGiQ=
golang.org/x/text v0.3.7/go.mod
h1:u+2+/6zg+i71rQMx5EYifcz6MCKuco9NR6JIITiCfzQ=
golang.org/x/text v0.3.8/go.mod
h1:E6s5w1FMmriuDzIBO73fBruAKo1PCIq6d2Q6DHfQ8WQ=
golang.org/x/text v0.14.0
h1:ScX5w1eTa3QqT8oi6+ziP7dTV1S2+ALU0bI+0zXKWiQ=
golang.org/x/text v0.14.0/go.mod
h1:18ZOQIKpY8NJVqYksKHtTdi31H5itFRjB5/qKTNYzSU=
golang.org/x/time v0.5.0
h1:o7cqy6amK/52YcAKIPlM3a+Fpj35zvRj2TP+e1xFSfk=
golang.org/x/time v0.5.0/go.mod
h1:3BpzKBy/shNhVucY/MWOyx10tF3SFh9QdLuxbVysPQM=
golang.org/x/tools v0.0.0-20180917221912-90fa682c2a6e/go.mod
h1:n7NCudcB/nEzxVGmLbDWY5pfWTLqBcC2KZ6jyYvM4mQ=
golang.org/x/tools v0.0.0-20191119224855-298f0cb1881e/go.mod
h1:b+2E5dAYhXwXZwtnZ6UAqBI28+e2cm9otk0dWdXHAEo=
golang.org/x/tools v0.1.12/go.mod
h1:hNGJHUnrk76NpqgfD5Aqm5Crs+Hm0VOH/i9J2+nxYbc=
golang.org/x/xerrors v0.0.0-20190717185122-a985d3407aa7/go.mod
h1:I/5z698sn9Ka8TeJc9MKroUUfqBBauWjQqLJ2OPfmY0=
gopkg.in/check.v1 v0.0.0-20161208181325-20d25e280405
h1:yhCVgyC4o1eVCa2tZl7eS0r+SDo693bJlVdllGtEeKM=
gopkg.in/check.v1 v0.0.0-20161208181325-20d25e280405/go.mod
h1:Co6ibVJAznAaIkqp8huTwlJQCZ016jof/cbN4VW5Yz0=
gopkg.in/yaml.v3 v3.0.1
h1:fxVm/GzAzEWqLHuvctI91KS9hhNmmWOoWu0XTYJS7CA=
gopkg.in/yaml.v3 v3.0.1/go.mod
h1:K4uyk7z7BCEPqu6E+C64Yfv1cQ7kz7rIZviUmN+EgEM=

Додаток Б

```
# config.yaml
port: 1234
mongodb_url: "mongodb://root:example@localhost:27017"
redis_url: "redis://default:pass@localhost:6379/0"
jwt_secret: "supersecretexample"
aes_internal_key: "aesinternalkey"
```

Додаток В

```
# docker-compose.yaml
version: '3.9'

services:

  redis:
    image: redis:7.2.4-alpine
    restart: always
    hostname: redis
    ports:
      - "6379:6379"
    environment:
      - REDIS_USERNAME=default
      - REDIS_PASSWORD=pass
      - REDIS_PORT=6379
      - REDIS_DATABASES=0

  mongo:
    image: mongo:7.0.6
    restart: always
    ports:
      - 27017:27017
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: example
    # Persistence storage
    # volumes:
    #   - "./db/data:/db/data"

  mongo-express:
    image: mongo-express:1.0.2-20-alpine3.19
    restart: always
    ports:
      - 8081:8081
    environment:
      ME_CONFIG_MONGODB_ADMINUSERNAME: root
      ME_CONFIG_MONGODB_ADMINPASSWORD: example
      ME_CONFIG_MONGODB_URL: mongodb://root:example@mongo:27017/
```

```
  depends_on:
    - mongo

deepenc:
  container_name: deepenc
  build:
    context: .
    dockerfile: Dockerfile
  restart: always
  ports:
    - "1234:1234"
  depends_on:
    - redis
    - mongo
    - mongo-express
  volumes:
     - "./config-example.yaml:/bin/config.yaml"
  command: "-config /bin/config.yaml"
```

Додаток Г

UNICHECK
by Turnitin

Ім'я користувача:
Інформаційних систем в економіці Шкуратовська Те...

ID перевірки:
1016317170

Дата перевірки:
04.06.2024 02:57:05 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
04.06.2024 07:11:50 EEST

ID користувача:
100005745

Назва документа: Урденко Д

Кількість сторінок: 38 Кількість слів: 5659 Кількість символів: 47179 Розмір файлу: 1.26 MB ID файлу: 1016114610

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

7.28%
Схожість

Найбільша схожість: 2.62% з джерелом з Бібліотеки (ID файлу: 1016112522)

3.6% Джерела з Інтернету | 183 ..... Сторінка 40

6.96% Джерела з Бібліотеки | 358 ..... Сторінка 41

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування | 11 сторінок

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА
Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці

ОСВІТНЬО ПРОФЕСІЙНА ПРОГРАМА «КОМП'ЮТЕРНІ НАУКИ»
галузь знань 12 «Інформаційні технології»
спеціальність 122 «Комп'ютерні науки»

Форма навчання: денна

КВАЛІФІКАЦІЙНИЙ БАКАЛАВРСЬКИЙ ПРОЄКТ

на тему

«ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ ПЕРЕДАЧІ ЗАШИФРОВАНИХ ПОВІДОМЛЕНЬ»

здобувача Урденка Дениса Олександровича

Науковий керівник:
д. е. н., професор
Мозгаллі О.П.

Робота допущена до захисту перед екзаменаційною комісією з атестації здобувачів вищої освіти
завідувач кафедри:
к.е.н., доцент.
_____ Тішков Б.О.

Київ 2024