МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНА ТЕХНОЛОГІЯ УПРАВЛІННЯ ТА РОЗРОБКИ ПРОЄКТІВ

Здобувач освіти гр. ІН.мз-11с Юлія ОСМОЛОВСЬКА

Науковий керівник, Ольга ШУТИЛЄВА
асистент кафедри комп'ютерних наук, к.ф.-м.н.

В.о. завідувача кафедри, Ігор ШЕЛЕХОВ
доцент кафедри комп'ютерних наук, к.т.н., доцент

Суми 2022

Сумський державний університет

(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«122 – Комп'ютерні науки»*

Затверджую:

в.о. зав. кафедрою _____

_____

"_____" _____2022 р.

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТОВІ

Осмоловській Юлії Сергіївні

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) *Інформаційне та програмне забезпечення передачі змінних середовища для управління та розробки проєктів*

затверджую наказом по інституту від "___" _____2022 р. № _____

2. Термін здачі студентом закінченого проєкту (роботи)_____

3. Вхідні данні до проєкту (роботи) _____

_____

_____

_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Літературний огляд та постановка задачі 2) Моделювання технологій сервісу для передачі зашифрованих даних та їх аналіз взаємодії; 3) Створення схеми бази даних веб додатку та розробка графічного інтерфейсу 4) Тестування інтерфейсу.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

_____

_____

_____

_____

_____

6. Консультанти до проєкту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____

(підпис)

Завдання прийняв до виконання _____

(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проєкту (роботи)	Примітка
1.	Аналіз поставленої проблеми предметної області, постановка задачі		
2.	Огляд стека технологій, протоколів, які плануються використовуватися		
3.	Розробка веб-системи з застосуванням досліджуваних технологій		
4.	Аналіз та тестування отриманої системи		
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент - дипломник _____

(підпис)

Керівник проєкту _____

(підпис)

РЕФЕРАТ

Записка: 67 стр, 14 рис, 11 додатків, 15 джерел

Об’єкт дослідження – процес проєктування інформаційного та програмного забезпечення передачі змінних середовища для управління та розробки проєктів

Мета роботи – розробка та програмна реалізація веб-додатку з шифруванням для комунікації між розробниками в рамках однієї команди.

Методи дослідження – методи проєктування веб-додатку для комунікації між багатьма користувачами, методи шифрування та дешифрування з симетричний ключем.

Результати – розроблено, спроєктовано та програмно реалізовано веб-додаток для передачі змінних середовища, який містить, авторизацію, автентифікацію та розділення користувачів на два рівні доступу, а також шифруванням та дешифруванням всіх повідомлень. В роботі було проведено аналіз присутніх на ринку аналогів, для обміну даними під час розробки програмного продукту, які є конфіденційними. Опираючись на проаналізовані дані, було досліджено розширений стандарт шифрування з режимом лічильника Галуа та вибрано його для основи використання побудови веб-додатку розроблений мовою програмування Ruby.

RUBY, POSTGRESQL, OPENSSL, ЛІЧИЛЬНИК З АВТЕНТИФІКАЦІЄЮ ГАЛУА, ШИФРУВАННЯ, ДЕШИФРУВАННЯ

ЗМІСТ

ВСТУП

Розробка програмного забезпечення частіше за все це командна робота. А отже вона вимагає злагодженості, швидкої комунікації, негайної реакції на труднощі, що виникають. Часто одна команда, може працювати над багатьма проєктами, утворюються підкоманди для вирішення нових задач, додаються нові працівники, хтось звільняється, хтось додається до команди. Звичайно, якщо розробники працюють в великих компаніях, велику частину зобов'язань на себе бере менеджерський відділ. Та це не означає, що маленькі підприємства на пару десятків людей, не мають тих же проблем що й великі.

Під час роботи над будь-яким програмним забезпеченням розробники використовують різні ключі доступа, паролі, секрети від додаткових фреймворків тощо. Важливість збереження всіх цих речей неймовірно велика. Відповідальним ставленням, до конфіденційної інформації забезпечується безпека даних клієнтів, працівників, користувачів платформи над якою працюють програмісти.

Під час розробки програмного продукту, девелопери безліч разів використовують змінні середовища. На великому проєкті їх можуть бути сотні. Вони використовуються для тестування, інтеграції з іншими програмними продуктами, деплоймента. Одна й та ж змінна на різних етапах проєктування, може мати різні ключі.

Метою магістерської роботи, стала реалізація дуже простої в користуванні платформи для передачі між розробниками, в шифрованому вигляді змінних середовища зокрема, та спілкування для рішення робочих задач. Також платформу можна використовувати, як простий месенжер, що дозволяє швидко перемикатись з однієї розмови до іншої, і мігрувати між компаніями та не хвилюватись, що будь-які дані можуть бути доступні користувачам інших компаній. Один користувач в рамках різних компаній може мати різні ролі, а отже й різні права.

В рамках магістерської роботи прийнято рішення реалізувати даний програмний продукт, використовуючи мову програмування Ruby, фреймворк Ruby on Rails, базу даних PostgreSQL, шифрування AES-GCM з використанням OpenSSL бібліотеки.

1. АНАЛІЗ ПРОБЛЕМИ І ПОСТАНОВКА ЗАДАЧІ

1.1 Секюрність розробки проєктів програмного забезпечення

Автентифікація є однією з головних і більш тривожних частин процесу розробки програмного забезпечення.

Автентифікація — це процес визначення того, чи є хтось або щось насправді тим або ким, про що він говорить [1]. Технологія автентифікації забезпечує контроль доступу для систем, перевіряючи, чи збігаються облікові дані користувача з обліковими даними в базі даних авторизованих користувачів або на сервері автентифікації даних. При цьому автентифікація забезпечує безпечні системи, безпечні процеси та безпеку корпоративної інформації. Автентифікація дозволяє організаціям підтримувати безпеку своїх мереж, дозволяючи лише автентифікованим користувачам або процесам отримувати доступ до їхніх захищених ресурсів. Це може включати комп'ютерні системи, мережі, бази даних, веб-сайти та інші мережеві програми чи служби.

Під час автентифікації облікові дані, надані користувачем, порівнюються з даними, збереженими в базі даних авторизованих користувачів на сервері локальної операційної системи або через сервер автентифікації . Якщо введені облікові дані збігаються з даними у файлі та автентифікований об'єкт має право використовувати ресурс, користувачеві надається доступ. Дозволи користувача визначають, до яких ресурсів користувач отримує доступ, а також будь-які інші пов'язані з ним права доступу, наприклад, протягом яких годин користувач може отримати доступ до ресурсу та яку частину ресурсу йому дозволено використовувати [2].

Розробник, несете відповідальність за те, щоб конфіденційні дані користувача були в безпеці та не були вразливими до будь-яких атак, які спричинили б їх злам і потрапляння під чужі руки. Якщо говорити, про роботу в фінансовій сфері, медичній, державній це питання є особливо критичним.

Усі облікові дані автентифікації, які використовуються в програмах і службах IT-структури, вважаються секретними . Це включає паролі, ключі ssh, ключі API, маркери OAuth і файли конфігурації.

Керування секретами можна розглядати як розширене керування паролями, яке включає створення, ротацію, відкликання та зберігання облікових даних.

Зрештою, сфера застосування в цьому випадку ширша, але мета залишається для захисту від несанкціонованого доступу до даних і систем, втрати даних і порушень [3].

Управління секретами сприяє кібербезпеці в трьох випадках. Вони такі:

- Безпека інфраструктури – запобігає вторгненню користувачів, пристроїв, програм та інших елементів мережі;
- Безпека хмарних служб – дозволяє обмежувати та керувати доступом до хмарних служб;
- Безпека даних – це дає змогу захистити критично важливі системи, серед інших ресурсів, від втрати та зламу даних.

Ще однією перевагою управління секретами є допомога організаціям у відповідності до вимог вимогливих стандартів кібербезпеки, таких як FIPS, NIST і HIPAA.

Є кілька способів забезпечити безпечний метод захисту секретів.

Централізоване управління секретами

Секрети повинні централізовано зберігатись в одному місці, щоб забезпечити більшу безпеку та полегшити керування ними. Це спрощує створення управління, безпеки та аудиту, щоб знати, хто і коли отримує доступ до цієї інформації.

ACL (списки контролю доступу)

Зібравши свої секрети в одному місці, переконайтеся, що потрібні люди мають до них доступ. Для цього ви можете створити списки ACL людини, машини та програми, які надають вам контроль над цим доступом.

Тимчасові облікові дані

Тимчасові облікові дані та шифрування повинні працювати разом: вони полягають у наявності динамічних секретів.

Шифрування

Важливо, щоб дані, що передаються або перебувають у стані спокою, могли бути зашифровані з централізованими ключами шифрування в управлінні секретами .

Аудит

Тепер вам може бути цікаво, як перевірити керування секретами та дізнатися, до чого хто з користувачів отримав доступ.

Кожен динамічний секрет може використовуватися окремим користувачем, який належним чином автентифікований під час отримання цієї інформації, а шифрування як послуга дозволяє знати, хто отримав доступ до операції шифрування та дешифрування [4].

1.2 Змінні середовища

Змінні середовища — це іменовані рядки, доступні для всіх програм. Змінні використовуються для адаптації поведінки кожної програми до середовища, у якому вона працює. Це динамічне значення, яке операційна система та інше програмне забезпечення може використовувати для визначення інформації, специфічної для комп'ютера або програмного продукти над яким працюєте.

Іншими словами, це щось, що представляє щось інше, наприклад розташування на вашому комп'ютері, номер версії , список об'єктів тощо. Існує два типи: змінні середовища користувача та змінні системного середовища [5].

Змінні та константи є двома ключовими будівельними блоками будь-якої мови програмування. Вони приймають значення, які змінюють результати програми, подібно до незалежних змінних у математичному рівнянні. І константи, і змінні позначають окремі області пам'яті, що містять дані, які програмне забезпечення використовує для виконання обчислень.

Різниця між ними полягає в тому, що постійні значення не можуть бути змінені під час виконання, змінні значення можуть бути змінені. Змінна — це місце для зберігання даних, які мають значення та символічне ім'я (ідентифікатор), яке супроводжується ним. Іншими словами, змінна - це місце, де зберігаються дані. Імена рядків даються для ідентифікації різних змінних.

Значення змінної можна змінити поза програмою за допомогою функцій, які часто надаються операційною системою або мікросервісом. Будь-яка кількість значень може бути створена та доступна для довідки в певний час як змінна середовища, яка складається з пари імені та значення (NVP).

Операційна система використовує змінні середовища, які є рядковими змінними, що зберігають дані, для керування службами та програмами. Динамічно назване значення, відоме як змінна середовища, може впливати на поведінку активних процесів на комп’ютері. Вони є компонентом середовища, в якому відбувається процес.

Програми використовують змінні середовища, щоб визначити, куди встановлювати файли, зберігати тимчасові файли та знаходити дані профілю користувача. Вони керують операційним середовищем операційної системи та програм [6].

Змінні середовища, на відміну від змінних оболонки, переносяться в дочірні процеси оболонки, і ці змінні, як правило, не є тим, з чим кінцеві користувачі повинні користуватися, оскільки вони загальносистемні та доступні для підоболонок і дочірніх процесів. У деяких випадках ми можемо змінити їх відповідно до наших потреб. Для програми JAVA, наприклад, встановлюється загальносистемний шлях, а для двійкових файлів встановлюється PATH. Майже завжди визначається або змінюється змінні середовища за допомогою команди експорту.

Змінні середовища користувача застосовуються до поточного сеансу оболонки і, як випливає з їх назви, визначаються користувачем. У результаті значення змінної під час входу в систему як один користувач на комп’ютері може відрізнятися від значення тієї самої змінної під час входу в систему як інший користувач.

Будь-який користувач, який увійшов у систему, має можливість вручну встановити ці типи змінних середовища. Змінні середовища визначають папки для тимчасових файлів, параметри програми, шляхи пошуку файлів та іншу інформацію подібного характеру. Кожен блок середовища користувача та блок середовища комп’ютера обслуговуються системою. Специфічні параметри середовища для кожного користувача на цьому конкретному комп’ютері представлені блоком системного середовища [7].

Змінні визначаються за допомогою пар ім’я-значення: NAME=будь-який рядок як значення. Ім'я змінної зазвичай пишеться великими літерами. Усе, що

слідує за знаком рівності, вважається значенням змінної до кінцевого символу переходу рядка.

У більшості випадків найзручніше зберігати ці змінні у файлі конфігурації, який зчитаються під час завантаження системи та входу користувача, щоб вони були доступні автоматично.

Змінні середовища успадковуються; тобто батьківська програма встановлює середовище для дочірнього процесу. Потрібно налаштувати параметри батьківського елемента так, щоб він передавав його всім своїм дочірнім елементам.

Батьківськими програмами, є різні оболонки та менеджери вікон, але кожна з них під час запуску читає окремий файл конфігурації (точковий файл).

1.3 Загальний алгоритм використання змінних середовища

Спочатку розглянемо, найпопулярніші сценарії використання змінних середовища:

Тип середовища

Змінні середовища часто використовуються для збереження назви середовища, у якому зараз працює програма. Логіка програми може використовувати це значення для доступу до потрібного набору ресурсів або ввімкнення/вимкнення певних функцій чи розділів програми.

Доменне ім’я

Доменне ім’я програми може відрізнятися залежно від середовища. Його ізоляція також допоможе вам легко вносити зміни в доменне ім’я вашої програми, не шукаючи його входження в кодовій базі.

URL- адреси API

Кожне середовище програми також може мати API, розгорнуті в різних середовищах.

Приватні ключі

Ключі до платних послуг і ресурсів мають бути ізольовані від вихідного коду програми, щоб вони випадково не потрапили в чужі руки.

Номери облікових записів служби

Зміна інформації, що стосується системи, наприклад номери облікових записів служби, клавіші тощо, залежно від середовища програми для керування ресурсами та моніторингу.

Змінні середовища визначаються на загальносистемному рівні. Змінні можна використовувати, наприклад, у коді веб-сайту, а також у сценарії оболонки, який ви створюєте для імпорту деяких даних у свою базу даних. Керуючи цими змінними в одному місці в системі, не доведеться турбуватися про те, де їх розмістити у вашому коді. Таким чином є можливість зберігати код і (що особливо важливо, конфіденційні) дані окремо.

Конфігурація

Робота зі змінними середовища під час роботи над програмним забезпеченням може мати багато позитивного впливу на робочі процеси розробки. Використовуючи їх, доведеться лише один раз потурбуватися про налаштування та конфігурацію програми.

Наприклад, у середовищі розробки вимкнувши кешування та показати більше журналів налагодження. У робочому середовищі точно не потрібно вмикати журнал налагодження, оскільки це призведе до зниження продуктивності.

Необхідність створювати окремі файли конфігурації для кожної ситуації призводить до великих накладних витрат і не є масштабованою. За допомогою змінних середовища розробник просто створює одну конфігурацію та налаштовує її на основі значень, які визначає.

Безпека

Зберігати паролі чи іншу конфіденційну інформацію системі контролю версій, наприклад Git , ніколи не буде гарною ідеєю. Для цього існує файл, .gitignor який забезпечує, недоступність репозиторія для вказаних файлів. Тож замість встановлення пароля безпосередньо у файлах конфігурації у кодовій базі використовується посилання на змінну середовища.

Ще одна перевага розділення коду та конфігурації полягає в тому, що після зміни пароля або стороннього ключа доведиться оновити лише змінну середовища. Для набуття чинності не потрібно змінювати код.

Мінімізація виробничих помилок

Помилка, яку розробники, хоч раз допускали, це робота в середовищі тестування з налаштуванням середовища виробництва. Наприклад, сервер вихідної пошти, налаштований у робочому стані, дублюється після відновлення або резервного копіювання в локальне середовище.

Якщо поштовий плагін підтримує використання змінних середовища для його налаштування, необхідно їх використовувати. Тому не доведеться турбуватися про заплановане завдання, яке полягає в розсиланні поштою деяких тестових замовлень, яка раніше була створена у своєму середовищі тестування. Якщо налаштувати своє невиробниче середовище, наприклад, на використання перехоплювача пошти, вони ніколи не потраплять у поштову скриньку користувачів [8].

Отже, однією з найважливіших причин, чому розробники повинні використовувати змінні середовища у своїх програмах, є дотримання популярного та корисного принципу проектування — розділення інтересів. Принцип проектування стверджує, що комп’ютерні програми мають бути розділені на окремі розділи для ефективного керування ними. Кожен розділ має ґрунтуватися на одному з основних завдань програми, і між такими розділами має бути мінімальний зв’язок.

Змінні середовища допомагають ізолювати критичні конфігураційні дані програми за допомогою файлів env або віддалених сховищ змінних. Таким чином ваші розробники отримають лише ту інформацію, яка їм потрібна.

Конфігурація програми є одною з таких проблем; тому її потрібно відокремити від основної програми. Один із найкращих способів зробити це — зберегти його у зовнішньому файлі та вставити його за потреби.

Таким чином ми дійшли до постановки задачі, яка буде детально розкрита в наступному розділі 2 - як швидко, безпечно та з мінімізацією використання ресурсів розробників, передавати в рамках команди, десятки, а інколи і сотні змінних середовища.

1.4 Аналіз використання змінних середовища, що вже реалізоване на ринку інформаційних технологій

Секрети програмного продукта належать до конфіденційної інформації. Якщо до них потраплять не ті люди, вони зможуть отримати доступ до внутрішньої архітектури програми та сторонніх ресурсів. Типовими прикладами є ключі AWS і дані системного облікового запису. Несанкціонований доступ до цих ключів може призвести до втрати грошей і даних програми. Хакери можуть навіть обмежити нормальну роботу програми.

Тому важливо зберегти ці секрети. Якщо залишити їх у кодовій базі, усі розробники отримають до них доступ. Не дотримання належних методів обфускації коду, додатки можуть бути оброблені зворотним шляхом, щоб отримати ключі, що містяться у коді. Виділення цих секретів за допомогою змінних середовища може запобігти подібним сценаріям.

Використання файлів .env

.env файли, безсумнівно, є найпростішим і найпопулярнішим способом керування змінними середовища. Зберігаються змінні середовища у файлі .env з кореневою назвою проекту. Програма запитує змінні в цьому файлі та завантажує їх для використання під час виконання. Ось як .env виглядає типовий файл:

VAR_FIRST=SOME_KEY_HERE

VAR_SECOND=SOME_OTHER_KEY_HERE

.env файли також дозволяють визначати набори змінних середовища та отримувати до них доступ на основі середовища виконання програми чи інших факторів. Замість простого збереження файлу .env можливо створити більше одного файлу та зберігати їх як .env.dev і .env.prod. У цих файлах можна визначити ті самі набори змінних, але з різними значеннями залежно від середовища.

Переваги:

1. Цей метод є найпростішим серед методів управління змінними середовища. Все, що потрібно зробити, це створити звичайний текстовий файл, який містить секрети, і зберегти його в корені проекту
2. Перемикати середовища так само просто, як змінити сам файл env. Ви можете зберігати декілька файлів під іменами .env.dev, .env.prod,

.env.uat тощо та налаштувати вихідний код для доступу до цих файлів залежно від середовища, у якому він працює.

3. Ви можете легко налаштувати .env файли в локальному середовищі розробки. На відміну від власних менеджерів змінних платформи, вам не потрібно розгортати свою програму, щоб використовувати функціональні можливості змінних середовища. Порівняно з секретними менеджерами, .env файли легше налаштувати локально, а доступ до секретів програми не залежить від мережі.
4. Є численні пакети з відкритим кодом, які допоможуть завантажувати та керувати секретами програми з env файлів.

Недоліки:

1. .env файли зберігають секрети вашої програми у формі пар ключ-значення. Звичайний формат для зберігання змінних середовища у .env файлі:

   Key1=Value1
2. Потрібно суворо дотримуватися вказаного формату, щоб програма могла успішно читати секрети. Одну невелику помилку десь між десятками або сотнями рядків змінних середовища, весь файл може бути не проаналізовано, і програма видаватиме непов’язані помилки у всьому. Той факт, що є помилка синтаксичного аналізу вашого .env файлу, може навіть не висвітлюватися.
3. Схильність до випадкового витоку секрету під час спільного використання/зберіганню Оскільки .env файли є простими текстовими файлами, вони вразливі до випадкового розкриття, якщо зберігаються на спільному жорсткому диску або надсилаються через незахищену мережу. Тому необхідно бути особливо обережним, щоб не розкрити секрети своєї програми, коли ви зберігаєте їх за допомогою. Безпосередньо рішенням цієї задачі і лежить в основі магістерської роботи.

Використання власного сховища змінних на платформі

Іншим популярним варіантом зберігання змінних середовища є використання сховища змінних вашої платформи розгортання. Більшість середовищ розгортання, таких як Heroku, AWS, Netlify тощо, надають користувачам простір для завантаження секретів.

Переваги:

1. Оскільки цей параметр повністю керується платформою розгортання, він буде безпечнішим, ніж зберігання секретів у звичайному текстовому файлі.
2. Оновлення змінних середовища, коли вони зберігаються незалежно, простіше — вам не потрібно редагувати вихідний код і створювати для нього новий випуск. Ви можете просто змінити значення на платформі та перебудувати свій проект. Він отримає нові значення під час наступного запуску.
3. Проблеми з форматуванням також зникли, оскільки більшість менеджерів розгортання на конкретній платформі розтирають ключі, коли ви їх вводите
4. Оскільки до платформ розгортання може отримати доступ вся ваша команда, ви можете легко поділитися секретами з потрібними людьми без необхідності надсилати текстові файли через Інтернет. Ви можете контролювати, хто має доступ до диспетчера змінних (у більшості випадків), і використовувати його як центральне сховище секретів ваших програм.

Недоліки:

1. Висока залежність від специфічної платформи, яка використовується. У деяких випадках ваша платформа розгортання може навіть не пропонувати таку послугу. Зміна платформи розгортання для отримання доступу до такої послуги може здатися не найкращим рішенням.
2. Оскільки платформи пропонуються та повністю управляються платформою розгортання, такі послуги можуть бути дуже

неуніфікованими. Переміщення змінних з однієї платформи на іншу може бути проблематичним.

3. Хоча такі служби чудово підходять для доступу до змінних середовища в розгортаннях програми, рідко існує шанс, що є можливість використовувати їх під час локальної розробки програми. У більшості випадків доведеться вдатися до керування локальними .env файлами. Хоча це виконує свою мету, це непотрібно ускладнює всю установку.

Використання секретних менеджерів

Секретні менеджери — це сторонні служби, які дозволяють вам повністю ізолювати секрети програми від вихідного коду/розгортання та отримувати їх за потреби через безпечні мережеві з'єднання.

Переваги:

1. Оскільки секрети зберігаються в повністю ізольованій службі, найімовірніше, ніколи не розкриються випадково під час обміну ними з колегами або через комісії контролю версій. Платформа третьої сторони дбає про безпеку ваших секретів, і вони зазвичай мають досить суворі SLA, коли йдеться про безпеку даних.
2. Секрети не залежать від вашої кодової бази та середовищ розгортання, є одноманітність в усіх середовищах. Не потрібно вживати особливих заходів для залучення нових розробників або вживати особливих заходів перед тим, як запускати програму у виробництво — більшість цих аспектів спрощено або про них потурбується менеджер секретів.

Недоліки:

1. Менеджери є повністю незалежними службами, вони мають власну вартість операцій. Отже, користувачі повинні нести ці витрати під час використання цих послуг.
2. Оскільки технологія досить нова, не можливо бути впевнені, наскільки добре вона буде прийнята в галузі найближчими днями. Хоча секретні менеджери демонструють великі перспективи з точки зору безпеки та

простоти керування, фактор вартості та проблеми з обробкою даних можуть призвести до досить повільного впровадження технології [9].

1.5 Постановка задачі

Отже, опираючись на власний досвід розробки програмного продукту, проаналізувавши існуючі на ринку рішення, для передачі змінних середовища, було прийнято рішення реалізації програмного забезпечення у вигляді веб додатка, з простим та легким дизайном і мінімалістичним інтерфейсом. Головна та основна задача майбутнього додатку, дозволити розробникам в рамках команд, а також базуючись на окремих проєктах в рамках цих же команд швидко в один - дві кліка, передавати змінні середовища. Тобто безпечність та надійність збереження даних, є первинним аспектом, на який потрібно орієнтуватись в ході розробки.

Постановка задачі:

1. Підібрати оптимальні рішенні для шифрування та дешифрування даних в повідомленнях.
2. Реалізувати програмний код для безпечної та захищеної роботи додатка.
3. Розробити графічний інтерфейс, опираючись на потреби майбутніх користувачів.
4. Протестувати розроблений веб – додаток, переконатись в можливості використання розділеного інтерфейсу для різних користувачів, різних проєктів, відгалудженості даних.

2. ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧІ

2.1 Шифрування AES-GCM з використанням OpenSSL

OpenSSL — це програма та бібліотека, яка підтримує багато різних криптографічних операцій, зокрема:

- Шифрування з симетричним ключем
- Генерація пари відкритий/приватний ключ
- Шифрування відкритим ключем
- Хеш-функції
- Створення сертифіката
- Цифрові підписи
- Генерація випадкових чисел

Кожна з операцій, які підтримує OpenSSL, має різноманітні параметри, наприклад файли введення/виведення, алгоритми, параметри алгоритмів і формати.

Для виконання своєї роботи, я обираю шифрування з симетричним ключем - AES (Advanced Encryption Standard). Алгоритми із симетричним ключем — це алгоритми криптографії, які використовують однакові криптографічні ключі як для шифрування відкритого тексту, так і для дешифрування зашифрованого тексту [9].

Розширений стандарт шифрування з режимом лічильника Галуа (AES-GCM) представлений Національним інститутом стандартів і технологій Сполучених Штатів Америки (NIST). AES-GCM підходить для використання в комунікаційних або електронних програмах [10]. AES-GCM — це режим роботи з блоковим шифруванням, який забезпечує високу швидкість автентифікованого шифрування та цілісність даних [11].

Режим лічильника Галуа, шифрування GCM, є рекомендованим алгоритмом для автентифікованого шифрування з пов’язаними даними. GCM побудовано на основі схваленого блочного шифру симетричного ключа з розміром блоку 128 біт, наприклад, алгоритму Advanced Encryption Standard (AES). Таким чином, GCM є режимом роботи алгоритму AES. GCM забезпечує гарантію конфіденційності даних, використовуючи різницю режиму роботи лічильника для шифрування. GCM забезпечує автентичність конфіденційних даних (приблизно до 64 гігабайт на

виклик) за допомогою універсальної хеш-функції, яка визначається над двійковим полем Галуа. GCM також може забезпечити автентифікацію для додаткових даних (практично необмеженої довжини на виклик), які не є зашифрованими. Якщо вхідні дані GCM обмежені даними, які не підлягають шифруванню, отримана спеціалізація GCM, яка називається GMAC, є просто режимом автентифікації вхідних даних.

Дві функції GCM називаються автентифікованим шифруванням і автентифікованим дешифруванням. Кожна з цих функцій є відносно ефективною та паралельною. Отже, реалізації з високою пропускною здатністю можливі як в апаратному, так і в програмному забезпеченні. GCM має кілька інших корисних характеристик, зокрема такі:

- Функції GCM є «онлайн» у тому сенсі, що довжина конфіденційних даних і додаткових неконфіденційних даних не повинна заздалегідь бути визначена. Натомість довжину можна обчислювати, коли дані надходять і обробляються.
- Функції GCM вимагають лише прямого напрямку основного блочного шифру.
- Автентичність захищених даних можна перевірити незалежно від відновлення конфіденційних даних із їх зашифрованої форми.
- Якщо унікальний рядок ініціалізації є передбачуваним, а довжина конфіденційних даних відома, тоді виклики блочного шифру в механізмі шифрування GCM можуть бути попередньо обчислені.
- Якщо деякі або всі додаткові неконфіденційні дані зафіксовані, тоді можна попередньо обчислити відповідні елементи механізму автентифікації GCM.

2.2 Алгоритм роботи режима Галуа/лічильник (GCM)

У режимі GCM для обробки використовується вектор ініціалізації (IV - initialization vector). Цей режим використовується для автентифікованого шифрування пов’язаних даних. GCM забезпечує конфіденційність і автентичність зашифрованих даних і автентичність додаткових автентифікованих даних (AAD -

additional authenticated data). AAD не зашифровано. Режим GCM вимагає, щоб IV був одноразовим, тобто IV має бути унікальним для кожного виконання режиму під заданим ключем. Кроки для шифрування GCM:

1. Хеш-підключ для функції GHASH генерується шляхом застосування блочного шифру до «нульового» блоку (формула 2.2)

$$\mathrm{GHASH}(H, A, C) = X_{m+n+1}. \quad (2.2)$$

де $H = E_k(0^{128})$ - хеш-ключ, string з 128 нульових біт, зашифрований з використанням блочного шифру, A - дані, які тільки автентифіковані (не зашифровані), C - зашифрований текст, m - число 128 - бітних блоків в A (з округленням у більшу сторону), n - кількість 128-бітних блоків у C (з округленням у більшу сторону), а змінна X і для i = 0, ..., m + n + 1 визначена нижньою

2. Блок попереднього лічильника генерується з IV. Зокрема, коли довжина IV становить 96 біт, то рядок заповнення $0^{31}||1$ додається до IV для формування блоку попереднього лічильника.

В іншому випадку IV доповнюється мінімальною кількістю «0» бітів, так щоб довжина результуючого рядка стала кратною 128 бітам (розмір блоку); цей рядок, у свою чергу, додається з 64 додатковими бітами «0», за якими слідує 64-бітове представлення довжини IV, і функція GHASH застосовується до результуючого рядка для формування блоку попереднього лічильника. Тобто автентифікований текст і зашифрований текст окремо доповнюються нулями до числа, кратного 128 біт, і об'єднуються в одне повідомлення S_i (формула 2.3)

$$S_i \begin{cases} A_i & \text{for } i = 1, \dots, m-1 \\ A_m^* \parallel 0^{128-v} & \text{for } i = m \\ C_{i-m} & \text{for } i = m+1, \dots, m+n+1 \\ C_n^* \parallel 0^{128-u} & \text{for } i = m+n. \\ \mathrm{len}(A) \parallel \mathrm{len}(C) & \text{for } i = m+n+1 \end{cases} \quad (2.3)$$

де len(A) і len(C) - це 64-бітові уявлення довжин бітів A і C , відповідно, v = len(A) mod 128 - довжина в бітах останнього блоку A , u = len (C) mod 128 - довжина в бітах останнього блоку C і || позначає конкатенацію бітових рядків.

Тоді X_i визначається в формулі (2.4)

$$X_i = \sum_{j=1}^{i} S_j * H^{i-j+1} = \begin{cases} 0 & \text{for } i = 0 \\ (X_{i-1} \oplus S_i) * H & \text{for } i = m+1, \dots, m+n+1 \end{cases} \quad (2.4)$$

Друга форма - це ефективний ітераційний алгоритм (який X і залежить від X_{i-1}), отриманий методом Хорнера до першого застосування. Тільки останній X_{m+n+1} залишається виходом. Якщо необхідно розпаралелити обчислення хеша, це можна зробити, k разів.

AAD і зашифрований текст додаються з мінімальною кількістю «0» бітів, можливо, жодного, так що довжина бітів отриманих рядків є кратною розміру блоку. До конкатенації цих рядків додається 64-бітне представлення довжини AAD і зашифрованого тексту для створення блоку u.

3. Функція GHASH застосовується до блоку u для отримання єдиного вихідного блоку.

Цей вихідний блок шифрується за допомогою функції GCTR із блоком попереднього лічильника, який було згенеровано на кроці 2, а результат скорочується до вказаної довжини тегу для формування тегу автентифікації. Зашифрований текст і тег повертаються як вихідні дані. Відкритий текст може мати будь-яку довжину. Зашифрований текст матиме таку саму довжину, що й відкритий текст.

2.3 Імплементація шифрування в web додаток Ruby on Rails

Консольна команда дозволяє нам взаємодіяти з програмою Rails з командного рядка. Це корисно для тестування швидких ідей із кодом і зміни даних на стороні сервера, не торкаючись веб-сайту.

В своєму проєкті, додаток буду реалізовувати на найсвіжіших версіях Ruby – 2.7.6 та Ruby on Rails 7.0.3. (рис. 2.1)

```
yuliia@ADA-MB-Pro-1 themis % ruby -v
ruby 2.7.6p219 (2022-04-12 revision c9c2245c0a) [x86_64-darwin22]
yuliia@ADA-MB-Pro-1 themis % rails -v
Rails 7.0.3
yuliia@ADA-MB-Pro-1 themis %
```

Рисунок 2.1 – Поточна версія Ruby та Ruby on Rails

Використовуючи описаний алгоритм в попередньому підрозділі, спробуємо реалізувати шифрування та дешифрування простого рядка, із будь-яким змістом. Таким чином ми переконаємось, в правильності наших рішень, і будемо точно на конкретному прикладі бачити результат виконання коду на кожному етапі. При необхідності можливо повторювати ітерації на інших даних.

Отримуємо вхідні дані (рис 2.2)

1. Ключ
2. Унікальний IV Для кожного шифрування даних слід створювати новий випадковий IV. Також можна спробувати використовувати nonce (номер, що використовується один раз) – він відкритий, але випадковий і непередбачуваний.
3. Дані, які обробляються лише з автентифікацією (пов’язані дані)
4. Дані обробляються шляхом шифрування та автентифікації
5. Зашифровані дані входу :тег автентифікації

Тег автентифікації є входом для розшифровки. Якщо хтось втрутився в наші пов’язані дані чи зашифровані дані, розшифровка GCM помітить це та не виведе жодних даних (або поверне повідомлення про помилку, і ми повинні відхилити отримані дані без їх обробки)

```
2.7.6 :001 > user = User.last
  User Load (0.6ms)  SELECT "users".* FROM "users" ORDER BY "users"."id" DESC LIMIT $1  [["LIMIT", 1]]
 => #<User id: 1, email: "yulia@gmail.com", password_digest: [FILTERED], created_at: "2022-11-17 09:05:08.197430000 +0000",
2.7.6 :002 > message = 'ruby on rails'
 => "ruby on rails"
2.7.6 :003 > cipher = OpenSSL::Cipher.new('aes-128-gcm')
 => #<OpenSSL::Cipher:0x00007fadf310c3d8>
2.7.6 :004 > cipher.encrypt
 => #<OpenSSL::Cipher:0x00007fadf310c3d8>
2.7.6 :005 > # Встановлюєм випадкового ключа з бібліотеки. key має бути 16 байт.
 => nil
2.7.6 :006 > key = cipher.random_key
 => "\xF9R\xBB!?\x14\x0F\xD89\xF9;\xDB1\x98\\@"
2.7.6 :007 > # Встановлюєм випадковий iv з бібліотеки. iv має бути 12 байт.
 => nil
2.7.6 :008 > iv = cipher.random_iv
 => "v\x8BJy\r\xE2\xBD\xA74\x04\x1A\xDC"
2.7.6 :009 > # Дані автентифікації можуть бути пустими рядками, але було б краще встановити якесь значення.
 => nil
2.7.6 :010 > # Використовуємо full_name поточного юзера, що запезпечить додаткову унікальність для кожного повідомлення.
 => nil
2.7.6 :011 > cipher.auth_data = user.full_name
 => "Yuliia"
2.7.6 :012 > # Завершення шифрування
 => nil
2.7.6 :013 > ciphertext = cipher.update(message) + cipher.final
 => "\xCE:\xA3e\xAB\xDF\xA7\r&\x93\xB6\xFBA"
2.7.6 :014 > auth_tag = cipher.auth_tag
 => "V\xE5Q\xC6g\xE6_\x86 S\x17\xE8)\xA5\xB2w"
2.7.6 :015 > auth_tag = auth_tag[0] # одного байта достатньо
 => "V"
2.7.6 :016 >
```

Рисунок 2.2 – Вхідні дані, шифрування

Шифрування та дешифрування є дуже схожими операціями для симетричних алгоритмів, це відображається в тому, що не потрібно вибирати різні класи для кожної операції, обидві можна виконувати за допомогою одного класу. Однак після отримання екземпляра Cipher потрібно повідомити екземпляру, що саме з ним робити, і який саме метод викликати: або **encrypt**, або **decrypt**.

```
2.7.6 :016 > cipher = OpenSSL::Cipher.new('aes-128-gcm')
 => #<OpenSSL::Cipher:0x00007fae07911600>
2.7.6 :017 > cipher.decrypt
 => #<OpenSSL::Cipher:0x00007fae07911600>
2.7.6 :018 > # Ключ має мати 16 байт і значення, яке ви встановили під час дешифрування.
 => nil
2.7.6 :019 > cipher.key = key
 => "\xF9R\xBB!?\x14\x0F\xD89\xF9;\xDB1\x98\\@"
2.7.6 :020 > # iv має складати 12 байтів і значення, яке ми встановили під час дешифрування.
 => nil
2.7.6 :021 > cipher.iv = iv
 => "v\x8BJy\r\xE2\xBD\xA74\x04\x1A\xDC"
2.7.6 :022 > # Дані автентифікації мають бути встановлені нами під час дешифрування.
 => nil
2.7.6 :023 > cipher.auth_tag = auth_tag
 => "V"
2.7.6 :024 > cipher.auth_data = user.full_name
 => "Yuliia"
2.7.6 :025 > data = cipher.update(ciphertext) + cipher.final
 => "ruby on rails"
2.7.6 :026 >
```

Рисунок 2.2 – Дешифрування

2.4 Фреймворк Ruby on Rails, як основа стеку технологій

Ruby — це об’єктно-орієнтована мова програмування. На Ruby впливають кілька інших ООП, включаючи Perl, Lisp, Eiffel, Smalltalk і Ada. Він рефлексивний і динамічний, з автоматичним керуванням пам'яттю. Він також підтримує різноманітні парадигми програмування, такі як імперативна, функціональна і, звичайно, об’єктно-орієнтована. Ruby on Rails ,який іноді називають просто Rails, — це потужна структура веб-програм із повним набором відкритих кодів, призначена для роботи на мові Ruby. Система маршрутизації повністю незалежна від веб-сервера. Це дозволяє створювати програми та сторінки, які можуть спілкуватися та збирати інформацію з веб-сервера та бази даних, а також відтворювати шаблони. Rails сумісний із найпопулярнішими інженерними протоколами, включаючи Model–View–Controller (MVC), шаблон активного запису, Don’t Repeat Yourself (DRY) і Convention over Configuration (CoC). Безпека Ruby потребує різноманітних прикладних методів, не існує жодного інструменту чи процесу, який міг би ефективно захистити від усіх вразливостей і загроз безпеці Ruby. Найпоширеніші загрози безпеці Ruby включають доступ до/зміну конфіденційних даних компанії, компрометацію облікового запису, обхід автентифікації та контролю доступу та представлення шахрайського вмісту. Професіонали з безпеки використовують численні методи для належного захисту програм Ruby on Rails. Оскільки Ruby подібний до інших об’єктно-орієнтованих мов програмування, ті самі методи безпеки, які застосовуються до них, також застосовуватимуться до безпеки Ruby. Деякі з найпоширеніших методів безпеки включають статичний аналіз, перегляд коду та тести на проникнення. Ruby on Rails дійсно має деякі інтегровані процеси безпеки, які допомагають запобігти деяким із найруйнівніших атак, таких як атаки впровадження SQL та XSS [11].

Основні “слабкі” місця, веб додатку, до якого повинна бути підвищена увага, під час планування бізнес-логіки, та розробки додатка безпосередньо. Розглянемо основні помилки.

- Не вдалося обмежити доступ до URL-адреси
- Запобігання SQL у Ruby

- Міжсайтовий сценарій (XSS)
- Міжсайтова підробка запитів (CSRF)
- Незахищене криптографічне сховище
- Порушена автентифікація та керування сеансами
- Недійсні переадресації та переадресації
- Незахищені прямі посилання на об’єкти
- Недостатній захист транспортного рівня
- Неправильна конфігурація безпеки

PostgreSQL, також широко відомий як Postgres, — це система реляційних баз даних із відкритим вихідним кодом із розширеними можливостями корпоративного рівня. Він підтримує реляційні запити у формі SQL, а також нереляційні запити у форматі JSON.

Безпека Postgres базується на трьох стовпах:

Безпека на рівні мережі, включаючи використання сокетів Unix Domain, сокетів TCP/IP і брандмауерів.

Безпека на транспортному рівні, яка забезпечує безпечний зв’язок із базою даних за допомогою SSL/TLS.

Функції безпеки на рівні бази даних, такі як ролі та дозволи, захист на рівні рядків (RLS) і аудит.

3. РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПЕРЕДАЧІ ШИФРОФАНИХ ЗМІННИХ СЕРЕДОВИЩА

3.1 Побудова бази даних та зв'язків між таблицями в рамках парадигми ModelViewController

Користувачі веб додатку мають бути чітко розмежовані в доступі до різного функціоналу сайта. Тільки юзер з ролью admin має можливість керувати компаніями створювати, редагувати, видаляти тощо. Зв'язок компанії з проєктом, в якому буде відбуватись комунікація між розробниками теж повинна знаходитись під повним контролем адміністратора. Жоден інший користувач, може тільки бути доданий до компанії, і тільки після цього залишати повідомлення у відповідних каналах, що відповідають тому чи іншому проєкту.

Після реєстрації користувач (не адміністатор) відразу потрапляє до головного чату, і має має в своєму інтерфейсі тільки можливіть залишати повідомлення, та отримувати від інших користувачів. Ці всі обмеження і доступи, забезпечує база даних PostgresSQL (кінцевий вигляд бази даних – рисунок 3.1)

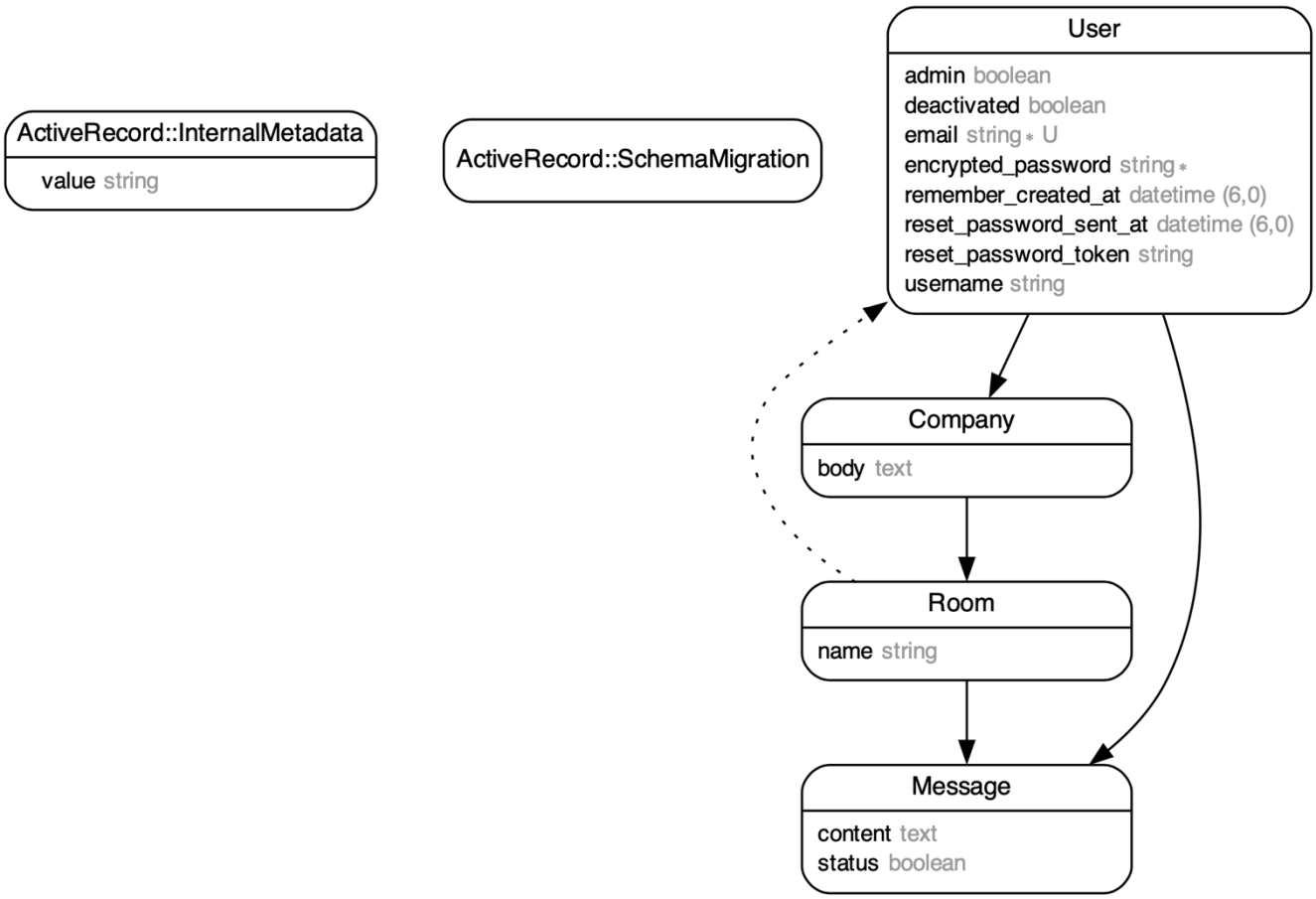

Рисунок 3.1 – Блок-схема бази-даних, що обслуговує веб - додаток

В ході розробки програмного продукту зміни до бази даних були забезпечені інстументом Ruby on Rails – міграціями (рис. 3.2). Відстеження змін, в базі даних, які необхідні виконати під час наступного розгортання, можна кожного разу вносити вручну, кожним девелопером, що працює над поточним продуктом.

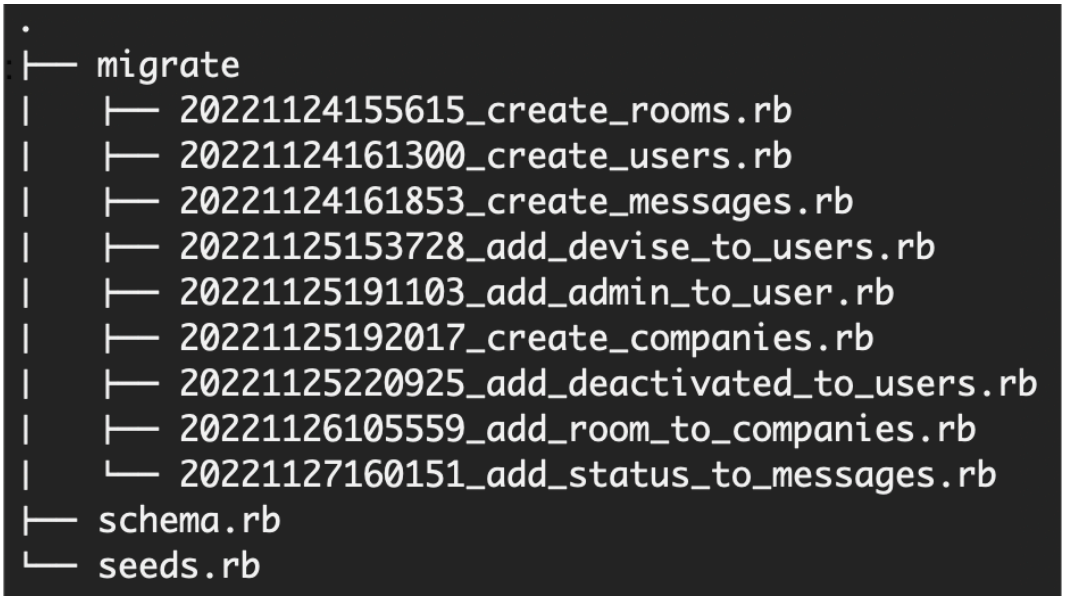

Рисунок 3.2 – Міграції, виконані в ході розробки програмного продукту

Моделі є класами в Rails. Вони взаємодіють з базою даних, зберігають дані, обробляють перевірку, транзакції тощо. Ця підсистема реалізована в бібліотеці ActiveRecord. ActiveRecord бібліотека забезпечує інтерфейс між таблицями бази даних і програмним кодом Ruby, який маніпулює записами бази даних. Реалізовані моделі в проєкті схематично зображені на рисунок 3.3.

В моделях вказуються валідації, які необхідні для коректної роботи зв’язків. Також така структура побудови веб додатку, дозволяє відносно легко покривати код тестами, та при найменьшому порушенні необхідних умов в моделі, тести відразу покажуть які поля або методи, зазнали змін та мають бути незайно повернуті або в попердній стан, або виправлені, щоб попередня бізнес логіка додатку, була цільна та непорушна.

RSpec - це інструмент тестування для Ruby, створений для розробки, керованої поведінкою (BDD - behavior-driven development). Це найбільш часто використовувана тестова бібліотека для Ruby у програмах.

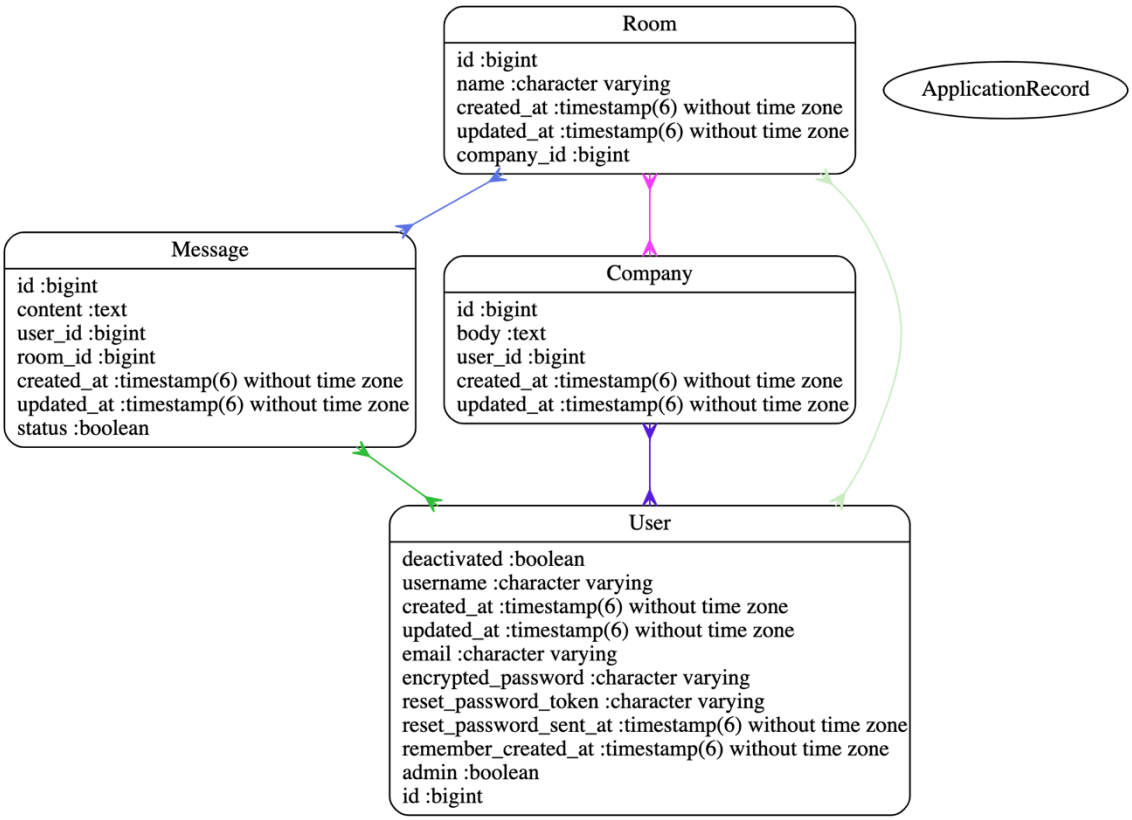

Рисунок 3.3– Блок-схема моделей проєкту

Контролер спрямовує трафік до представлень і моделей – рисунок 3.4. Він запитує моделі для даних із бази даних і відображає потрібний результат за допомогою перегляду. Ця підсистема реалізована в бібліотеці ActionController і є посередником даних між ActiveRecord і ActionView. Контролер - це тип класу, який відповідає за визначення сенсу запиту, зробленого в програмі, і створення відповідного результату. Він діє як посередник між поглядами та моделлю. Перегляди використовуються для відображення даних користувачеві, тоді як моделі визначають методи маніпулювання даними. Він спілкується з моделлю, щоб перевірити, чи надісланий запит доступний у базі даних.

Маршрутизатор (ДОДАТОК Г) спочатку викликає контролер і необхідну дію для виконання. Потім Rails створює екземпляр контролера та запускає метод із тим же ім’ям, що й дія.

Це означає, що кожного разу, коли робимо запит, створюється об’єкт (примірник контролера), і коли запит завершується, об’єкт знищується.

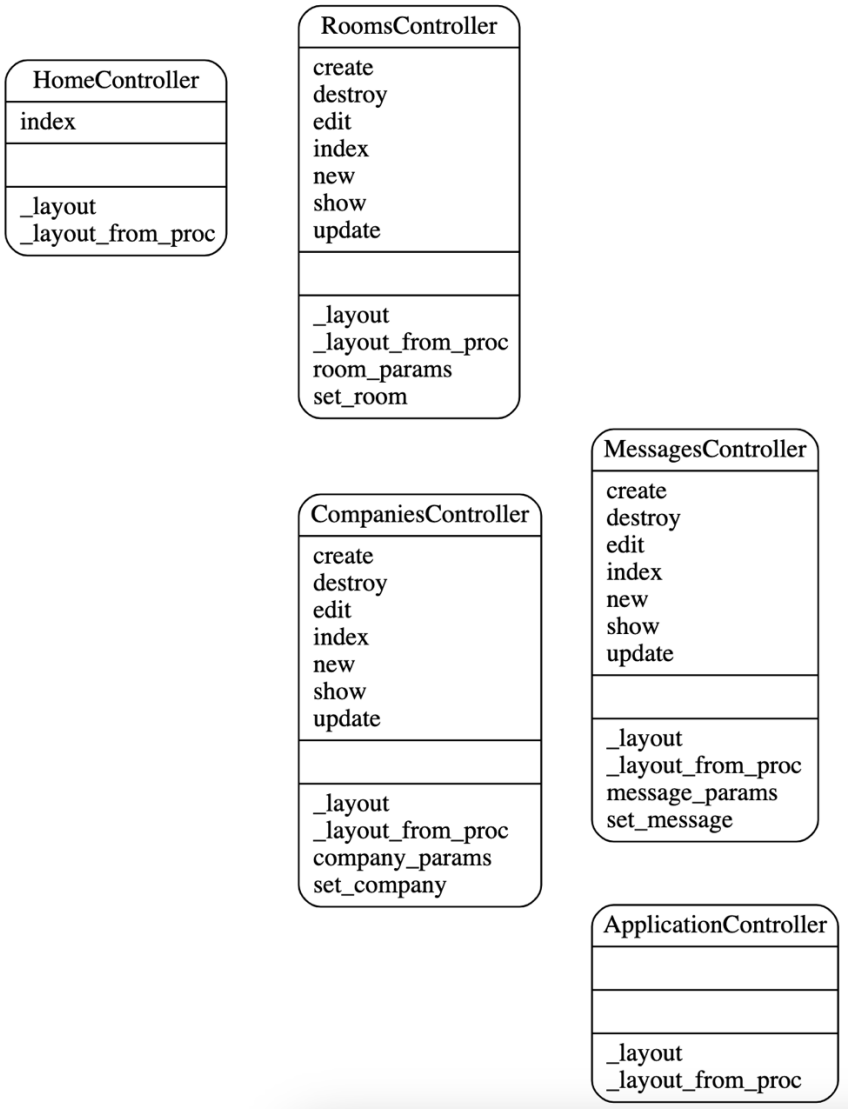

Рисунок 3.4 – Блок-схема контролерів проєкту

3.2 MessageEncryptor як інструмент, для використання OpenSSL

Для реалізації головної функції веб додатка – шифрування та дешифрування, використовується ActiveSupport::MessageEncryptor клас. Робота цього API детально описана та розібрана в розділі 2.3.

В ході розробки було вирішено створити, окремий сервіс (ДОДАТОК В), для шифрування та дешифрування, це два симетричні методи **encrypt** та **dencrypt**. Кожен метод відповідно використовуються в різних місцях. **encrypt** на стороні backend-у в контролері, де створюється повідомлення та зберігаються в базі даних (рисунок 3.5).

Отже, при написанні повідомлення в чаті, значення з повідомлення перетворюється в шифр:

message_params[:content] => Create example message

, та в ході роботи сервіса, дані перетворюються в шифр:

[["content", "w3nf/RlstNTuPyz8VMO34PVo8R91V3RKBI4F5+gWzE4=--byfHYt4c/Y/Fr+NF--nSU12qEgpz92HbiejXeAJQ=="]

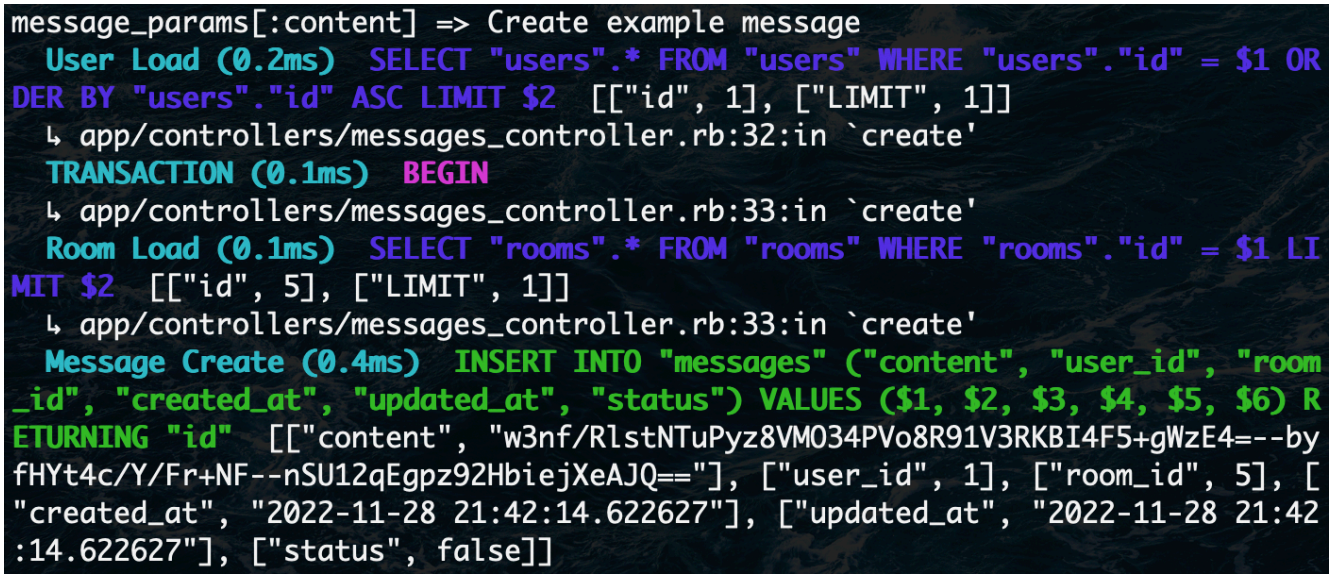

Рисунок 3.5 – Збереження зашифрованих даних в базу даних

В свою чергу для відображення на стороні frontend-а використовується хелпер, який за допомогою сервіса, перетворює його в повідомлення, для можливості прочитати користувачу, на сторінці веб - браузера.

3.3 Розробка та проєктування інтерфейса

Розглянемо приклад роботи месенджера, до якого підключені декілька користувачів (рис. 3.6).

Залогінивший адміністратор потрапляє на головну сторінку управління користувачами, компаніями, кімнатами в яких ведуться розмови. Адміністратор бачить посиляння до управляння: окремо кімнатами, окремо компаніями. Також він одразу бачить список всіх користувачів, які додані в усі його компанії. Якщо залогінений користувач не адміністратор, доступа до цієї панелі управління не має. Він одразу потрапляє до сторінки з чатами, і може відразу спілкуватись з колегами в рамках проєкту, якій йому необхідний.

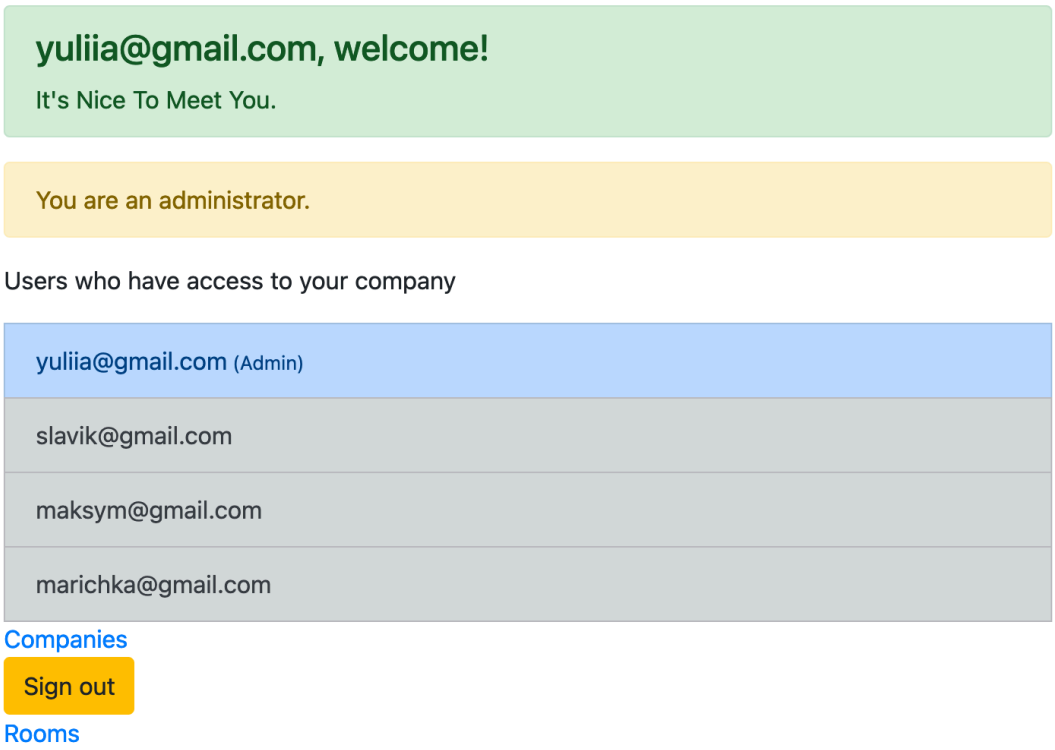

Рисунок 3.6 – Головна сторінка управління компаніями

Перейдемо до головної сторінки адміністратора (рисунок 3.7). Ввійшовши в свій акаунт, адміністратор спочатку потрапляє на головну сторінку керування своєї компанії (або компаніями, якщо їх декілька) та управління чатами для кожного проєкту окремо (рисунок 3.8). Створення окремої розмови, можливе тільки при обов'язковій умові: необхідно вибрати з запропонованого списку компанію. Їх видно в запропонованому списку, які можливо вибрати за допомогою чекбокса – компанії, що доступні для поточного адміністратора.

Зайшовши по посиланню "Companies" адміністратор бачить чітко відокремлену інформації по кожній компанії окремо. Для кожної компанії видно назву кімнати для розмови, і список користувачів, які підключені до кожної розмови окремо. Користувачі в рамках однієї компанії різні юзери, можуть бути підключені до різних розмов в будь яких комбінаціях. Окрім адміністратора. Адміністратор має доступ до всіх розмов своєї компанії.

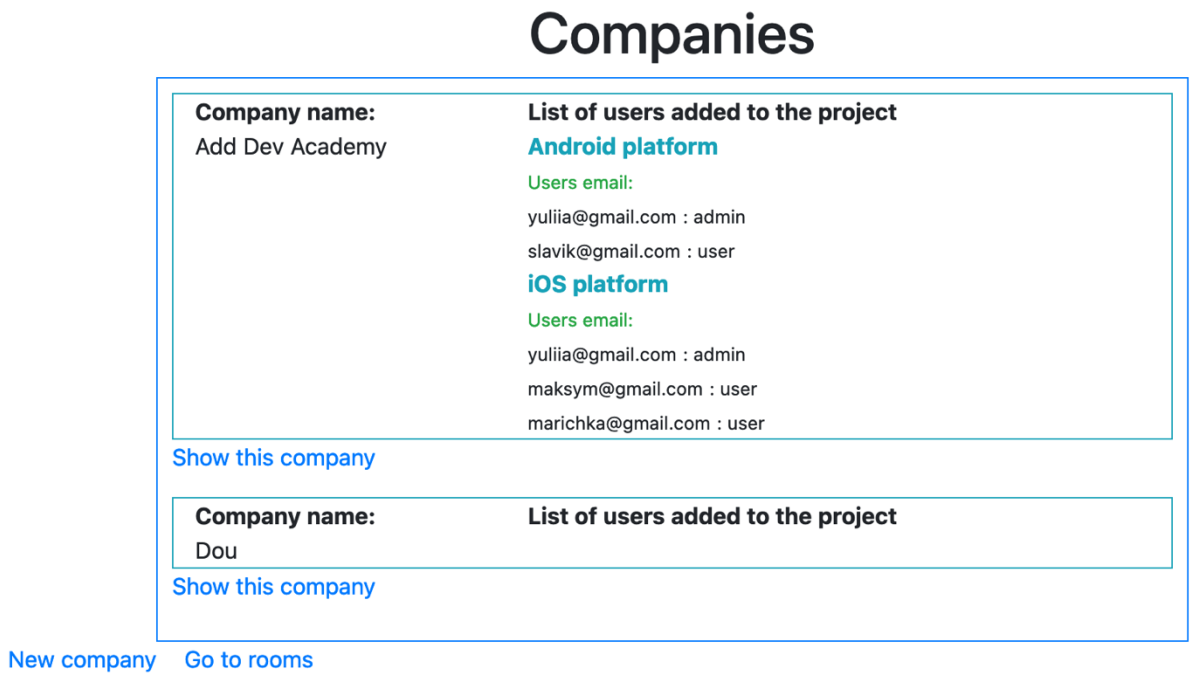

Рисунок 3.7– Панель управління компаніями

Перейшовши по посиланню “Go to rooms” адміністратор потрапляє, безпосереднь на сторінку з кімнатами, де має можливість їх створювати. На рисунку 3.8 видно цю частину інтерфейсу.

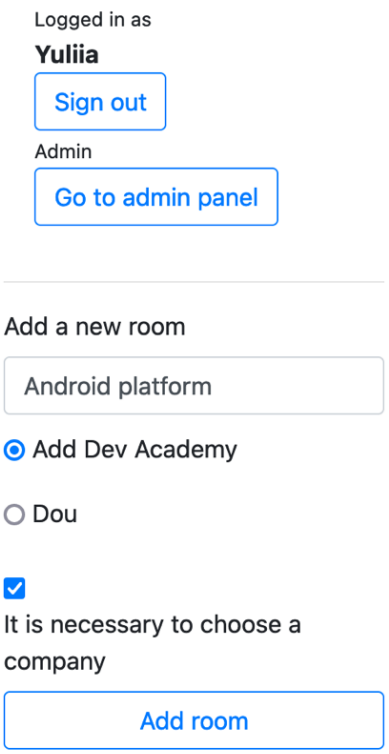

Рисунок 3.8 – Приклад створення нової кімнати, для обговорення нового проєкта

На рисунку 3.9 ми бачимо основні частини інтерфейсі – шапка розмови, в якій вказано назву проєкта, назва компанії до якого належить даний проєкт, а також кількість користувачів доданих до розмови. Дизайн поточної розмови в рамках проєкта доволі простий, бо робота чату заточена для вирішення вузької проблеми, тому зайві елементи тільки заважатимуть юзерам орієнтуватись в веб додатку.

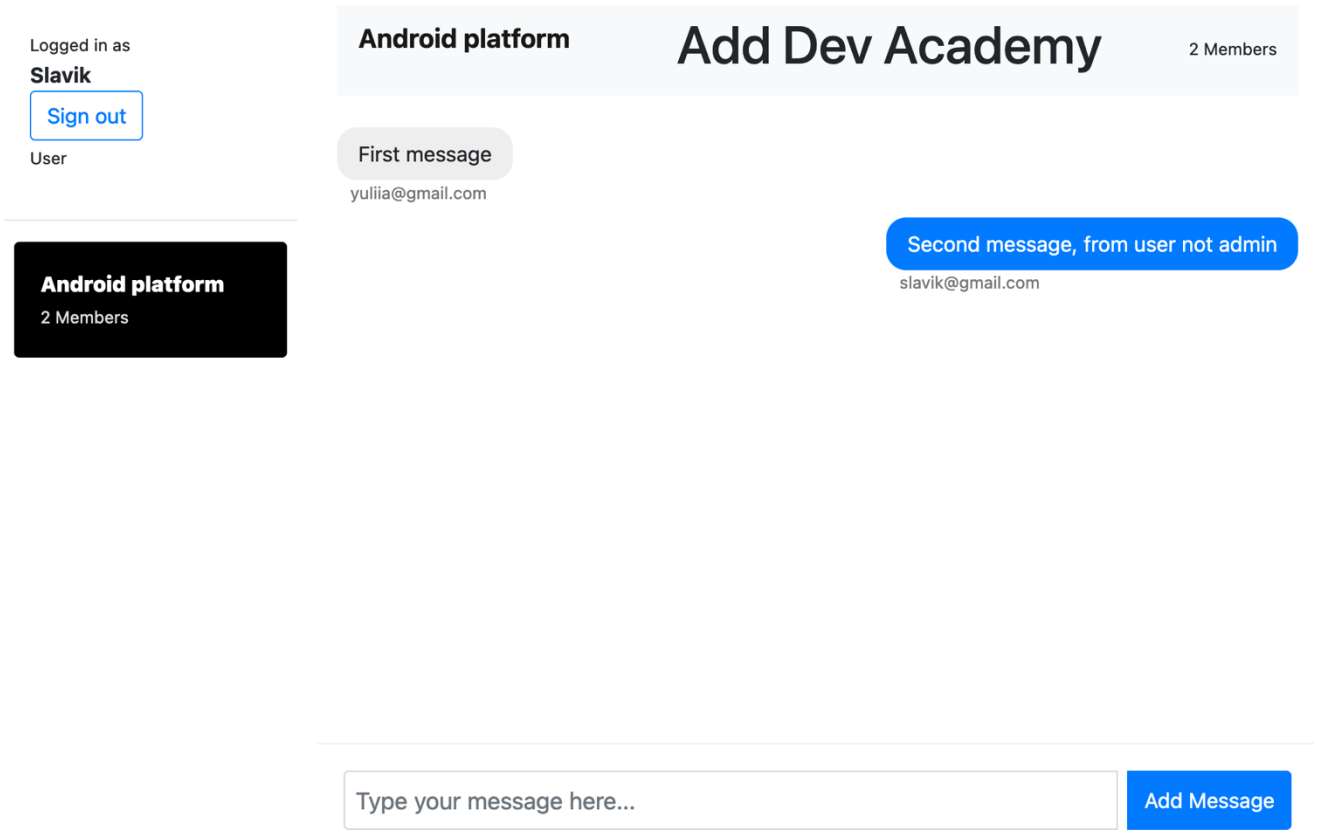

Рисунок 3.9 – Головна сторінка чату

Сторінка з чатом для простого користувача та адміністратора теж дещо відрізняється. У користувача немає можливості створювати нові розмови для проєктів, та мати доступ до управління будь-чим, окрім свого акаунта.

На рисунку 3.10 продемонстовано розмову трьох розробників, які комунікуть між собою та вирішують роботі моменти з приводу деяких змінних середовища.

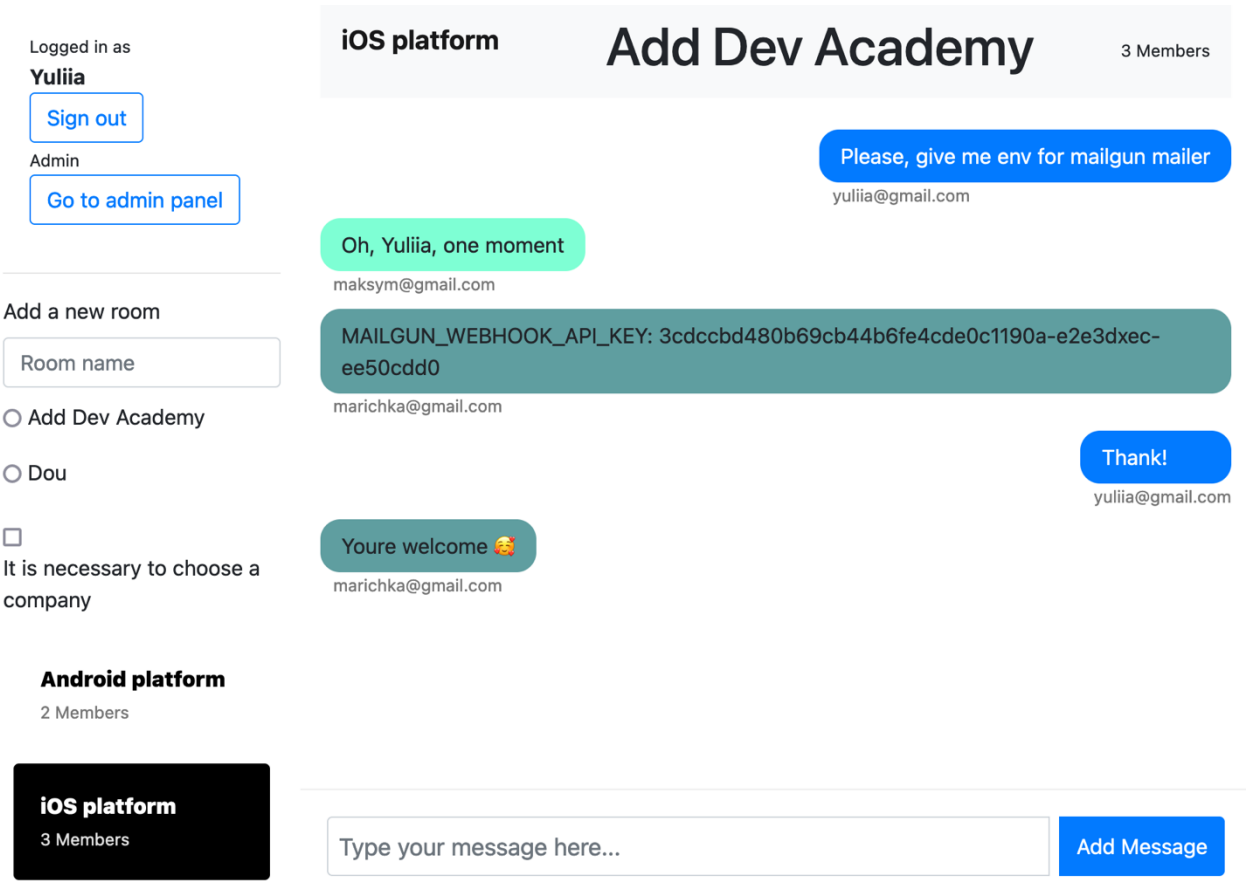

Рисунок 3.10 – Розмова в чаті трьох користувачів

Кожен окремий користувач має свій колір повідомлення, окрім поточного юзера, повідомлення якого знаходяться по правій стороні та мають в фоні синій колір. Також під кожним повідомленням відображається електронна пошта користувачі.

В лівому стовпчику відображаються всі кімнати які підключені до поточної компанії. Перемикатись між ними дуже легко - просто одним кліком, вибирати потрібну. Вона відразу по перемиканню, підсвічується іншим кольором.

За допомогою запита до бази даних:

```
room = Room.find_by(name: 'iOS platform')
messages = Message.where(room: room)
```

використовуючи rails консоль, переконаємось, що дані які відповідають за контент в повідомленнях зберігається зашифровано – рисунок 3.11.

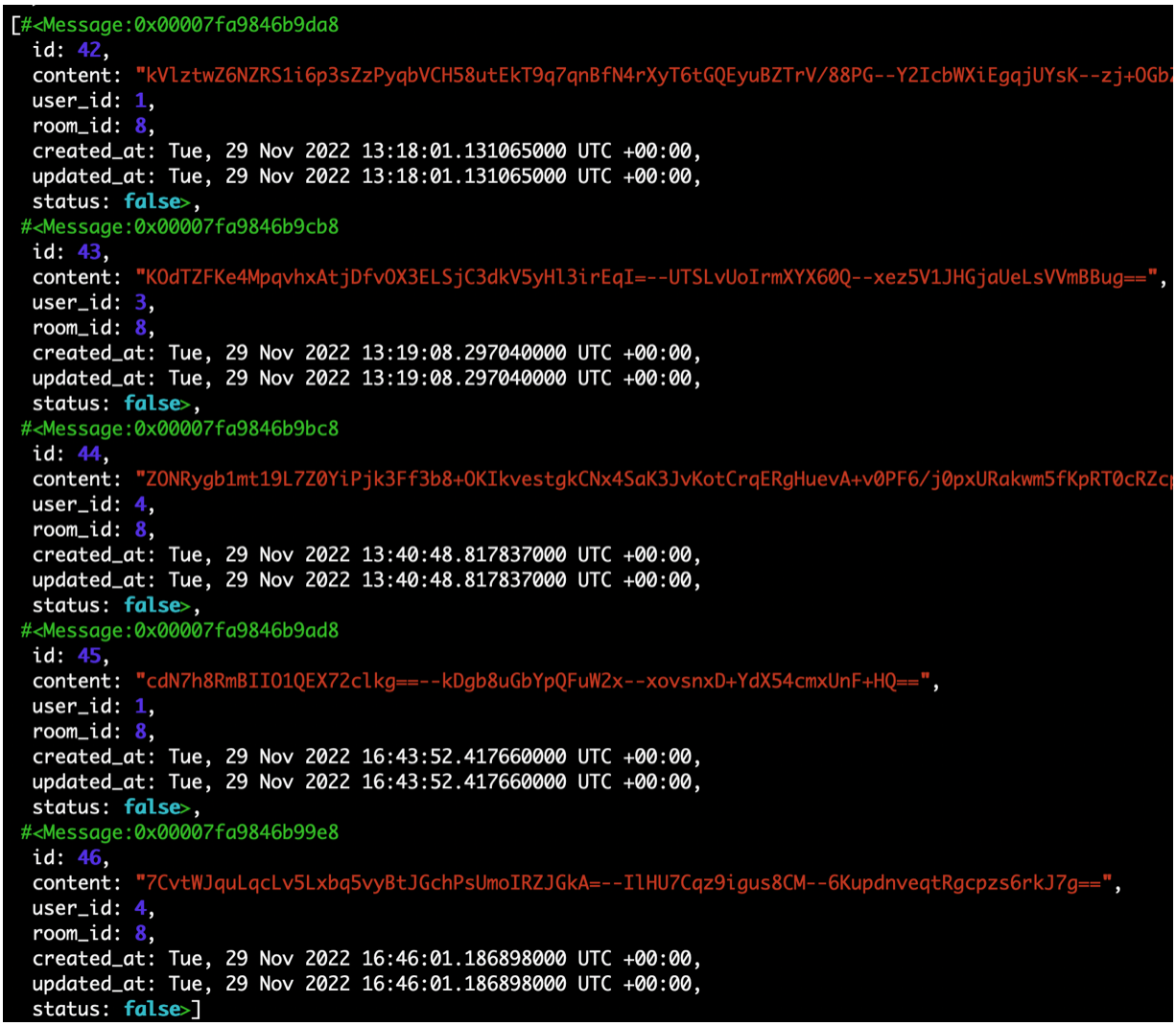

Рисунок 3.11 – Розмова в чаті трьох користувачів

Отже, без доступу до змінних середовища, які використовуються для шифрування/розшифрування повідомлення в чаті, доступ до бази даних, нічого не дасть зловмисникам. А втрата змінних середовища з одного боку не дозволить, прочитати вже збереженні повідомлення з бази даних, але з іншого боку інформація про розборку проєкту залишиться недоступна до всіх, і для чужих рук та очей також.

ВИСНОВКИ

Кожна команда розробників потребує спілкування. Такі аспекти процесу розробки, як керування завданнями, обговорення робочий планів, прийняття нагальний та швидких рішень тощо, здаються незначними, але мають критичне значення для командної роботи. Оперативність комунікації в рамках робочого процесу розробки програмного забезпечення, в світі, коли IT технології виходять на перші щаблі, важко недооцінювати. Тому розв'язання точкових проблем, які допомагають більше концентруватись на розробці майбутнього продукту, а не розґалуджувати увагу на дотичні труднощі, в результаті економлять ресурси як бізнесу, так і кінцевому користувачу.

В ході виконання кваліфікаційної магістерської роботи було:

1. Розглянуто та проаналізовано існуючі інструменти для передачі змінних середовищах.
2. Досліджено сучасні методи шифрування.
3. Опираючись на проведений аналіз, було підібрано відповідно до поставлених задач стек технологій, який дозволив реалізувати легкий в користуванні, але максимально безпечний додаток.
4. Розроблено веб-додаток, що вирішує задачі швидкої та безпечної комунікації в рамках різних команд. Інформаційну систему було реалізовано мовою програмування Ruby, фреймворком Ruby on Rails, базою даних PostgreSQL та бібліотекою OpenSSL.

Отриманий програмний продукт, можна використовувати під час розробки будь-яких інформаційних технологій. Він дозволяє швидше комунікувати між розробниками, і закриває таку нагальну проблему, як передача конфіденційних даних, що мають бути захищені.

СПИСОК ЛІТЕРАТУРИ

1. Magnusson, Andrew**.** The Definitive Guide to Authentication. www.strongdm.com - Режим доступу: https://www.strongdm.com/authentication.

2. Shacklett, Mary E. What is authentication? www.techtarget.com. - Режим доступу: https://www.techtarget.com/ searchsecurity/definition/ authentication.

3. How to Properly Manage Secrets in Development Projects . senhasegura.com. - Режим доступу: https://senhasegura.com/ challenges-to-manage-secrets-in-development-projects/.

4. Hassan, Ahmed Shamim. A definitive guide to authentication for software developers. betterprogramming.pub. - Режим доступу: https://betterprogramming.pub/how-do-you-authenticate-mate-f2b70904cc3a.

5. Fisher, Tim. User and System Environment Variables, and How to Find Their Values. www.lifewire.com. - Режим доступу: https://www.lifewire.com/what-are-environment-variables-2625868.

6. Horne, Starr. The Rubyist's Guide to Environment Variables. www.honeybadger.io. - Режим доступу: https://www.honeybadger.io/blog/ruby-guide-environment-variables/.

7. What is the Difference Between User Variables and System Variables? www.baeldung.com. - Режим доступу: https:// www.baeldung.com/cs/user-vs-system-variables.

8. Houben, Stijn. 3 benefits of environment variables and how to use them. hyperlane.co. - Режим доступу: https://hyperlane.co/blog/the-benefits-of-environment-variables-and-how-to-use-them.

9. Stemann, Phillip. Environment Variables: What They Are and How To Use Them. kinsta.com. - Режим доступу: https://kinsta.com/knowledgebase/what-is-an-environment-variable/.

10. Bull, Ian. Tutorial: AES Encryption and Decryption with OpenSSL. eclipsesource.com. - Режим доступу: https://eclipsesource.com/blogs/2017/01/17 /tutorial-aes-encryption-and-decryption-with-openssl/.

11. Qin, Leo. Aes chain block Cipher vs Galoiscounter modes of operation. www.leozqin.me. - Режим доступу: https://www.leozqin.me/aes-chain-block-cipher-vs-galoiscounter-modes-of-operation/.v

12. Advanced Encryption Standard with Galois Counter Mode using Field Programmable Gate Array. Ahmad, Nabihah. Bristol, United Kingdom : IOP Publishing, 2017 p.

13. Ruby on Rails SecurIty . checkmarx.com. - Режим доступу: https://checkmarx.com/glossary/ruby-on-rails-preventative-security/.

14. Galois/Counter Mode (GCM) . www.ibm.com. - Режим доступу: https://www.ibm.com/docs/en/zos/2.3.0?topic=operation-galoiscounter-mode-gcm.

15. Я. Р. Совин, В. В. Хома, В. І. Отенко. ПОРІВНЯННЯ AEAD-АЛГОРИТМІВ ДЛЯ ВБУДОВАНИХ СИСТЕМ ІНТЕРНЕТУ РЕЧЕЙ. місце видання: Національний університет “Львівська політехніка”, кафедра захисту інформації, 2019. Т. 1, No 1.

ДОДАТОК А

Лістинг модуля db/schema.rb

```
ActiveRecord::Schema[7.0].define(version: 2022_11_27_160151) do
  # These are extensions that must be enabled in order to support
this database
  enable_extension "plpgsql"

  create_table "companies", force: :cascade do |t|
    t.text "body"
    t.bigint "user_id", null: false
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["user_id"], name: "index_companies_on_user_id"
  end

  create_table "messages", force: :cascade do |t|
    t.text "content"
    t.bigint "user_id", null: false
    t.bigint "room_id", null: false
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.boolean "status", default: false
    t.index ["room_id"], name: "index_messages_on_room_id"
    t.index ["user_id"], name: "index_messages_on_user_id"
  end

  create_table "rooms", force: :cascade do |t|
    t.string "name"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.bigint "company_id", null: false
```

```
    t.index ["company_id"], name: "index_rooms_on_company_id"
  end

  create_table "users", force: :cascade do |t|
    t.string "username"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.string "email", default: "", null: false
    t.string "encrypted_password", default: "", null: false
    t.string "reset_password_token"
    t.datetime "reset_password_sent_at"
    t.datetime "remember_created_at"
    t.boolean "admin", default: false
    t.boolean "deactivated", default: true
    t.index ["email"], name: "index_users_on_email", unique: true
    t.index ["reset_password_token"], name:
"index_users_on_reset_password_token", unique: true
  end

  add_foreign_key "companies", "users"
  add_foreign_key "messages", "rooms"
  add_foreign_key "messages", "users"
  add_foreign_key "rooms", "companies"
end
```

ДОДАТОК Б

Лістинг представлення app/views/home/index.html.rb

```
    <div id="<%= dom_id company %> " class="container border border-
info">
      <div class='row'>
        <div class='col-3'>
          <strong>Company name:</strong>
          <%= company.body %>
        </div>
        <div class='col-1'>
        </div>
        <div class='col-6'>
        <b>List of users added to the project</b>
        <% company.rooms.each do |room| %>
          <strong class="text-info"><%= room.name %></strong><br>
          <small class="text-success">Users email:</small><br>
          <% room.users.uniq.each do |user| %>
            <small><%= user.email %></small>
            <% if user.admin? %>
              <small>: admin</small>
            <% else %>
              <small>: user</small>
            <% end %>
            <br>
          <% end %>
        <% end %>
      </div>
      </div>
    </div>
```

ДОДАТОК В

Лістинг сервіса app/services/encryption_servise.rb

```
class EncryptionServise
  salt = SecureRandom.random_bytes(
    ActiveSupport::MessageEncryptor.key_len
  )

  KEY = ActiveSupport::KeyGenerator.new(
    Rails.application.credentials.dig(:secret_key_base)
  ).generate_key(
    salt,
    ActiveSupport::MessageEncryptor.key_len
  ).freeze

  private_constant :KEY

  delegate :encrypt_and_sign, :decrypt_and_verify, to:
:encryptor

  def self.encrypt(value)
    new.encrypt_and_sign(value)
  end

  def self.decrypt(value)
    new.decrypt_and_verify(value)
  end

  private

  def encryptor
    ActiveSupport::MessageEncryptor.new(KEY)
```

```
    end
end
```

ДОДАТОК Г

Лістинг файла Gemfile

Лістинг файла config/routes.rb

```
source "https://rubygems.org"
git_source(:github) { |repo| "https://github.com/#{repo}.git" }

ruby "2.7.6"

# Bundle edge Rails instead: gem "rails", github: "rails/rails", branch: "main"
gem "rails", "~> 7.0.4"

# The original asset pipeline for Rails [https://github.com/rails/sprockets-rails]
gem "sprockets-rails"

# Use postgresql as the database for Active Record
gem "pg", "~> 1.1"

# Use the Puma web server [https://github.com/puma/puma]
gem "puma", "~> 5.0"

# Use JavaScript with ESM import maps [https://github.com/rails/importmap-rails]
gem "importmap-rails"

# Hotwire's SPA-like page accelerator [https://turbo.hotwired.dev]
gem "turbo-rails"
```

```
    # Hotwire's modest JavaScript framework [https://stimulus.hotwired.dev]
    gem "stimulus-rails"

    # Build JSON APIs with ease [https://github.com/rails/jbuilder]
    gem "jbuilder"

    # Use Redis adapter to run Action Cable in production
    gem "redis", "~> 4.0"

    # Windows does not include zoneinfo files, so bundle the tzinfo-data gem
    gem "tzinfo-data", platforms: %i[ mingw mswin x64_mingw jruby ]

    # Reduces boot times through caching; required in config/boot.rb
    gem "bootsnap", require: false

    # Use Sass to process CSS
    gem "sass-rails"

    group :development, :test do
      # See https://guides.rubyonrails.org/debugging_rails_applications.html#debugging-with-the-debug-gem
      gem "debug", platforms: %i[ mri mingw x64_mingw ]
    end

    group :development do
      # Use console on exceptions pages [https://github.com/rails/web-console]
      gem "web-console"
```

```
    gem "rails-erd"

    gem "railroady"
  end

  group :test do
    # Use system testing
[https://guides.rubyonrails.org/testing.html#system-testing]
    gem "capybara"
    gem "selenium-webdriver"
    gem "webdrivers"
  end

  gem "devise", "~> 4.8"




  Rails.application.routes.draw do
    get 'users/index'
    resources :companies
    resources :messages
    resources :rooms
    devise_for :users

    # Define your application routes per the DSL in
https://guides.rubyonrails.org/routing.html

    # Defines the root path route ("/")
    root "home#index"
```

```
end
```

ДОДАТОК Ґ

Лістинг представлення app/views/rooms/index.html.erb

```
<div class="container-fluid">
  <div class="row">
    <div class="col-3 rooms-sidebar">
      <div class="card no-outline">
        <div class="card-body">
          <small>Logged in as</small><br>
          <b><%= current_user.email.split('@')[0].capitalize
%><br></b>
          <%= button_to "Sign out ", destroy_user_session_path,
method: :delete , class: "btn btn-outline-primary" %>
          <% if current_user.admin %>
            <small>Admin</small><br>
            <%= link_to "Go to admin panel", root_path , class: "btn
btn-outline-primary" %>
          <% else %>
            <small>User</small>
          <% end %>
        </div>
      </div>
      <hr>
      <% if current_user.admin %>
        <%= render 'form', room: Room.new %>
      <% end %>
      <%- @rooms.each do |room| %>
        <%= link_to room, class: "room-link" do %>
          <% active_class = (@room == room) ? 'active' : '' %>
          <div class="card no-outline m-2 room-card <%= active_class
%>" >
            <div class="card-body">
```

```
                <span class="name">
                  <b><%= room.name %></b>
                </span><br>
                <span class="member-count">
                  <small><%= room.users.uniq.count %> Members</small>
                </span>
              </div>
            </div>
          <% end %>
        <% end %>
      </div>
      <div class="col-9">
        <% if @room.present? %>
          <div class="chat-room">
            <nav class="navbar navbar-light bg-light mb-4">
              <span class="navbar-brand" href="#">
                <b><p><%= @room.name %></p></b>
              </span>
              <h1 class="display-6"><%= @room.company.body %></h1>
              <small><%= @room.users.uniq.count %> Members</small>
            </nav>
            <div class="chat-block">
              <% @room.messages.each do |message| %>
                  <div class="message mb-2 <%= 'me' if message.user ==
current_user %> " id="message_<%= message.user.id %>" >
                    <div class="content-container">
                      <div class="content" >
                        <%= decrypt_helper(message.content) %>
                      </div>
                      <div class="author">
                        <%= message.user.email %>
```

```
                    </div>
                  </div>
                </div>
              <% end %>
            </div>
            <div class="chat-box">
              <%= render 'messages/form', message: Message.new, room:
@room %>
            </div>
          </div>
        <% end %>
      </div>
    </div>
</div>
```

ДОДАТОК Д

Лістинг представлення app/views/messages.html.rb

```
<div class="container">
<% if user_signed_in? %>
  <div class="alert alert-success top-block" role="alert">
  <h4 class="alert-heading"><%= current_user.email %>, welcome!</h4>
  <p class="mb-0">It's Nice To Meet You.</p>
  </div>
  <% if current_user.admin %>
    <div class="alert alert-warning" role="alert">
      <span>You are an administrator.</span>
    </div>
    <p>Users who have access to your company</p>

    <% @users = User.all %>
    <% @companies = Company.all %>

    <% @users.each do |user| %>
      <% if user.admin %>
        <div class="list-group-item list-group-item-primary d-
block">
          <%= user.email %><small> (Admin)</small>
        </div>
      <% elsif !user.admin %>
        <div class="list-group-item list-group-item-secondary">
          <%= user.email %>
        </div>
      <% end %>
    <% end %>
    <%= link_to 'Companies', companies_path %>
  <% end %>
```

```
  <%= button_to "Sign out", destroy_user_session_path, method:
:delete, class: "btn btn-warning" %>
<% else %>
  <%= button_to "Sign in", new_user_session_path %>
<% end %>

<% if user_signed_in? %>
  <%= link_to 'Rooms', rooms_path %>
<% end %>
</div>
```

ДОДАТОК Е

Лістинг файлу app/assets/stylesheets/custom.scss

```
body {
  overflow-y: hidden;
  ::-webkit-scrollbar {
    display: none;
  }
  -ms-overflow-style: none;
}

.chat-room {
  height: 100vh;

  .message {
    min-height: 59px;
    .content-container {
      display: inline-block;
      .content {
        background-color: #eeeeee;
        padding: 8px 16px;
        border-radius: 15px;
      }
      .author {
        font-size: 0.8rem;
        color: #777777;
        margin-left: 10px;
      }
    }
    &.me {
      .content-container {
        float: right;
```

```
        .content {
          background-color: #007bff;
          color: white;
        }
      }
    }
}

  .chat-box {
    position: absolute;
    bottom: 0;
    padding: 20px;
    width: 100%;
    margin-left: -15px;
    background-color: white;
    border-top: 1px solid #eeeeee;
    input[type=text] {
      height: 45px;
      font-size: 18px;
      padding: 8px;
    }
    .btn {
      height: 45px;
    }
  }
}

.card.no-outline {
  border: none;
}
```

```
.room-link {
  &:hover {
    text-decoration: none;
  }
  .room-card {
    transition: background-color 0.5s ease;
    &:hover {
      background-color: #eeeeee;
      border-radius: 20px;
    }
    &.active {
      background-color: black;
      .name {
        color: white;
      }
      .member-count {
        color: #e0e0e0;
      }
    }
    .name {
      font-weight: bold;
      color: black;
    }
    .member-count {
      color: #777777;
      font-weight: light;
    }
  }
}

.rooms-sidebar {
```

```
  border-radius: 1px solid #eeeeee;
}

img {
  opacity: 0.33;
  margin: 100px;
}

.d-block {
  display: block ruby!important;
}

h2 {
  &.title {
  padding: 30px;
} }

.control-me {
  display: none;
}

#toggle:checked ~ .control-me {
  display: block;
}

.checkbox {
  padding-top: 10px;
}

.box-company {
  padding: 10px;
```

```
}

.checkbox-done {
  display: block ruby;
}

.top-block {
  margin-top: 15px;
}
```

ДОДАТОК Є

Лістинг представлення app/views/messages/_form.html.erb

```
<%= form_with(model: message) do |form|%>
  <% if message.errors.any? %>
    <div style="color: red">
      <h2><%= pluralize(message.errors.count, "error") %> prohibited
this message from being saved:</h2>

      <ul>
        <% message.errors.each do |error| %>
          <li><%= error.full_message %></li>
        <% end %>
      </ul>
    </div>
  <% end %>
  <%= form.hidden_field :room_id, value: room.id%>

  <div class="input-group">
    <%= form.text_field :content, placeholder: "Type your message
here...",  class: "form-control mr-2" %>
    <div class="input-group-append">
      <%= form.submit "Add Message", class: "btn btn-primary" %>
    </div>
  </div>
<% end %>
```

ДОДАТОК Ж

Лістинг представлення app/views/companies/_form.html.erb

```
<%= form_with(model: company) do |form| %>
  <% if company.errors.any? %>
    <div style="color: red">
      <h2><%= pluralize(company.errors.count, "error") %> prohibited
this company from being saved:</h2>

      <ul>
        <% company.errors.each do |error| %>
          <li><%= error.full_message %></li>
        <% end %>
      </ul>
    </div>
  <% end %>

  <div class="container">
    <div class="form-group">
      <%= form.label :body, style: "display: block" %>
      <%= form.text_area :body %>
    </div>

    <div class="form-group">
      <%= form.submit %>
    </div>
  </div>
<% end %>
```

ДОДАТОК З

Лістинг контролера app/controllers/companies_controller.rb

```
class CompaniesController < ApplicationController
  before_action :set_company, only: %i[ show edit update destroy
]
  before_action :authenticate_user!, except: [:show, :index]

  # GET /companies or /companies.json
  def index
    @companies = Company.all
  end

  # GET /companies/1 or /companies/1.json
  def show
  end

  # GET /companies/new
  def new
    @company = Company.new
  end

  # GET /companies/1/edit
  def edit
  end

  # POST /companies or /companies.json
  def create
    @company = Company.new(company_params)

    @company.user = current_user
```

```
        respond_to do |format|
          if @company.save
            format.html { redirect_to company_url(@company), notice:
"Company was successfully created." }
            format.json { render :show, status: :created, location:
@company }
          else
            format.html { render :new, status: :unprocessable_entity
}
            format.json { render json: @company.errors, status:
:unprocessable_entity }
          end
        end
      end

      # PATCH/PUT /companies/1 or /companies/1.json
      def update
        respond_to do |format|
          if @company.update(company_params)
            format.html { redirect_to company_url(@company), notice:
"Company was successfully updated." }
            format.json { render :show, status: :ok, location:
@company }
          else
            format.html { render :edit, status: :unprocessable_entity
}
            format.json { render json: @company.errors, status:
:unprocessable_entity }
          end
        end
      end
```

```
        # DELETE /companies/1 or /companies/1.json
        def destroy
          @company.destroy

          respond_to do |format|
            format.html { redirect_to companies_url, notice: "Company
was successfully destroyed." }
            format.json { head :no_content }
          end
        end

        private
          # Use callbacks to share common setup or constraints between
actions.
          def set_company
            @company = Company.find(params[:id])
          end

          # Only allow a list of trusted parameters through.
          def company_params
            params.require(:company).permit(:body)
          end
      end
```

ДОДАТОК И

Лістинг контролера app/controllers/messages_controller.rb

```
class MessagesController < ApplicationController
  before_action :set_message, only: %i[ show edit update destroy ]

  # GET /messages or /messages.json
  def index
    @messages = Message.all
  end

  # GET /messages/1 or /messages/1.json
  def show
  end

  # GET /messages/new
  def new
    @message = Message.new

  end

  # GET /messages/1/edit
  def edit
  end

  # POST /messages or /messages.json
  def create
    criper = EncryptionServise.encrypt(message_params[:content])

    @message = Message.new(content: criper, room_id:
message_params[:room_id])
    @message.user = current_user
```

```
    @message.save
    redirect_to request.referrer
  end

  # PATCH/PUT /messages/1 or /messages/1.json
  def update
    respond_to do |format|
      if @message.update(message_params)
        format.html { redirect_to message_url(@message), notice:
"Message was successfully updated." }
        format.json { render :show, status: :ok, location: @message
}
      else
        format.html { render :edit, status: :unprocessable_entity }
        format.json { render json: @message.errors, status:
:unprocessable_entity }
      end
    end
  end

  # DELETE /messages/1 or /messages/1.json
  def destroy
    @message.destroy

    respond_to do |format|
      format.html { redirect_to messages_url, notice: "Message was
successfully destroyed." }
      format.json { head :no_content }
    end
  end
```

```
  private
    # Use callbacks to share common setup or constraints between
actions.
    def set_message
      @message = Message.find(params[:id])
    end

    # Only allow a list of trusted parameters through.
    def message_params
      params.require(:message).permit(:content, :user_id, :room_id)
    end
end
```