Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

ДОСЛІДЖЕННЯ КОМПОНЕНТНИХ МОДЕЛЕЙ ПРИ РЕАЛІЗАЦІЇ ПРОЄКТІВ З МЕТОЮ ПІДВИЩЕННЯ БЕЗПЕКИ
(тема)

Виконав:
студент 2 курсу, групи ІНФм-23-1
Бутенко П.В.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Кобилін О.А.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Бутенку Павлу Владленовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження компонентних моделей при реалізації проєктів з метою підвищення безпеки

затверджена наказом по університету від 25 листопада 2024 року № 1246Ст

2. Термін подання студентом роботи до екзаменаційної комісії 27 грудня 2024 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, мова програмування PHP, framework Laravel, мова програмування Javascript, framework Vue, framework Electron, проксі-сервер.

4. Перелік питань, що потрібно опрацювати в роботі

1. Аналіз існуючих підходів до забезпечення безпеки під час передачі даних.
2. Математичні основи шифрування даних.
3. Розробка компонентних моделей для реалізації проєктів.
4. Порівняльний аналіз застосунку з шифруванням даних з застосунком без шифрування.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми, постановка задачі, реалізація компонентних моделей, порівняння компонентних моделей.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	25.11.2024	
2	Аналіз завдання, підбір літератури	26.11.24-02.12.24	
3	Аналіз літератури з досліджуваної проблеми	03.12.24-07.12.24	
4	Аналіз технічних засобів і програмних засобів	08.12.24-11.12.24	
5	Розробка архітектури компонентних моделей	12.12.24-17.12.24	
6	Програмна реалізація	18.12.24-26.12.24	
7	Оформлення пояснювальної записки	27.12.24-02.01.25	
8	Перевірка на плагіат	03.01.2025	
9	Рецензування	04.01.2025	
10	Підготовка презентації та доповіді	05.01.25-06.01.25	
11	Занесення роботи в електронний архів	07.01.2025	
12	Попередній захист кваліфікаційної роботи	07.01.2025	

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Кобилін О.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 83 с., 2 табл., 15 рис., 1 дод., 41 джерела.

КОМПОНЕНТНА МОДЕЛЬ, БЕЗПЕКА, ПРОКСУВАННЯ, ШИФРУВАННЯ, ПОРІВНЯЛЬНИЙ АНАЛІЗ, КРОСПЛАТФОРМЕНІСТЬ, ВЕБЗАСТОСУНОК, ДЕСКТОП ЗАСТОСУНОК, REST API, AES, RSA, РОБОТА З ДАНИМИ.

Об’єктом дослідження є порівняння безпеки вебзастосунку та десктоп застосунку.

Метою дослідження є розробка та впровадження компонентних моделей, які забезпечують надійний захист даних у системах, враховуючи специфіку різних типів реалізацій і сучасні вимоги безпеки.

Використано мови програмування PHP та JavaScript, фреймворк Laravel, фреймворк Vue, фреймворк Electron. Проведено аналіз підходів до захисту даних під час передачі через REST API, включаючи використання гібридного шифрування на основі алгоритмів AES та RSA, а також інтеграцію проксі-сервера для додаткового захисту даних.

У результаті дослідження реалізовано компонентні моделі для забезпечення безпечної передачі та зберігання даних.

COMPONENT MODEL, SECURITY, PROXYING, ENCRYPTION, BENCHMARKING, CROSS-PLATFORM, WEB APPLICATION, DESKTOP APPLICATION, REST API, AES, RSA, DATA HANDLING.

The object of the research is to compare the security of a web application and a desktop application.

The aim of the study is to develop and implement component models that provide reliable data protection systems, taking into account the specifics of different types of implementations and modern security requirements.

The programming languages used are PHP and JavaScript, Laravel framework, Vue framework and Electron framework. The approaches to data protection during transmission via the REST API were analysed, including the use of hybrid encryption based on AES and RSA algorithms, as well as the integration of a proxy server for additional data protection.

The result of the aim, component models were implemented to ensure secure data transmission and storage.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface (інтерфейс програмування застосунків)

JWT – JSON Web Token (вебтокен JSON)

DevTools – Development Tools (інструменти розробки)

REST – Responsibility State Transfer (передача репрезентативного стану)

CORS – Cross Origin Resource Sharing (спільне використання ресурсів з різних джерел)

SQL – Structured Query Language (структурована мова запитів)

XSS – Cross Site Scripting (міжсайтовий скриптинг)

CSRF – Cross Site Request Forgery (підробка міжсайтових запитів)

AES – Advanced Encryption Standard (розширений метод шифрування)

DES – Data Encryption Standard (стандарт шифрування даних)

XOR – Exclusive OR (виняткове або)

ORM - Object-Relational Mapping (об’єктно-реляційне відображення)

HTTP - HyperText Transfer Protocol (протокол передачі гіпертексту)

HTML – HyperText Markup Language (мова розмітки гіпертексту)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

RSA – Rivest, Shamir, Adleman (Рівест, Шамір, Адлеман)

IV – Initialization Vector (вектор ініціалізації)

OAEP – Optimal Asymmetric Encryption Padding (оптимальне асиметричне шифрування)

CBC – Cipher Block Chaining (ланцюжок блоків шифрування)

ВСТУП

У сучасному інформаційному просторі безпека застосунків стає дедалі важливішою. Сучасні вебінтерфейси та десктопні програми вимагають ефективних рішень для захисту даних і забезпечення надійної комунікації між компонентами систем. Тому дослідження компонентних моделей у процесі розробки проєктів для підвищення безпеки є критичним завданням, що потребує особливої уваги.

Проблематика, розглянута в кваліфікаційній роботі, пов’язана з необхідністю покращення безпеки застосунків через ефективне використання компонентних моделей. Компонентна архітектура дозволяє розробникам створювати більш гнучке та модульне середовище для розробки систем, що сприяє кращому управлінню безпекою даних і зменшенню ризиків, пов’язаних із вразливостями системи.

Це дослідження має високу практичну цінність, оскільки його результати можуть служити основою для розробки більш безпечних застосунків. Запропоновані підходи та методи спрямовані на покращення управління безпекою даних і зниження ризиків, пов’язаних із вразливостями у компонентних моделях.

Актуальність дослідження полягає у зростаючій потребі забезпечення високого рівня безпеки застосунків через ефективне впровадження компонентних моделей у розробку систем.

1 АНАЛІЗ КОМПОНЕНТНИХ МОДЕЛЕЙ ТА ЇХ РОЛЬ У ЗАБЕЗПЕЧЕННІ БЕЗПЕКИ

1.1 Огляд компонентних моделей та їх застосування

Компонентні моделі є важливим аспектом сучасної програмної інженерії, забезпечуючи основи для створення модульних і масштабованих програмних систем. Вони дозволяють розробляти програмне забезпечення шляхом об’єднання окремих автономних компонентів, кожен з яких виконує певну функцію або набір функцій. Цей підхід сприяє підвищенню гнучкості розробки, спрощує процеси підтримки та оновлення систем, а також дозволяє легше інтегрувати нові функціональні можливості.

Основна концепція компонентних моделей полягає в тому, що система складається з декількох незалежних компонентів, які можуть взаємодіяти між собою через чітко визначені інтерфейси. Кожен компонент є самодостатнім модулем з певним набором функцій і даних, які можуть бути використані іншими компонентами в системі. Це дозволяє розробникам створювати програмні продукти, які легко змінюються і масштабуються без значних зусиль [1, 2].

Застосування компонентних моделей є поширеним у багатьох сферах програмування. Вони використовуються для розробки як вебзастосунків, так і десктопних програм, де важливо забезпечити модульність, гнучкість і можливість масштабування. Наприклад, у веброзробці компоненти можуть відповідати за управління користувачами, обробка запитів або управління базами даних. У десктопних програмах компоненти можуть охоплювати такі функції, як обробка даних, взаємодія з апаратним забезпеченням або інтерфейс користувача.

Компонентні моделі забезпечують значні переваги, такі як підвищення надійності та зменшення витрат на розробку та підтримку програмного забезпечення. Вони сприяють створенню легко підтримуваних і

розширюваних систем, дозволяючи додавати нові функціональні можливості без необхідності значних змін у базовій архітектурі програми. Завдяки цьому підходу можна легше адаптувати програмне забезпечення до змінних вимог бізнесу та технологій, а також підвищити рівень безпеки системи шляхом ізоляції різних компонентів [3].

Використання компонентних моделей також допомагає знизити складність розробки великих систем, розподіляючи завдання на менші, більш керовані частини. Це не тільки спрощує процес розробки, але й дозволяє розробникам зосередитися на конкретних аспектах функціональності та покращити якість програмного забезпечення.

Крім того, компонентні моделі є важливими для забезпечення безпеки застосунків. Завдяки чіткій структурі компонентів і стандартам взаємодії між ними, можна впроваджувати заходи безпеки на рівні кожного окремого компонента, що забезпечує більш високий рівень захисту даних і зменшує ризики уразливостей. Це особливо актуально в умовах сучасної кібербезпеки, де кожен компонент може бути потенційною точкою атаки.

Отже, компонентні моделі продовжують відігравати важливу роль у сучасній розробці програмного забезпечення. Вони дозволяють створювати більш надійні, гнучкі і масштабовані системи, що відповідають вимогам різних галузей і ринків [4]. З огляду на це, дослідження компонентних моделей і їх застосування в контексті забезпечення безпеки застосунків є актуальним і важливим напрямком для подальших розробок.

1.2 Аналіз існуючих підходів до забезпечення безпеки

Компонентні моделі є основою багатьох сучасних програмних систем, дозволяючи розробляти гнучкі, масштабовані і легко підтримувані програми. Проте, розподілений характер компонентних систем також створює нові виклики щодо забезпечення безпеки. Оскільки компоненти можуть

взаємодіяти між собою через стандартизовані інтерфейси, важливо забезпечити, щоб ці взаємодії були захищені від потенційних загроз, таких як несанкціонований доступ, витоки даних або модифікація інформації.

Аналіз існуючих підходів до забезпечення безпеки в компонентних моделях дозволяє виявити найбільш ефективні методи і стратегії, які використовуються для захисту даних і забезпечення надійності системи [5]. Ці підходи можуть варіюватися від традиційних методів контролю доступу та аутентифікації до більш сучасних технік захисту, таких як шифрування, цифрові підписи і моніторинг активності. Дослідження існуючих підходів до безпеки допомагає зрозуміти, як можна мінімізувати ризики і підвищити загальний рівень захисту застосунків на базі компонентних моделей.

1.2.1 Основні концепції та принципи забезпечення безпеки в компонентних системах

Для забезпечення безпеки компонентних систем використовуються декілька основних концепцій, які є фундаментальними для створення захищених програм. Ці концепції включають захист конфіденційності, цілісності та доступності даних, а також впровадження механізмів аутентифікації, авторизації та управління доступом.

Конфіденційність є однією з ключових концепцій забезпечення безпеки. Вона передбачає, що дані повинні бути захищені від несанкціонованого доступу або розголошення.

Цілісність даних є ще однією важливою концепцією, яка гарантує, що дані залишаються незмінними і достовірними під час зберігання і передачі. Для забезпечення цілісності використовуються методи хешування і цифрові підписи, що дозволяють виявляти будь-які спроби несанкціонованих змін. У контексті компонентних моделей цілісність даних має бути забезпечена як на рівні окремих компонентів, так і на рівні всієї системи [6, 7].

Аутентифікація та авторизація є невід'ємними частинами процесу захисту компонентних систем. Аутентифікація підтверджує особу користувача, а авторизація визначає, які ресурси і дії доступні користувачу після його аутентифікації. Це дозволяє забезпечити, щоб доступ до системи мали тільки ті користувачі, які мають на це право, і тільки до тих компонентів і функцій, до яких їм дозволено доступ.

Управління доступом – це процес встановлення і контролю доступу користувачів і компонентів до ресурсів і даних системи. Це включає визначення чітких політик доступу, які регулюють взаємодію між різними частинами системи і допомагають запобігти несанкціонованому доступу до чутливих даних.

Останнім важливим аспектом є виявлення та реагування на інциденти безпеки. Це включає моніторинг активності системи, логування дій користувачів і компонентів, а також впровадження механізмів для швидкого виявлення і реагування на загрози. Вчасне виявлення підозрілих дій дозволяє мінімізувати шкоду від атак і швидко відновити систему до нормальної роботи.

1.2.2 Розгляд сучасних методів і практик забезпечення безпеки в вебінтерфейсах та десктопних застосунках

Безпека є важливим аспектом у розробці систем, особливо коли вони побудовані на компонентних моделях із використанням сучасних технологій.

Для вебзастоснків, що використовують Laravel як API сервер і Vue як вебклієнт, існує ряд сучасних методів і практик забезпечення безпеки:

Використання JWT для аутентифікації та авторизації. JWT є стандартом для створення токенів доступу, які використовуються для забезпечення аутентифікації та авторизації користувачів у вебінтефейсах [8]. У випадку з Laravel, JWT часто використовується для передачі інформації між клієнтом та

сервером, що дозволяє безпечно зберігати дані про сеанси користувачів і знижувати ризик несанкціонованого доступу. JWT забезпечує безпеку завдяки криптографічному підпису, що дозволяє впевнитися в достовірності даних токена і його незмінності.

Використання проксі-серверів. Проксі-сервери використовуються для додаткового рівня захисту між клієнтом і сервером. Вони можуть обробляти запити, маскуючи справжнє місцезнаходження серверів і приховуючи внутрішню структуру мережі [9, 10]. Це знижує ризик прямого доступу до серверів Laravel, забезпечуючи додатковий рівень безпеки для застосунків.

Приховування і маніпулювання запитами. У вебсистемах важливо забезпечити захист від маніпуляцій із запитами, які можуть бути легко видимими в DevTools браузера. Одним з методів є використовувати механізми для приховання API запитів, так щоб важлива інформація або механізми аутентифікації були менш очевидними для зовнішнього користувача.

Розподілена архітектура і захист на рівні API. Laravel, використовуючи REST API для обміну даними з клієнтами дозволяє впроваджувати політики безпеки на рівні API. Це включає обмеження частоти запитів, контроль доступу на основі ролей, а також впровадження додаткових засобів безпеки, таких як CORS, для контролю того, які домени можуть взаємодіяти з API.

Захист від атак через сторонні модулі. Electron застосунки можуть включати багато сторонніх модулів і бібліотек, що може стати вразливістю. Важливо регулярно перевіряти ці модулі на предмет безпеки та оновлювати їх до останніх версій.

Компонентний підхід дозволяє ефективно використовувати різні методи і практики безпеки, розділяючи застосунок на окремі модулі, кожен з яких має свій рівень захисту. Це підвищує гнучкість і стійкість системи, дозволяючи швидко адаптуватися до нових загроз і забезпечити захист даних та взаємодії між компонентами. Інтеграція безпеки на рівні компонентів також сприяє кращій ізоляції потенційних загроз і спрощує процеси виявлення та реагування на інциденти.

1.2.3 Порівняльний аналіз підходів до безпеки в різних компонентних моделях

Компонентні моделі відіграють ключову роль у сучасній розробці програмного забезпечення, дозволяючи розробляти системи, які можуть бути легко масштабовані, підтримувані та адаптовані до змін. Водночас різні підходи до реалізації компонентних моделей мають свої унікальні переваги та недоліки щодо забезпечення безпеки. Розглянуто порівняльний аналіз підходів до безпеки в різних компонентних моделях, включаючи моделі, використані в дослідженні вебзастосунків і десктопних систем.

Вебінтерфейси та десктопні програми мають різні архітектурні вимоги і особливості, що впливає на вибір підходів до безпеки. Вебсистеми, зважаючи на їхню відкриту природу і доступність через Інтернет, зазвичай стикаються з більшою кількістю загроз, ніж десктопні. Це включає такі загрози, як SQL ін’єкції, XSS атаки, CSRF атаки, перехоплення трафіку. Для захисту від цих загроз у вебінтерфейсах зазвичай використовуються такі підходи, як шифрування трафіку, використання токенів безпеки, а також захист на рівні API.

З іншого боку, десктопні програми, побудовані на технологіях, таких як Electron, взаємодіють із серверною частиною через локальні або внутрішні мережі, що знижує ризик певних видів атак, характерних для вебінтерфейсах. Однак десктопні застосунки також мають свої уразливості, такі як загрози, пов’язані з доступом до локальних даних, або атаки через експлуатацію вразливостей у сторонніх бібліотеках і модулях. Для захисту десктопних систем часто застосовуються методи шифрування локальних даних, цифрового підпису коду, а також ізоляція процесів і обмеження прав доступу.

У контексті вебінтерфейсів та десктопних програм на основі компонентних моделей безпека часто реалізується на рівні API. Це включає використання аутентифікації та авторизації через JWT, контроль доступу на основі ролей, а також захист від атак типу brute force і DDoS. Цей підхід

дозволяє централізовано керувати безпекою, знижуючи ризики, пов’язані з розподіленими системами.

Кожен з підходів до безпеки в компонентних моделях має свої переваги і недоліки. Для вебзастосунків основною перевагою є можливість централізованого управління безпекою на рівні API, що дозволяє швидко реагувати на нові загрози і впроваджувати необхідні заходи захисту. Однак це також може створити точку відмови, оскільки компрометація API сервера може призвести до втрати доступу до всіх ресурсів.

Десктопні системи, навпаки, зазвичай менш вразливі до загроз, характерних для вебпрограм, оскільки вони працюють у більш контрольованому середовищі. Однак безпека десктопних програм часто залежить від оновлення і підтримки локальних механізмів захисту, таких як антивірусні програми і шифрування даних, що може створювати додаткові виклики в управлінні.

Порівняльний аналіз підходів до безпеки показує, що обидва типи застосунків мають свої унікальні виклики та вимоги до забезпечення безпеки. Вибір конкретного підходу до безпеки залежить від типу програми, його архітектури, середовища виконання і вимог до захисту даних. Компонентні моделі дозволяють впроваджувати безпеку на різних рівнях, що забезпечує більш гнучкий і адаптивний підхід до захисту сучасних застосунків.

Загалом, забезпечення безпеки в компонентних моделях вимагає комплексного підходу, що включає використання сучасних технологій і практик, а також постійного моніторингу та вдосконалення захисних механізмів. Це дозволяє створювати системи, які не лише відповідають сучасним вимогам безпеки, але й здатні швидко адаптуватися до нових загроз і викликів.

1.3 Аналіз сучасних тенденцій

1.3.1 Огляд сучасних технологій та інструментів для розробки безпечних компонентних моделей

Забезпечення безпеки в компонентних моделях – ключовий аспект сучасної розробки програмного забезпечення. Для створення надійних застосунків використовуються різні технології та інструменти, що допомагають мінімізувати ризики та забезпечити захист даних і компонентів.

Laravel пропонує широкий набір вбудованих функцій для захисту застосунків. Це включає захист від поширених атак, таких як SQL ін’єкції і CSRF, а також підтримку аутентифікації і шифрування. Завдяки цим можливостям Laravel забезпечує безпеку даних і управління доступом.

Vue дозволяє створювати компонентні інтерфейси користувача з вбудованими модулями захисту. Підтримка розмежування доступу до різних частин програми і захист від XSS забезпечують безпечне виконання коду на стороні клієнта.

Використання Electron для створення десктопних застосунків забезпечує контроль над доступом до системних ресурсів і управління безпекою на локальному рівні. Це досягається за допомогою налаштувань прав доступу і обмеження можливостей взаємодії з операційною системою.

JWT дозволяє реалізувати безпечну аутентифікацію і авторизацію, зберігаючи дані користувача в захищеному вигляді. Цей інструмент часто використовується для обміну інформацією між клієнтом і сервером в безпечному форматі.

Проксі-сервери використовується для захисту серверної частини, обробки запитів і контролю трафіку. Проксі сервери дозволяють маскувати архітектуру і запобігати атакам, зберігаючи цілісність мережевої взаємодії.

Використання криптографічних алгоритмів для шифрування даних як на сервері, так і на клієнті забезпечує захист конфіденційної інформації від несанкціонованого доступу.

1.3.2 Використання компонентних моделей у контексті безпеки даних і інформаційної безпеки

Компонентні моделі в сучасному програмному забезпеченні сприяють підвищенню безпеки даних і інформаційної безпеки завдяки своїй модульній архітектурі. Вони дозволяють реалізовувати різні заходи захисту на рівні окремих компонентів, забезпечуючи гнучкість та ефективність в управлінні безпекою [11].

Компонентні моделі забезпечують ізоляцію даних і функцій, розділяючи програмне забезпечення на окремі модулі, які взаємодіють через чітко визначені інтерфейси. Такий підхід дозволяє знижувати ризики, пов’язані з компрометацією окремих частин системи. У застосунках, що використовують API сервер на базі Laravel, кожен компонент може мати свої власні правила доступу до даних, обмежуючи можливості несанкціонованого доступу.

Компонентні моделі дозволяють ефективно реалізовувати контроль доступу і аутентифікацію, розділяючи права доступу до даних і функцій між різними компонентами системи. Це включає використання токенів безпеки, таких як JWT, для аутентифікації користувачів і забезпечення захищеного доступу до ресурсів. Кожен компонент може мати свою власну систему управління доступом, що підвищує загальний рівень безпеки системи.

Використання компонентних моделей також дозволяє підвищити безпеку передачі даних між компонентами. Це досягається через шифрування даних на рівні транспортного протоколу і використання захищених каналів зв’язку. Додатково, кожен компонент може реалізовувати свої механізми захисту, включаючи шифрування даних перед їх зберіганням або передачею, що зменшує ризик втрати або крадіжки даних [12].

Компонентний підхід дозволяє розробникам легше інтегрувати інструменти моніторингу і виявлення загроз на рівні кожного компонента. Це забезпечує можливість швидкої ідентифікації потенційних загроз і реагування на них, знижуючи ризики та підвищуючи безпеку застосунка.

1.3.3 Тенденції та виклики використання компонентних моделей у розробці безпечних проєктів

Компонентні моделі набувають все більшого поширення в розробці програмного забезпечення завдяки своїм перевагам у забезпеченні гнучкості та масштабованості. Однак сучасні реалії безпеки вимагають нових підходів і рішень для захисту програм, що побудовані на компонентній архітектурі.

Використання хмарних платформ як AWS, Azure або Google Cloud для розгортання компонентів дозволяє забезпечити додаткову гнучкість та швидкість масштабування. Це вимагає реалізації додаткових заходів безпеки для захисту даних і контролю доступу в розподілених середовищах, що розширює можливості компонентних моделей у контексті безпеки.

Перехід на моделі безпеки, що не довіряють жодному компоненту за замовчуванням, стає стандартом. Компонентні моделі природньо підтримують цей підхід через можливість впровадження багаторівневого контролю доступу та обов’язкової аутентифікації між компонентами [13]. Це дозволяє зменшити ризики несанкціонованого доступу і знизити ймовірність внутрішніх загроз.

Зростає популярність автоматизованих систем для тестування безпеки компонентних модулів. Використання інструментів для автоматизованого виявлення уразливостей і аналізу коду дозволяє швидко виявляти потенційні проблеми і усувати їх на етапі розробки, що суттєво знижує ризик виникнення вразливостей у продуктивному середовищі.

Кожен компонент може мати свої власні залежності та вимоги, що може призводити до конфліктів і ускладнень при оновленні або заміні компонентів. Невідповідність версій або невчасні оновлення можуть створювати додаткові вразливості, які зловмисники можуть використати для атак.

Розподілений характер компонентних моделей вимагає впровадження засобів для захисту даних як у відправленому стані, так і в спокої. Це потребує інтеграції шифрування на різних рівнях і забезпечення безпеки між компонентами, що обмінюються даними.

Масштабованість компонентних моделей повинна поєднуватися з високим рівнем безпеки. Забезпечення надійної роботи системи при збільшенні кількості компонентів або ресурсів стає важливим викликом. Це вимагає вдосконалених механізмів для моніторингу безпеки і автоматизованого управління ризиками в умовах зростання навантаження.

Впровадження ефективних механізмів управління доступом стає більш складним завданням у компонентних моделях, де кожен компонент може мати різні вимоги до безпеки. Це вимагає створення єдиних політик і стандартів для управління доступом і моніторингу активності, що може бути складним у великих розподілених системах.

1.4 Постановка задачі дослідження

При розробці компонентних моделей для забезпечення безпеки у сучасних програмах необхідно вирішити низку завдань, спрямованих на створення надійних та безпечних систем. Дослідження фокусується на двох реалізаціях: вебзастосунку та десктопному застосунку

Перший етап дослідження полягає у створенні архітектури обох програм, яка забезпечить безпечну взаємодію між компонентами. Для вебзастосунку необхідно налаштувати API сервер на Laravel з підтримкою безпечного обміну даними через REST API. Водночас, для десктопного застосунку важливо поєднати можливості Electron для створення десктоп інтерфейсу з доступом до API Laravel, забезпечуючи при цьому безпеку обміну даними.

Другий етап передбачає реалізацію заходів безпеки у кожній із реалізацій. Налаштування аутентифікації та авторизації, управління доступом до даних, впровадження політик безпеки на рівні кожного компонента.

Третій етап стосується інтеграції засобів захисту для десктоп системи. Це передбачає використання проксі-серверів для захисту серверної частини.

Останній етап дослідження передбачає аналіз результатів. На основі отриманих даних необхідно оцінити ефективність реалізованих заходів безпеки, виявити слабкі місця та запропонувати шляхи їх покращення. Це може включати впровадження нових технологій або покращення існуючих методів захисту.

Таким чином, основне завдання дослідження полягає в розробці та оцінці компонентних моделей, що здатні забезпечити високий рівень безпеки у застосунках.

Об’єктом дослідження є порівняння безпеки вебзастосунку та десктоп застосунку.

Метою дослідження є розробка та впровадження компонентних моделей, які забезпечують надійний захист даних у системах, враховуючи специфіку різних типів реалізацій і сучасні вимоги безпеки.

Для досягнення мети необхідно вирішити такі завдання:

– розробити архітектуру для обох типів застосунків, яка забезпечить безпечну взаємодію між компонентами;

– реалізувати заходи безпеки для захисту даних і управління доступом;

– інтегрувати засоби захисту для забезпечення безперебійної роботи і мінімізації ризиків;

– провести тестування і оцінку безпеки розроблених систем для виявлення вразливостей.

2 АРХІТЕКТУРА ТА МАТЕМАТИЧНІ ОСНОВИ СИСТЕМИ БЕЗПЕКИ

2.1 Математичне моделювання вимог безпеки

Забезпечення безпеки інформаційних систем є ключовим аспектом сучасних технологій. У світі, де обсяг даних постійно зростає, а кіберзагрози стають все більш складними та різноманітними, важливо розробляти системи, здатні ефективно захищати інформацію від несанкціонованого доступу, модифікації та інших форм атак. Математичне моделювання вимог безпеки дозволяє формалізувати та аналізувати ці вимоги, забезпечуючи надійність та стійкість інформаційних систем до різноманітних загроз [14].

В процесі розробки безпечної системи необхідно враховувати три основні аспекти: конфіденційність, цілісність та доступність даних. Кожен аспект має свої специфічні вимоги та механізми забезпечення, які можна моделювати для оцінки ефективності. Математичне моделювання дозволяє не лише визначити необхідні параметри для захисту, але й прогнозувати потенційні ризики та визначати оптимальні стратегії їх мінімізації.

Крім того, математичне моделювання вимог безпеки сприяє системному підходу до розробки безпечних систем, забезпечуючи чітку структуру та методологію для оцінки та впровадження заходів безпеки. Це дозволяє створювати системи, які не тільки відповідають поточним вимогам безпеки, але й здатні адаптуватися до нових загроз та викликів.

Конфіденційність даних полягає в забезпеченні того, що інформація доступна лише уповноваженим користувачам і недоступна для несанкціонованих суб’єктів [15, 16]. Математично конфіденційність можна формалізувати за допомогою моделей контролю доступу (2.1).

$$A(u) = \begin{cases} 1, & \text{якщо } u \text{ має право доступу,} \\ 0, & \text{в іншому випадку.} \end{cases}, \qquad (2.1)$$

де $A(u)$ – функції доступу;

u – корисувач.

Для забезпечення конфіденційності використовуються криптографічні алгоритми, які перетворюють відкриті дані у зашифрований формат, недоступний для розуміння без відповідного ключа. Процес шифрування даних з використанням ключа описується функцією шифрування, також при шифрування даних необхідним є і зворотній процес розшифрування:

$$C = E_K(M),\ M = D_K(C), \tag{2.2}$$

де C – зашифровані дані;

K – ключ шифрування;

M – вхідні дані;

E – шифрування;

D – розшифрування.

Забезпечення конфіденційності досягається, якщо без ключа неможливо відновити повідомлення (рис. 2.1). Це гарантує, що навіть при перехопленні зашифрованих даних зловмисник не зможе отримати доступ до їх змісту.

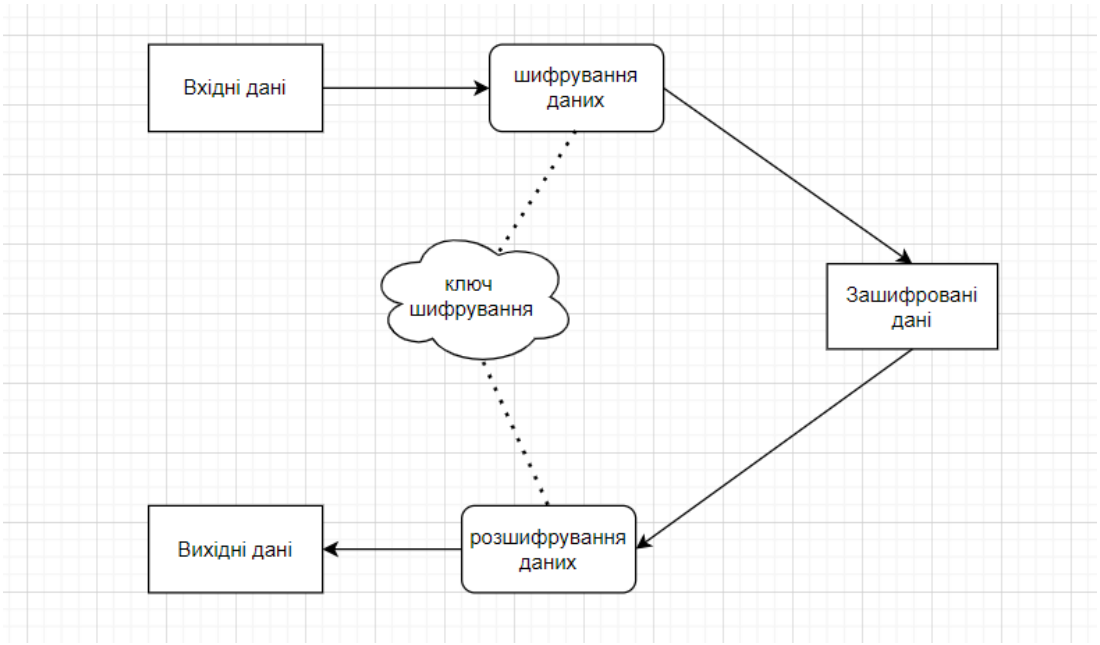

Рисунок 2.1 – процес шифрування та розшифрування даних

Цілісність гарантує, що інформація не була змінена або пошкоджена під час передачі чи зберігання. У системі цілісність даних забезпечується через процеси шифрування та розшифрування, які мають внутрішні механізми перевірки коректності даних. Під час розшифрування даних за допомогою функції здійснюється перевірка коректності ключа та відповідності структури даних. Якщо дані змінені або пошкоджені, процес розшифрування завершиться помилкою показано на рисунку 2.2.

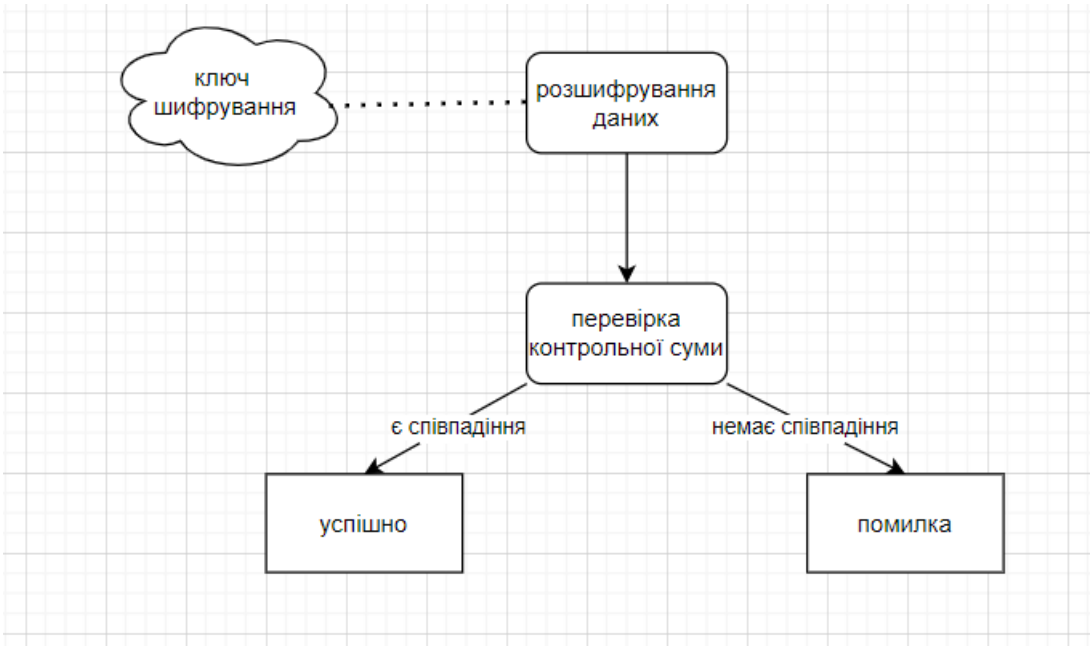

Рисунок 2.2 – Процес перевірки цілісності при розшифруванні

Цілісність даних гарантується тим, що будь-які зміни в зашифрованих даних призводять до невдалого процесу розшифрування, що дозволяє виявити модифікацію даних. Основними властивостями механізмів забезпечення цілісності даних є здатність виявляти будь-які зміни в зашифрованих даних. Якщо дані були змінені, то при спробі зчитування буде отримана помилка.

Доступність системи означає, що ресурси системи доступні уповноваженим користувачам у потрібний момент часу [17–19]. Це можна моделювати за допомогою коефіцієнта готовності системи, який визначається як відношення часу працездатності системи до суми часу працездатності разом з часом простою:

$$A = T_{\text{роб}}/T_{\text{роб}} + T_{\text{відмов}}, \quad (2.3)$$

де A – коефіцієнт готовності системи;

$T_{\text{роб}}$ – час працездатності;

$T_{\text{відмов}}$ – час простою.

Підвищення доступності досягається шляхом використання відмовостійких рішень, резервування компонентів та регулярного обслуговування системи. Це дозволяє мінімізувати час простою та забезпечити безперервний доступ до ресурсів.

2.2 Математичні основи шифрування даних

Шифрування даних є одним із ключових механізмів забезпечення безпеки інформаційних систем. Воно дозволяє перетворювати відкриту інформацію у захищений формат, який недоступний для несанкціонованих осіб. Основною метою шифрування є забезпечення конфіденційності, цілісності та автентичності даних під час їх передачі та зберігання. Математичні основи шифрування визначають ефективність та надійність використовуваних алгоритмів, що є критично важливим для захисту інформації від різноманітних загроз та атак.

Існують два основні типи шифрування: симетричне та асиметричне. Симетричне шифрування використовує один і той же ключ для процесів шифрування та розшифрування даних, що робить його швидким та ефективним для обробки великих обсягів інформації. Одним із найбільш поширених симетричних алгоритмів є AES, який забезпечує високий рівень безпеки та широко застосовується в різних сферах, від державних установ до комерційних організацій [20].

Асиметричне шифрування навпаки, використовує пару ключів: відкритий та закритий. Відкритий ключ доступний для всіх, тоді як закритий

залишається конфіденційним для власника. Цей підхід вирішує проблему безпечного обміну ключами, характерну для симетричного шифрування, та дозволяє реалізувати додаткові функції, такі як цифрові підписи та безпечний обмін даними. RSA є одним із найбільш відомих асиметричних алгоритмів, який широко використовується для захисту інформації в Інтернеті.

Математичні принципи, на яких базуються симетричні та асиметричні алгоритми шифрування, значно впливають на їхню криптостійкість та ефективність. Симетричні алгоритми, такі як AES, покладаються на складні операції над блоками даних та ключами, що ускладнює їхній аналіз та зламування без відома ключа. Асиметричні алгоритми, як RSA, базуються на вирішенні складних математичних задач, таких як факторизація великих простих чисел, що робить їх надзвичайно важкими для вирішення з використанням сучасних методів криптоаналізу. Вибір відповідного алгоритму шифрування залежить від конкретних вимог до безпеки та продуктивності системи, що дозволяє забезпечити баланс між рівнем захисту та ефективністю роботи інформаційної системи.

2.2.1 Симетричне шифрування з використанням AES

Advanced Encryption Standard є одним із найпоширеніших симетричних алгоритмів шифрування, стандартизованих Національним інститутом стандартів і технологій США у 2001 році [21]. AES був обраний як заміна попереднього стандарту DES завдяки своїй високій криптостійкості, ефективності та гнучкості.

Основана мета впровадження AES полягає в забезпеченні надійного захисту інформації, що передається або зберігається, за допомогою використання одного і того ж симетричного ключа як для процесу шифрування, так і для розшифрування даних. Завдяки цьому, якщо ключ буде належним чином прихований і захищений, сторонній клієнт не зможе

отримати доступ до вихідної інформації. При цьому важливо, щоб ключ тільки у сервера та клієнта, оскільки тоді таємність повідомлень буде порушено.

AES працює з блоками даних розміром 128 біт та підтримує три різні довжини ключів: 128, 192 та 256 біт. Кожна з цих конфігурацій визначає кількість раундів шифрування: 10, 12 та 14 відповідно. Всі варіанти AES базуються на тій самій основній структурі, яка включає серію перетворень, спрямованих на ускладнення аналізу та зламування шифру.

Основною математичною структурою AES є операції над полями Галуа, що забезпечують виконання як нелінійних, так і лінійних трансформацій над даними. Ключові компоненти AES включають наступні етапи:

– нелінійна підстановка;
– лінійна операція;
– матрична операція;
– операція додавання ключа.

Нелінійна підстановка кожного байта стану виконується за допомогою I-блоку, який забезпечує дифузію та замінює кожен байт на нове значення, роблячи зв’язок між вхідними й вихідними бітами складним для аналізу.

Лінійна операція зміщує байти кожного рядка матриці стану на певну кількість позицій, додаючи випадкову перестановку даних.

Матрична операція виконує змішування байтів стовпців матриці стану, забезпечуючи ще більшу дифузію, щоб навіть незначні зміни ключа чи вхідних даних кардинально впливали на підсумковий зашифрований текст.

По завершенню кожного раунду виконується XOR операція. Раундові ключі генеруються завдяки процедурі розширення первинного ключа. Додавання ключа на кожному етапі гарантує, що без відповідного ключа неможливо повернутися від зашифрованих даних до вхідних даних.

Для кращого розуміння взаємодії цих етапів в алгоритмі AES розглянуто загальну структуру, представлену на рисунку 2.3, яка включає послідовність раундів та основні операції, що виконуються завжди на кожному етапі процесу шифрування.

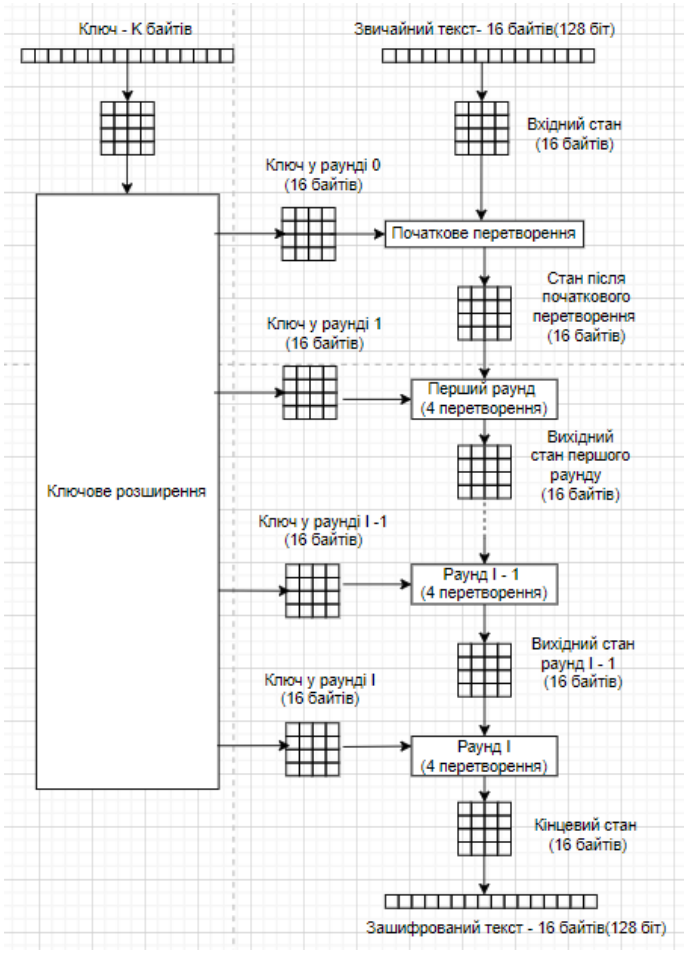

Рисунок 2.3 – Загальна структура AES

Однією з ключових особливостей AES є використання I-блоку, який є нелінійною функцією підстановки. Даний блок забезпечує дифузію та замінює кожен байт стану на нове значення, яке є функцією оригінального значення та поля Галуа [22]. Операція змішування колонок виконується шляхом множення стовпців матриці стану на фіксовану матрицю в полі Галуа. Це забезпечує лінійну дифузію даних. Множення забезпечує змішування байтів стовпців, що сприяє високій стійкості алгоритму (2.4).

$$MixColumns = (I_0, I_1, I_2, I_3) = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} I_0 \\ I_1 \\ I_2 \\ I_3 \end{pmatrix}, \qquad (2.4)$$

де I_i - байти стовпця матриці стану.

Для кращого розуміння механізму змішування колонок детально розглянуто матричне множення, яке показано на рисунку 2.4.

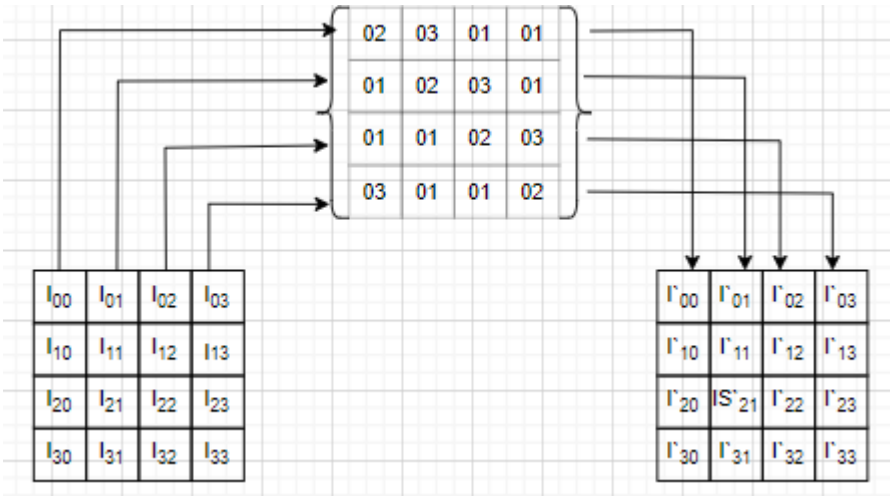

Рисунок 2.4 – Матричне множення в операції змішування колонок

Процес шифрування починається з додавання ключа початкового раунду до стану через операцію додавання ключа. Потім слідує серія раундів, кожен з яких включає операції: підстановки байтів, зміщення рядків, змішування колонок та додавання ключа раунду. Кількість раундів залежить від довжини ключа:

– 10 раундів у 128 бітного;
– 12 раундів у 192 бітного;
– 14 раундів у 256 бітного.

Останній раунд відрізняється від попередніх тим, що операція змішування колонок не включена у процес шифрування. Це спрощує процес розшифрування, зберігаючи при цьому високу стійкість алгоритму. Розшифрування відбувається у зворотному порядку блоку, використовуючи зворотні операції:

– додавання ключа раунду;
– зворотне змішування колонок;
– зворотне зміщення рядків;
– зворотна підстановка байтів.

Безпека AES забезпечується кількома ключовими аспектами. По-перше, використання I-блоку вводить нелінійність у алгоритм, що ускладнює диференційний та лінійний криптоаналіз. По-друге, операції зміщення рядків та змішування колонок забезпечують широку дифузію даних, поширюючи зміни одного байту по всій матриці стану. Це зменшує кореляції між відкритим та зашифрованим текстами, ускладнюючи спроби криптоаналізу алгоритму. По-третє, кожен раунд використовує унікальний ключ раунду, що генерується за допомогою ключового розширення. Це ускладнює прогнозування ключів навіть при відомості одного з ключів раунду, забезпечуючи стійкість до атак на основний ключ. Правильна реалізація AES повинна враховувати захист від атак на побічні канали, таких як таймінгові атаки або атаки через вимірювання споживаної енергії [23]. Це досягається шляхом використання сталих часових затримок та уніфікованих операцій шифрування, що мінімізує можливість виявлення інформації про ключ через побічні канали.

Процес генерації ключів раунду в AES є критично важливим. Початковий ключ розширюється до необхідної кількості ключів раунду за допомогою алгоритму ключового розширення. Цей процес включає в себе наступні етапи:

- використання константи раунду;
- заміна та поворот слова;
- з’єднання слів.

Константи раунду використовуються для забезпечення нелінійності в процесі розширення ключа. Вони додаються до вихідних даних на певних етапах, що запобігає передбачуваності ключів раунду.

Заміна та поворот слова застосовуються до слів ключа, забезпечуючи нелінійні перетворення та перестановку байтів.

При з’єднання слів, нові слова ключа генеруються шляхом XOR попередніх слів ключа з обробленими словами раунду, забезпечуючи динамічне та непередбачуване розширення ключа.

2.2.2 Асиметричне шифрування з використанням RSA

Алгоритм RSA є одним із найважливіших та найпоширеніших методів асиметричного шифрування в сучасній криптографії. Він базується на глибоких математичних концепціях теорії чисел, зокрема на властивостях простих чисел, модульній арифметиці та функції Ейлера. Далі розглянуті математичні основи RSA, процес генерації ключів, шифрування та розшифрування даних [24].

Основою RSA є складність факторизації великих чисел, зокрема добутків двох великих простих чисел. Ця задача є обчислювально важкою. Алгоритм використовує модульну арифметику, де операції виконуються за модулем певного числа, що дозволяє працювати з великими числами без втрати точності.

Модульна арифметика є фундаментом алгоритму RSA і полягає у виконанні арифметичних операцій над цілими числами за заданим модулем. Вона дозволяє обчислювати остачі від ділення, що є ключовим для створення криптографічних алгоритмів. У модульній арифметиці результат операції завжди знаходиться в межах від 0 до n - 1, де n – модуль.

Функція Ейлера визначає кількість натуральних чисел, менших за n і взаємно простих з ним. Ця функція є критичною для RSA, оскільки використовується при обчисленні закритого ключа:

$$\varphi(m) = (p-1) \times (q-1), \tag{2.5}$$

де p – випадкове велике просте число;

q – випадкове велике просте число.

Щоб глибше зрозуміти, як функція Ейлера інтегрується в RSA, необхідно звернутися до теореми Ейлера.

Теорема Ейлера встановлює фундаментальну властивість модульної арифметики, пов’язану з функцією Ейлера. Вона стверджує, що для будь-якого

цілого числа, яке є взаємно простим, виконується рівність, далі приведено теорему Ейлера:

$$a^{\varphi(m)} \equiv 1 \ (mod\ m), \tag{2.6}$$

де a – ціле число, яке є взаємно простим;

$\varphi(m)$ – функція Ейлера;

$mod\ m$ – операція взяття залишку від ділення.

Ця теорема є похідною від поняття функції Ейлера і безпосередньо використовує її для опису поведінки чисел у модульній арифметиці. У контексті RSA, теорема Ейлера дозволяє гарантувати, що операції шифрування та розшифрування є взаємно оберненими. Функція Ейлера і теорема Ейлера є фундаментальними складовими математичної основи RSA, забезпечуючи його криптографічну стійкість і правильність роботи.

Щоб алгоритм міг ефективно забезпечувати безпечне шифрування та розшифрування даних, необхідно згенерувати пару ключів: відкритий та закритий. Процес генерації цих ключів базується на використанні великих простих чисел та математичних принципів модульної арифметики і функції Ейлера. Далі розглянуто детально кроки, процес генерації ключів у RSA:

Крок 1. Вибір двох великих простих чисел. Ці числа повинні бути випадковими та достатньо великими, зазвичай понад 1024 біти, щоб забезпечити складність факторизації їх добутку.

Крок 2. Обчислення модуля. Модуль є добутком обраних простих чисел.

Крок 3. Розрахунок функції Ейлера.

Крок 4. Вибір відкритого експонента. В більшості випадків встановлюється число 65537, оскільки воно є простим і забезпечує ефективність обчислень.

Крок 5. Знаходження закритого експонента. Закритий експонент є зворотнім до відкритого експонента за модулем обчисленої функції Ейлера.

Крок 6. Формування ключів.

На Рисунку 2.5 представлено схему процесу генерації ключів RSA, що допомагає візуально зрозуміти взаємозв’язки між компонентами алгоритму.

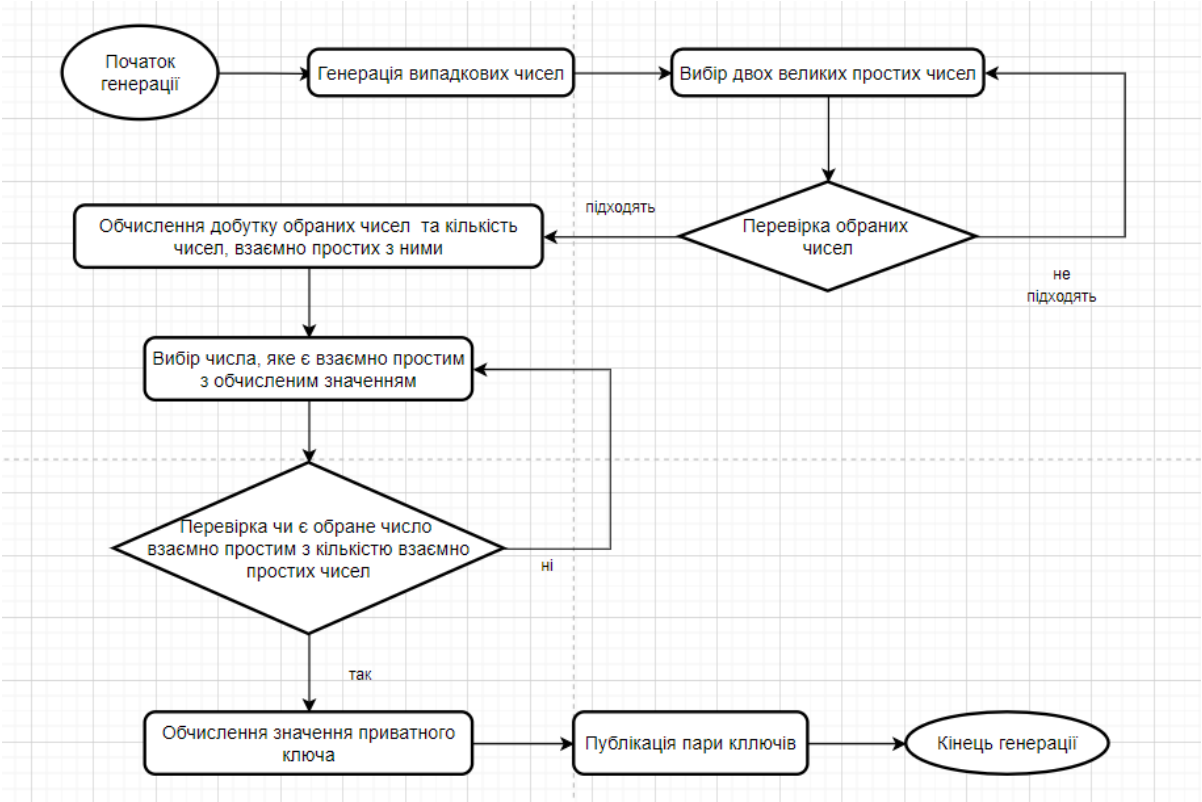

Рисунок 2.5 – Схема генерації RSA ключів

Шифрування повідомлення здійснюється за допомогою публічного ключа, цей процес можна формалізувати наступним чином:

$$C = m^e \ mod\ n, \tag{2.7}$$

де C – зашифроване повідомлення;

m – стартові дані;

e – відкритий експонент, частина відкритого ключа;

n – модуль, який є частиною відкритого ключа.

Цей процес перетворює початкове повідомлення в форму, яку може розшифрувати лише власник закритого ключа.

Розшифрування здійснюється за допомогою приватного ключа (2.8)

$$M = c^d \ mod\ n, \tag{2.8}$$

де M – розшифроване повідомлення;

c – зашифроване повідомлення;

d – закритий експонент, частина закритого ключа;

n – модуль, який є частиною як відкритого, так і закритого ключів.

Завдяки математичним властивостям, встановленим теоремою Ейлера, це рівняння дозволяє точно відновити оригінальне повідомлення.

Розуміння математичних основ RSA та його правильна реалізація є важливими для підтримання високого рівня безпеки в інформаційних системах. Постійний розвиток обчислювальних технологій вимагає уважного ставлення до розмірів ключів та оновлення криптографічних практик для запобігання потенційним загрозам.

2.3 Автентифікація та авторизація з використанням JWT

JWT є стандартом відкритого формату для передачі безпечної інформації між сторонами як компактний, самодостатнім форматом для безпечної передачі інформації між сторонами у вигляді JSON об’єктів. Він дозволяє перевіряти цілісність і автентичність даних за допомогою цифрового підпису або шифрування, що робить його ефективним інструментом для управління доступом у розподілених системах.

JWT складається з трьох основних компонентів: заголовка, навантаження та підпису. Ці компоненти кодуються за допомогою base64 та об’єднуються крапками, утворюючи єдиний токен. Кожен з компонентів виконує специфічну функцію у забезпеченні безпеки та достовірності даних.

Заголовок містить метадані про тип токена та алгоритм підпису, який використовується для його створення. Формалізовано заголовок можна

представити як JSON об’єкт за допомогою якого визначається, який алгоритм буде використовуватися:

$$H = \{ alg, typ \}, \tag{2.9}$$

де H – заголовок;

alg – алгоритм;

typ – тип токену.

Навантаження містить твердження, які несуть інформацію про користувача або інші дані, необхідні для автентифікації та авторизації [25, 26]. Твердження можуть бути зареєстрованими, публічними або приватними. Зазвичай навантаження представляється у наступному вигляді:

$$P = \{ \text{sub},\ \text{name},\ \text{iat}, \text{exp}\}, \tag{2.10}$$

де sub – ідентифікатор користувача;

name – ім’я користувача;

iat – час початку дії;

exp – час закінчення дії.

Підпис використовується для перевірки цілісності токена та підтвердження його автентичності. Підпис формується шляхом шифрування поєднуючі складові частини. Підпис виглядає наступним чином:

$$S = ALG(base64(H)\ .\ base64(P)\ , sk), \tag{2.11}$$

де S – підпис;

ALG – алгоритм підпису;

H – заголовок;

P – навантаження;

sk – секретний ключ.

Найпоширенішим серед способів підпису є HMAC з використанням алгоритму SHA256. HMAC забезпечує цілісність та автентичність повідомлень за допомогою секретного ключа та хеш-функції. Математично HMAC визначається наступним чином:

$$HMAC(K, m) = H(\ (K' \oplus opad) \vee H(\ (K' \oplus ipad) \vee\ m)\ ), \quad (2.12)$$

де K – приватний ключ;

m – повідомлення;

H – хеш-функція;

K' - ключ, доповнений або обрізаний до блочного розміру хеш-функції;

$opad$ – константа заповнювач;

$ipad$ – константа заповнювач.

Процес генерації підпису включає піднесення заголовка та навантаження до ступіня, визначеного відкритим експонентом, та застосування HMAC-SHA256 з використанням секретного ключа. Підпис забезпечує захист від підробки токенів, оскільки будь-яка зміна в заголовку або навантаженні призведе до невідповідності підпису при верифікації. Це досягається завдяки використанню секретного ключа, який відомий лише довіреним сторонам.

Автентичність токена визначається тим, що виключно власник секретного ключа здатен сформувати коректний підпис для JWT. Таким чином, отримувач токена може бути впевнений, що він надійшов від авторизованого користувача і не був змінений зловмисниками.

Такий підхід робить створення підроблених токенів без доступу до секретного ключа неможливим. Завдяки цьому взаємодія між відправником та отримувачем набуває більшої надійності, а ризик несанкціонованого використання інформації помітно зменшується. Завдяки впроваджений підходам система автентифікації демонструє високий рівень надійності та відповідає встановленим вимогам безпеки.

Захист від підробки токенів досягається через наступні механізми:

– секретний ключ використовується для генерації та верифікації підпису, забезпечуючи, що лише довірені сторони можуть створювати валідні токени;

– алгоритм підпису гарантує складність відновлення секретного ключа або створення валідного підпису без знання цього ключа;

– цілісність даних підтверджується перевіркою підпису, що забезпечує, що дані не були змінені після створення токена.

Таким чином, JWT забезпечує надійний механізм для автентифікації та авторизації, дозволяючи ефективно контролювати доступ до ресурсів у розподілених системах без необхідності зберігання стану сесій на сервері.

2.4 Механізм захисту за допомогою проксі-сервера

Одним із ефективних засобів підвищення рівня захисту є використання проксі-серверів. Проксі-сервери виконують функцію посередника між клієнтами та цільовими серверами, надаючи додатковий рівень абстракції та контролю над передачею даних. Це дозволяє приховати реальну адресу серверів, забезпечуючи анонімність та захист від прямих атак.

Проксі-сервери інтегруються в архітектуру інформаційних систем як ключові компоненти, що управляють потоком інформації між користувачами та ресурсами мережі. Вони здійснюють маршрутизацію запитів, забезпечують кешування даних, фільтрацію контенту та реалізують політики доступу. Важливим аспектом є можливість контролювати та аналізувати трафік, що проходить через проксі, що сприяє виявленню потенційних загроз та забезпеченню відповідності вимогам безпеки.

Однією з основних функцій проксі-сервера є приховування реальної IP-адреси цільових серверів. Це досягається шляхом заміщення власної IP-адреси проксі-сервера адресою клієнта під час пересилання запиту. Таким чином,

зовнішні спостерігачі не можуть безпосередньо визначити місцезнаходження та ідентичність реальних серверів, що значно ускладнює здійснення атак типу DDoS, проникнення через вразливості сервера та інші види кіберзагроз.

Математична модель маршрутизації через проксі-сервер може бути представлена наступним чином:

$$R(C \to P \to S) = (C \to P) \cup (P \to S), \tag{2.13}$$

де R – маршрут між клієнтом та сервером;

C – клієнт;

P – проксі-сервер;

S – цільовий сервер.

Запропонована конфігурація та оточення дозволяє гнучко реагувати на зміни умов експлуатації та зростання потреб у продуктивності. Розробники можуть швидко додавати нові вузли, удосконалювати політики доступу чи коригувати конфігурацію фільтрів без зупинки системи. Такий динамічний підхід відповідає внутрішнім нормативам, а також загальноприйнятим стандартам безпеки, забезпечуючи як стійкість до потенційних збоїв, так і готовність до впровадження нових технологій.

Ця модель демонструє, що проксі-сервер є критичним вузлом, через який проходить весь трафік між клієнтом та сервером. Також він може використовувати механізми шифрування трафіку, що гарантує захист інформації від перехоплення та змін у процесі передачі. Проксі-сервери можуть здійснювати автентифікацію користувачів, перевіряючи їхні права доступу та забезпечуючи відповідність політикам безпеки організації.

3 ДОСЛІДЖЕННЯ ТА ТЕСТУВАННЯ КОМПОНЕНТНИХ МОДЕЛЕЙ

3.1 Обґрунтування вибору середовища програмної реалізації

3.1.1 Обґрунтування вибору Laravel як серверного середовища

Laravel є потужний та гнучкий інструмент для створення вебзастосунків, який підтримує швидку розробку проєктів різного масштабу. Однією з ключових особливостей Laravel є його зручні засоби для управління маршрутизацією, що дозволяють легко налаштовувати REST API та інші види інтерфейсів для взаємодії з клієнтами. Використання спеціалізованого файлу «routes/api.php» дозволяє розділяти маршрути на різні групи, враховуючи різні рівні доступу та можливості системи. Це особливо важливо для підтримки масштабованості проєктів, де кількість маршрутів може значно зрости. Такий підхід допомагає забезпечити чітку структуру та підтримуваність коду, що є особливо важливим при створенні проєктів з великою кількістю функціональностей.

Він пропонує інтеграцію з багатьма популярними сервісами, що дозволяє розширити функціональність системи без необхідності розробляти власні рішення. Це забезпечує розробникам можливість швидко додавати нові сервіси до проєкту, використовуючи вже існуючі бібліотеки та компоненти, що пройшли перевірку на безпеку [27]. Фреймворк підтримує роботу з базами даних за допомогою Eloquent ORM, що дозволяє зручно маніпулювати даними у різних базах за допомогою об’єктно-орієнтованого підходу.

Ще однією важливою перевагою є система міграцій, яка дозволяє легко керувати структурою бази даних протягом усього життєвого циклу проєкту. Міграції дозволяють створювати, змінювати та видаляти таблиці у базі даних без необхідності вручну вводити SQL-запити, що підвищує безпеку та забезпечує контроль версій бази даних. Цей підхід дозволяє зберігати всю історію змін структури бази даних у вигляді міграційних файлів.

Laravel має потужні засоби для розробки фронтенду, підтримуючи інтеграцію з сучасними JavaScript фреймворками. Це дозволяє легко створювати динамічні інтерфейси користувача, які відповідають сучасним стандартам веброзробки. Використання вбудованого шаблонного механізму, спрощує розробку інтерфейсу, забезпечуючи зручний синтаксис та можливість інтеграції зі стилями та скриптами.

Активна підтримка спільнотою та регулярні оновлення дозволяють розробникам завжди користуватися найновішими засобами безпеки та оптимізації. Регулярні оновлення фреймворку забезпечують захист від останніх загроз, таких як SQL ін’єкції або XSS-атаки, завдяки вбудованим механізмам захисту та верифікації даних.

Laravel є одним з небагатьох фреймворків, який пропонує простий та інтуїтивний доступ до інструментів для реалізації процесів шифрування, що значно полегшує роботу з конфіденційними даними. Фреймворк підтримує використання стандартних алгоритмів шифрування, що гарантує високу безпеку та сумісність з більшістю сучасних систем [28]. Вбудовані «encryptors» та «decryptors» забезпечують можливість зберігати зашифровану інформацію в базах даних або передавати її через API без необхідності додаткової конфігурації.

Також він дозволяє розробникам використовувати свій власний набір криптографічних ключів, що дає можливість точно налаштовувати рівень безпеки та відповідати конкретним вимогам. Інтеграція з OpenSSL дозволяє легко генерувати ключі, керувати сертифікатами та налаштовувати протоколи безпеки, що значно полегшує процеси, пов’язані з шифруванням. Зручний API для шифрування та розшифрування даних дозволяє швидко налаштовувати захист у проєкті, навіть якщо розробник не має глибоких знань у криптографії. Завдяки широкому набору функцій та доступності документації, процеси шифрування стають доступними, що робить цей фреймворк одним з найзручніших для роботи з безпекою.

3.1.2 Обґрунтування вибору Vue

Vue.js є одним із провідних інструментів для створення сучасних інтерфейсів користувача, що відзначається своєю гнучкістю, легкістю інтеграції та потужною функціональністю. У рамках даного проєкту використання Vue у зв’язці з іншими технологіями, такими як axios, Vuetify, Pinia Store та Vite, є оптимальним вибором для забезпечення швидкої та безпечної розробки клієнтської частини вебзастосунку [29]. Ця комбінація технологій дозволяє досягти високої продуктивності, зручності використання та легкості налаштування, що робить її незамінною.

Однією з основних переваг є його модульність та компонентний підхід до розробки інтерфейсів користувача. Це дозволяє створювати компоненти, які легко підтримуються, повторно використовуються та оновлюються. Компонентна архітектура сприяє підвищенню безпеки, оскільки ізольовані компоненти мінімізують вплив помилок чи уразливостей на всю систему [30]. Крім того, фреймворк підтримує сучасні стандарти безпеки, включаючи захист від поширених вразливостей через використання шаблонів з автоматичним екрануванням даних.

Важливим аспектом у використанні Vue є інтеграція з axios – бібліотекою для здійснення HTTP-запитів. Axios дозволяє легко обробляти запити до сервера та управляти відповідями, забезпечуючи надійний механізм передачі даних. У контексті проєкту, він використовується для інтеграції з API, що забезпечує безпечну взаємодію клієнтської частини з сервером [31]. Axios також має вбудовані засоби обробки помилок та повторного надсилання запитів, що підвищує стійкість системи до можливих збоїв у мережі.

Ще однією суттєвою перевагою Vue є можливість легкого інтегрування Vuetify – популярної бібліотеки, яка базується на концепції Material Design. Використання Vuetify забезпечує створення естетично привабливих та функціональних інтерфейсів користувача, що відповідають сучасним стандартам дизайну. Бібліотека надає широкий вибір компонентів, таких як

форми, таблиці, діаграми та інші елементи інтерфейсу, що дозволяє швидко створювати зручні та інтуїтивно зрозумілі інтерфейси користувача. Важливо зазначити, що Vuetify також включає інструменти для перевірки введених даних, що дозволяє покращити безпеку форми, автоматизуючи процес валідації [32].

Використання Pinia Store як основного засобу управління станом застосунку є ще одним значним аспектом у виборі Vue. Pinia Store пропонує простий, але потужний підхід до централізованого управління даними, дозволяючи зберігати глобальний стан програми в одному місці. Це підвищує безпеку та зручність розробки, оскільки всі дані зберігаються в одному місці, що дозволяє легко керувати доступом до них, впроваджуючи контроль доступу на рівні стану. Використання бібліотеки знижує ризик помилок, пов’язаних із синхронізацією даних між різними компонентами застосунку, завдяки реактивності Vue, яка автоматично оновлює компоненти при зміні стану.

Не менш важливою є роль Vite у побудові сучасного середовища. Це потужний інструмент для налаштування та керування процесом збирання проєкту. Завдяки своїй високій продуктивності та мінімальним витратам ресурсів, Vite забезпечує швидкий процес збірки та миттєве оновлення змін у коді під час розробки. Це особливо важливо для великих проєктів, де швидкість оновлення інтерфейсу має велике значення.

Ще однією перевагою є інтеграція з системами шифрування та безпеки. Завдяки Vue.js можна легко налаштувати підтримку JWT для автентифікації та авторизації користувачів, що дозволяє забезпечити захист даних при взаємодії з сервером. Використовуючи механізми шифрування на стороні сервера, Vue дозволяє створити безпечний клієнт-серверний зв’язок, де всі дані, що передаються, шифруються та захищаються від несанкціонованого доступу.

Завдяки своїй простоті та широкій підтримці спільноти розробників, Vue надає великий набір готових рішень та плагінів, що полегшують розробку

програм. Це дозволяє значно скоротити час розробки, підвищуючи ефективність праці розробників та зменшуючи ризики, пов’язані з використанням нестабільних чи застарілих бібліотек.

Вибір Vue разом із супутніми технологіями забезпечує не тільки високу продуктивність розробки, але й безпеку, зручність використання та можливість швидкої адаптації до змін у вимогах проєкту. Завдяки своїм можливостям він дозволяє створювати стабільні та захищені інтерфейси, що відповідають сучасним стандартам програмування та безпеки.

3.1.3 Обґрунтування вибору Electron

Electron є потужним середовищем для створення кросплатформових десктопних проєктів, що дозволяє поєднати вебтехнології з можливостями настільного програмного забезпечення. Використовуючи JavaScript, HTML і CSS, розробники можуть створювати застосунки, які працюють на різних операційних системах, зберігаючи єдину кодову базу. Основною перевагою Electron є можливість інтеграції з Node.js, що забезпечує доступ до системних ресурсів та функцій, недоступних для звичайних вебінтерфейсів [33]. У рамках даного проєкту фреймворк був обраний як основне середовище для розробки десктопного проєкту, через його здатність легко інтегруватися з Vue, забезпечуючи зручний користувацький інтерфейс та сучасні можливості фронтенду. Фреймворк дозволяє використовувати той самий Vue.js код, що й у вебверсії, створюючи єдиний користувацький досвід на різних платформах. Це значно спрощує підтримку та оновлення застосунку, оскільки основна функціональність залишається спільною як для веб, так і для десктопних версій.

Electron дозволяє вбудовувати вебтехнології в десктоп ситеми, що робить його ідеальним для інтеграції з Vue, який використовується для створення фронтенду. Контент програми, що відображається на екрані,

базується на Vue, що надає можливість швидкого рендерингу та зручної організації інтерфейсу. Завдяки використанню Vite як збирача проєкту, досягається оптимізація продуктивності та мінімізація часу на компіляцію, що є важливим аспектом для швидкості роботи десктоп застосунка.

На рисунку 3.1 приведено структуру системи, де показано, як виконується взаємодія Electron з клієнтською частиною при роботі як десктоп застосунка.

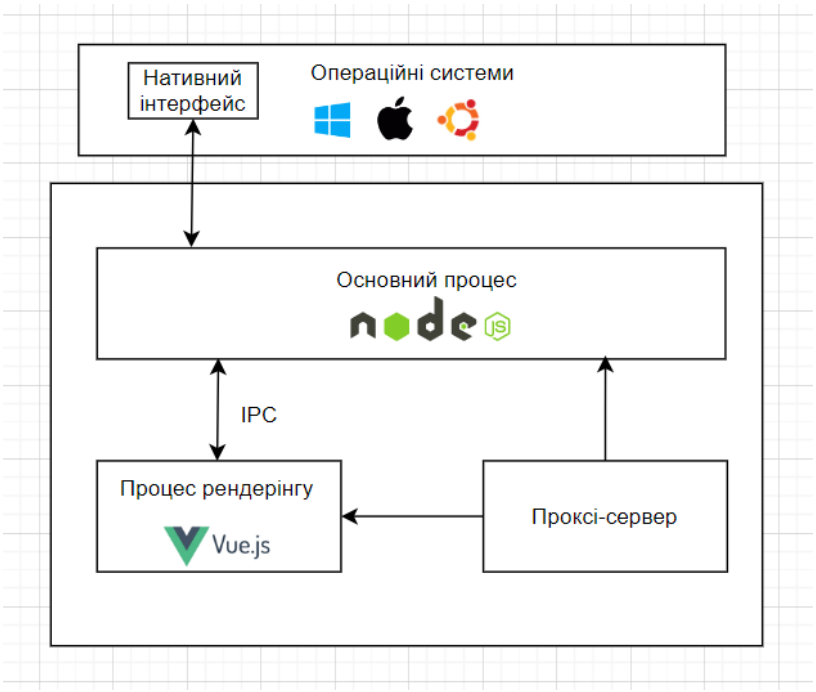

Рисунок 3.1 – Взаємодія компонентів при роботі як десктоп застосунку

Фреймворк також забезпечує можливість використання Node.js для виконання серверної логіки в межах самого десктопного середовища [34]. Це дозволяє виконувати складні обчислення, обробку даних, а також реалізовувати маршрутизацію запитів за допомогою Express, що є важливим для структурованої організації API всередині десктоп системи. В рамках цього проєкту Node.js використовується для проксі-сервера, який перенаправляє всі запити з клієнта на сервер, забезпечуючи додатковий рівень безпеки.

Однією з переваг використання Electron є підтримка кросплатформових збірок. За допомогою Electron Builder створюється універсальний десктоп

застосунок, який може працювати на Windows, MacOS та Ubuntu без необхідності створення окремих версій для кожної платформи. Ця особливість значно скорочує час на розробку та тестування, дозволяючи зосередитись на функціональних аспектах проєкту, а не на питаннях сумісності. Electron Builder також дозволяє створювати інсталяційні файли та пакети, що спрощує розгортання та оновлення застосунка на різних платформах. Це забезпечує зручність для кінцевого користувача, оскільки весь процес установки та оновлення стає максимально автоматизованим.

Безпека є ключовим аспектом при розробці будь-якого застосунка, особливо коли йдеться про обробку конфіденційних даних. Electron надає доступ до системних функцій, що дозволяє реалізовувати складні механізми шифрування даних безпосередньо на стороні клієнта. У рамках цього проєкту використовуються інструменти для шифрування та розшифрування даних на стороні Electron. Процеси шифрування виконуються безпосередньо перед відправленням даних на сервер, що зменшує ризик перехоплення конфіденційної інформації.

Фреймворк має зручну архітектуру, що дозволяє легко налаштовувати програму та додавати нові функції. Використання JavaScript та Node.js у якості основної мови програмування значно спрощує реалізацію нових функцій, таких як додаткові інтерфейси або розширення можливостей шифрування. Vue у свою чергу, дозволяє легко інтегрувати нові компоненти інтерфейсу, роблячи систему гнучкою для змін та адаптації під потреби конкретного проєкту [35]. Ця гнучкість дозволяє швидко реагувати на зміни вимог до безпеки або додавати нові методи шифрування, не змінюючи основної структури проєкту. Доступність великої кількості бібліотек для Node.js дозволяє розширювати можливості застосунка без значних зусиль.

3.2 Програмна реалізація безпечних компонентів

3.2.1 Створення та налаштування серверної частини

Laravel використовує концепцію middleware для забезпечення безпеки доступу до ресурсів. Middleware є проміжним шаром між HTTP-запитом і контролером, що дозволяє перевірити права доступу або автентичність користувача перед тим як передавати запит на обробку.

Представлено AdminMiddleware, який контролює доступ до ресурсів тільки для користувачів тільки якщо вони мають роль адміністратора (рис. А.1). Якщо користувач не має необхідних прав, система повертає відповідь із кодом 403, запобігаючи несанкціонованому доступу. Цей механізм дозволяє легко налаштовувати ролеву модель контролю доступу на основі різних рівнів авторизації.

Authenticate, який перевіряє автентичність користувача (рис. А.2). У випадку, коли користувач не авторизований, його перенаправляють на сторінку входу або, якщо очікується JSON відповідь, повертають JSON повідомлення. Це дозволяє більш гнучко управляти запитами в системах, що використовують REST API.

Middleware у Laravel забезпечує централізоване управління доступом до ресурсів та дозволяє легко адаптувати систему до змін у політиці безпеки, знижуючи ризик помилок у конфігурації прав доступу.

Для збереження та управління даними було обрано реляційну базу даних, яка забезпечує ефективну організацію даних за допомогою чіткої структурованої схеми.

Цей підхід дозволяє зберігати дані, обробляти запити та маніпулювати даними відповідно до потреб бізнес процесу, гарантуючи цілісність та безпеку даних. Основою для роботи з базою даних у системі є Eloquent ORM, яка входить до фреймворку Laravel, що надає зручний об’єктно-реляційний інтерфейс при цьому дотримуючись стандартів при побудові реляційних баз даних.

На рисунку 3.2 показана структура бази даних, яка використовується у проєкті. Вона складається з таких основних таблиць, як users, team, competitions, participants, competition_type, heat та distance, що забезпечують логічну організацію даних та взаємозв’язок між ними. Схема бази даних створена з урахуванням вимог щодо безпеки та можливостей для майбутнього розширення.

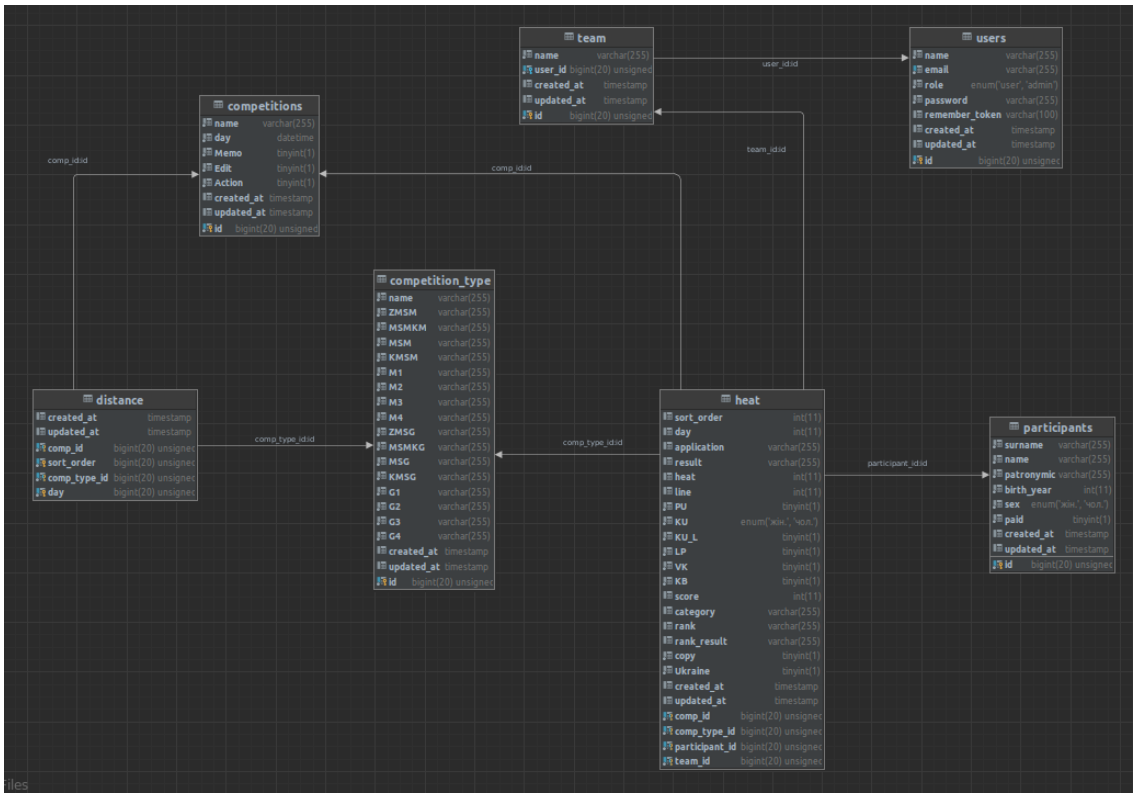

Рисунок 3.2 – Схема структури бази даних

Таблиця users та модель User містить інформацію про користувачів системи, включаючи їхні електронні адреси, ролі паролі та токени для автентифікації (рис. А.3). Вона є основою для забезпечення безпеки доступу до даних, дозволяючи розділяти права доступу та керувати доступом до різних функцій системи.

Team зберігає інформацію про команди та має зв’язок з таблицею users через зовнішній ключ user_id (рис. А.4). Це дозволяє організовувати командну структуру у рамках проєкту, прив’язуючи конкретних користувачів до певних команд.

Змагання зберігаються у таблиці competitions та керуються за допомогою моделі Competition, містить інформацію про різні змагання, які організовуються у межах проєкту (рис. А.5). Таблиця пов’язана з іншими таблицями, такими як distance, яка зберігає інформацію про дистанції, що використовуються у кожному змаганні.

Таблиця distance та модель Distance, описує дистанції для кожного змагання (рис. А.6). Вона має зв’язок із таблицею competitions через ключ comp_id, а також з таблицею competition_type через ключ comp_type_id. Це дозволяє гнучко налаштовувати дистанції в межах кожного заходу.

Сутність participants зберігає персональні дані учасників змагань, включаючи їхні ім’я, прізвище, рік народження та інші характеристики (рис. А.7). Вона інтегрується з іншими таблицями для створення комплексної інформаційної моделі змагань.

CompetitionType містить інформацію про різні категорії та типи змагань, що допомагає структуризувати дані змагань і забезпечити їх чітке розмежування (рис. А.8). Це робить систему більш гнучкою та дозволяє легко додавати нові категорії змагань у майбутньому.

Таблиця heat та модель Heat є однією з центральних таблиць бази даних, яка описує сесії змагань (рис. А.9). Вона має складний первинний ключ, що складається з кількох полів «comp_id, comp_type_id, participant_id, team_id», забезпечуючи унікальну ідентифікацію кожної сесії змагань. Таблиця має зв’язки з іншими моделями, такими як Competition, CompetitionType, Participant, та Team, що дозволяє ефективно організовувати змагання та аналізувати результати.

Архітектура бази даних не лише забезпечує цілісність та безпеку даних, а й дозволяє виконувати складні запити для отримання необхідної інформації. Це особливо важливо у системі, де безпека даних є пріоритетом, оскільки використання зовнішніх ключів дозволяє ефективно керувати взаємозв’язками між таблицями. Завдяки цій структурі система має високий рівень надійності та безпеки, забезпечуючи легкий доступ до даних та їх управління.

Завдяки використанню Eloquent ORM, кожна таблиця має відповідну модель, яка дозволяє зручно працювати з даними, здійснювати їх маніпуляції та підтримувати безпеку на високому рівні. Це робить систему масштабованою та гнучкою, що дозволяє легко додавати нові функції та розширювати функціонал без значних змін в архітектурі бази даних.

Контролери в Laravel відіграють ключову роль у взаємодії між клієнтською частиною та базою даних, організовуючи обробку HTTP-запитів, виконання бізнес-логіки та валідацію даних. Для кожної моделі було створено окремий контролер, що дозволяє забезпечити структуру програми, побудовану за принципом MVC.

Контролери для цих моделей виконують стандартні CRUD: створення, читання, оновлення, видалення операції, що забезпечують доступ та маніпуляції з відповідними даними. Наприклад, контролер UserController керує операціями над користувачами, включаючи фільтрацію за ролями, створення нових записів користувачів, оновлення даних та їх видалення. Аналогічний підхід використовувався для контролерів TeamController, CompetitionController, CompetitionTypeController, DistanceController, HeatController, та ParticipantController. Наведено приклад з HTTP контролера (рис А.10).

Особливе місце у системі займає основний контролер AuthController, який відповідає за автентифікацію та авторизацію користувачів (рис. А.11, рис. А.12). У ньому реалізовано механізми створення користувачів, із використанням вбудованих засобів Laravel для валідації вхідних даних. Важливим аспектом є шифрування паролів за допомогою сервісу «Hash::make», що забезпечує безпеку зберігання чутливої інформації в базі даних, зберігаючи тільки хеш паролів Під час автентифікації AuthController генерує токен доступу за допомогою функції createToken, який забезпечує безпечний доступ до ресурсів системи. Це дозволяє ефективно керувати сесіями користувачів та підтримувати високий рівень безпеки під час взаємодії з сервером.

Маршрути забезпечують доступ до серверної частини за допомогою HTTP-запитів, визначаючи, які URL адреси відповідають за конкретні дії. У даній системі всі API маршрути розташовані у файлі api.php і організовані для роботи з різними сутностями, такими як користувачі, учасники, команди, змагання тощо (рис. А.13, рис. А.14). Усі маршрути використовують методи, що відповідають стандартним CRUD операціям: створення, читання, оновлення та видалення даних.

Для підвищення безпеки в API маршрутах активно використовуються middleware, такі як «auth:sanctum», «decrypt.request» та «encrypt.response», що забезпечують автентифікацію користувачів, дозволяють доступ лише авторизованим користувачам, шифрують та розшифровують дані. Це допомагає захистити дані та обмежити доступ до чутливої інформації.

3.2.2 Реалізація клієнтської частини

Структура клієнтської частини, створена на основі Vue.js, основна увага приділяється побудові архітектури, налаштуванню інструментів для управління станом, дизайну інтерфейсу та захисту даних. Клієнтська частина використовує сучасні підходи до організації коду, що дозволяє досягти високої зручності використання, ефективності та безпеки при взаємодії з користувачами.

Центральною точкою ініціалізації інтерфейсу є вхідний файл проєкту «main.js» (рис. А.15, рис. А.16). У ньому відбувається підключення основних бібліотек, таких як Vue, що відповідає за реактивне відображення інтерфейсу, обробку даних з серверу, зміну клієнтського інтерфейсу, та Vuetify, що надає широкий спектр компонентів для побудови сучасного дизайну користувацького інтерфейсу. З використанням Vuetify створюється світла тема з налаштуванням основного кольору, що задає стильову основу для всього проєкту [36].

Для керування станом застосунку використовується Pinia, де завдяки підключеному плагіну «piniaPluginPersistedstate», стан системи зберігається між сесіями. Це дозволяє зберігати інформацію про користувачів та налаштування інтерфейсу навіть після закриття браузера.

Однією з ключових функцій файлу є налаштування маршрутизації за допомогою об’єкта router, що забезпечує навігацію між сторінками застосунку. Маршрутизація включає перевірку аутентифікації користувача, де перед кожним переходом виконується валідація доступу до захищених ресурсів. У випадку, якщо доступ вимагає авторизації, але користувач не автентифікований, він буде автоматично переадресований на сторінку входу. Це забезпечує захист даних та обмежує доступ до внутрішніх частин системи лише для авторизованих користувачів.

Завершальний етап ініціалізації включає монтування інтерфейсу на DOM елемент за допомогою «app.mount("#app")», що дозволяє Vue управляти відображенням інтерфейсу користувача.

Для реалізації запитів до API в застосунку використовуються константи, що забезпечують зручність і повторне використання коду. Константи містять методи, що звертаються до API за допомогою бібліотеки axios, забезпечуючи стандартизовану взаємодію з сервером. Це є важливим аспектом компонентної архітектури, оскільки дозволяє легше керувати логікою запитів та зберігати її в одному місці.

Кожна з API функцій представлена окремими константами, які здійснюють операції наступні операції: отримання, додавання, оновлення та видалення даних для всіх сутностей проєкту (рис. А.17). Такий підхід дозволяє забезпечити інкапсуляцію логіки запитів і легкий доступ до даних з різних частин застосунку.

Завдяки цьому, у компонентній моделі Vue код стає більш організованим і зрозумілим, спрощується тестування та підтримка, а також зменшується ймовірність помилок, оскільки всі звернення до API централізовані та стандартизовані.

Маршрутизація є важливою частиною, яка визначає навігацію між різними сторінками та компонентами. Файл маршрутизації (рис. А.18, рис. А.19) налаштовує маршрути програми, використовуючи «vue-router», що дозволяє створювати багатосторінкові SPA. У файлі визначені шляхи для кожного компоненту, включаючи такі, як сторінка входу «/login» або головна сторінка змагань «/home».

Кожен маршрут має атрибут «meta», що визначає, чи потрібна автентифікація для доступу до сторінки. Це дозволяє контролювати доступ користувачів до різних частин застосунку, наприклад, сторінки учасників змагань «/participants» або списку команд «/coaches» доступні лише для авторизованих користувачів. Такий підхід допомагає захистити конфіденційні дані та обмежити доступ до важливих ресурсів.

Використання мета атрибутів також дозволяє легко налаштовувати правила навігації без необхідності складних перевірок всередині кожного компонента. Це робить код більш організованим і забезпечує зручне адміністрування правил доступу до системи.

Використання сховищ даних реалізовано за допомогою Pinia. Сховища дозволяють централізовано керувати станом застосунку, забезпечуючи зручний доступ до даних з будь-якої частини компонента, а також їхнє збереження та оновлення. Використання такого підходу спрощує маніпуляції з даними та забезпечує кращу організацію компонентної архітектури.

Основним прикладом є «useAuthStore» (рис. А.20, рис. А.21), який відповідає за управління аутентифікацією та авторизацією користувачів у системі. Це сховище включає функціонал для реєстрації, входу, виходу, отримання списку тренерів та окремих користувачів, а також оновлення даних про користувачів. Завдяки централізації аутентифікаційних даних через це сховище, компоненти можуть легко перевіряти статус аутентифікації та отримувати необхідні дані про користувача без повторення коду.

Ще одним прикладом є «useHeatStore», який обробляє дані про сесії змагань heats. Це сховище дозволяє отримувати список сесій, додавати нові,

оновлювати існуючі, а також видаляти дані про сесії. Такий підхід забезпечує гнучке керування даними змагань, полегшуючи їх відображення та обробку у відповідних компонентах.

Ключовим моментом використання Pinia є можливість зберігання стану між оновленнями сторінок, що реалізується за допомогою плагіну для постійного збереження даних. Це гарантує, що дані користувача або сесій залишаться доступними навіть після перезавантаження сторінки.

3.2.3 Реалізація десктопної частини

Electron забезпечує платформу для розробки десктоп застосунку на основі вебтехнологій, таких як HTML, CSS, та JavaScript, використовуючи Node.js для інтеграції з серверними процесами. Розглянута основна структура застосунку, що включає ініціалізацію головного вікна програми та налаштування взаємодії через ProxyServer для безпечної маршрутизації запитів користувача та відповідей сервера (рис. А.22).

Реалізована основна логіка запуску програми за допомогою фреймворку Electron. Стартовий файл ініціалізує застосунок, створюючи нове вікно «BrowserWindow», яке виступає інтерфейсом основним користувача. Параметри вікна забезпечують безпечне виконання коду, дозволяючи інтегрувати Node.js «nodeIntegration: true» та вимикаючи ізоляцію контексту «contextIsolation: false». Це дозволяє програмі отримувати доступ до ресурсів Node.js, що є необхідним для десктопних рішень з широкими функціональними можливостями.

Крім створення вікна, тут також ініціалізується серверна частина на базі бібліотеки Express. Сервер запускається на порту 5180, забезпечуючи маршрутизацію для компонентів, що використовують цей порт. Шлях до ресурсів інтерфейсу визначено через «express.static», де вказується статичний шлях до збірки Vue.js інтерфейсу.

Electron Builder використовується для створення інсталяційних файлів проєкту на основі Electron, що забезпечує зручну конфігурацію та підтримку збірки під різні платформи. У файлі конфігурації (рис. А.23) налаштовується «appId» проєкту, шляхи до вихідної директорії та файлів, які включаються до збірки під платформи. Процес зборки оптимізовано для трьох основних платформ: Windows, MacOS і Linux.

Для MacOS налаштований формат «zip», а інсталяційний файл називається відповідно до версії застосунка. У конфігурації для Windows обраний формат «portable» з архітектурою «x64», що дозволяє отримати універсальний інсталяційний файл для операційної системи Windows. Для Linux використовуються «deb» пакети, що забезпечують сумісність із багатьма дистрибутивами.

Сам builder спрощує процес створення багатоплатформного дистрибутива програми, дозволяючи налаштувати параметри встановлення та видалення, а також зберігаючи єдину структуру для збирання.

3.2.4 Реалізація шифрування даних з використанням AES та RSA

Застосування шифрування для захисту даних під час передачі є одним із ключових компонентів у системі безпеки. Реалізовано гібридне шифрування: AES використовується для шифрування безпосередніх даних, а RSA – для захисту симетричного ключа (рис. А.24, рис. А.25). Такий підхід забезпечує баланс між ефективністю обробки даних і стійкістю до атак.

Клас «ProxyServer» відповідає за обробку зашифрованих запитів і відповідей між десктопним клієнтом і сервером. Це забезпечує конфіденційність і цілісність даних під час їх передачі через використання комбінації AES та RSA. Під час ініціалізації «ProxyServer» створюється випадковий симетричний ключ AES, який шифрується публічним ключем RSA сервера. Це гарантує, що тільки сервер з відповідним приватним ключем

зможе розшифрувати симетричний ключ і використовувати його для обробки запитів і відповідей.

Метод «encryptRequest» призначений для шифрування тіла запиту перед відправкою на сервер. Спочатку дані запиту перетворюються у формат JSON, після чого генерується вектор ініціалізації – випадкова послідовність байтів, необхідна для режиму шифрування «aes-256-cbc», що додає додатковий рівень захисту від атак на симетричні алгоритми. З використанням AES ключа і згенерованого IV створюється шифр, який застосовується для шифрування JSON даних запиту. Закінчений результат шифрування представлений у форматі Base64 для уникнення проблем із кодуванням під час передачі. У заголовки запиту додаються зашифрований симетричний ключ AES та IV, що дозволяє серверу розшифрувати дані. Такий підхід до шифрування підвищує захищеність від потенційних атак на дані завдяки використанню AES ключа, захищеного RSA.

Для обробки відповідей від сервера «ProxyServer» використовує метод «decryptResponse». На початкових етапах здійснюється перевірка статусу відповіді – метод обробляє тільки успішні відповіді. Далі отримується IV з заголовків відповіді, що є обов’язковою умовою для успішної розшифровки. Зашифровані дані відповіді декодуються з Base64, що повертає оригінальні зашифровані дані. За допомогою AES дешифратора з симетричним ключем та отриманим IV дані повертаються у початковий вигляд. Використання IV у кожному запиті забезпечує неповторюваність шифрування навіть при однакових запитах, що підвищує захист від атак, основаних на аналізі шаблонів у зашифрованих даних.

Конфігурація проксі-сервера налаштована в методі proxyConfig, де задаються параметри обробки запитів і відповідей. Використовуючи бібліотеку «http-proxy-middleware», конфігурація дозволяє перехоплювати запити, застосовуючи метод «encryptRequest» перед відправленням, а «decryptResponse» – для обробки отриманих відповідей. Це забезпечує повний контроль над передачею даних, що зберігає їх конфіденційність і захищає від

витоків інформації. Реалізація шифрування у проксі-сервері надає високий рівень безпеки для даних від початку і до кінця, розглядаючі частину клієнта як процес передачі даних від генерації симетричного ключа до самої обробки.

На стороні сервера в Laravel реалізовані механізми шифрування та розшифрування, що працюють через окремі проміжні middlewares. Цей підхід забезпечує багаторівневу безпеку та дозволяє безпечно обробляти дані, захищаючи їх на всіх етапах передачі.

Спочатку процес розшифрування запитів у «DecryptRequestMiddleware» (рис. А.26, рис. А.27). Коли сервер отримує зашифрований запит, він перевіряє наявність заголовка «X-Symmetric-Key», який містить симетричний ключ, зашифрований публічним ключем сервера. Після виявлення цього заголовка сервер завантажує приватний ключ з локальної файлової системи та використовує його для розшифрування симетричного ключа за допомогою бібліотеки «phpseclib3». При цьому важливим є застосування стандартів шифрування RSA з OAEP та хешуванням SHA-256 для забезпечення криптостійкості проєкту.

Далі за допомогою симетричного ключа та значення IV змінної, що передається в заголовку як «X-IV» параметер. Сервер розшифровує цей параметер за допомогою ключа, потім аналогічно основні дані запиту. Зашифрований контент витягується з поля data у запиті, а далі обробляється за допомогою алгоритму AES-256 у режимі CBC. Після успішного розшифрування вміст замінюється на декодовану інформацію, яку можна зберігати чи обробляти як стандартний вхідний запит.

Після обробки запиту сервер відповідає клієнту, використовуючи «EncryptResponseMiddleware» (рис. А.28). У цьому middleware перед передачею даних назад клієнту вони шифруються тим же симетричним ключем та новим ініціалізаційним вектором, що генерується на кожен запит. Після шифрування контенту метод «openssl_encrypt» кодує вміст, а потім результат у форматі Base64 відправляється клієнту разом із заголовком «X-IV», що містить новий IV для розшифрування на стороні клієнта.

Така комплексна система шифрування створює надійний захист даних на всіх етапах їх передачі та обробки. На стороні клієнта реалізоване динамічне шифрування через проксі-сервер, де використовується симетричний ключ AES для кожного сеансу, що забезпечує високу швидкість шифрування та надійність даних. Серверний компонент в свою чергу, інтегрує механізми розшифрування та подальшого шифрування відповіді, використовуючи асиметричне RSA шифрування для захисту симетричного ключа при його передачі.

Цей підхід поєднує швидкість та захищеність, характерні для AES, з надійною криптостійкістю RSA, що робить його оптимальним рішенням для забезпечення захищеного обміну даними в сучасних програмах. Взаємодія між клієнтською та серверною частинами системи досягає високого рівня захисту від зовнішніх загроз, оскільки дані залишаються зашифрованими на всіх етапах їх передачі та обробки, гарантуючи користувачам безпечний доступ до функціоналу системи та надійне збереження конфіденційної інформації.

3.3 Тестування та аналіз ефективності

Тестування компонентів системи безпеки є ключовим етапом для забезпечення надійності та захисту даних у реалізованій архітектурі. Основна увага приділяється перевірці стійкості до зовнішніх атак, а також продуктивності системи за умови використання активного шифрування та використання проксі-сервера.

Тестування безпеки передбачає виявлення вразливостей, здатних призвести до порушення безпеки системи, включаючи перевірку на стійкість до таких типових загроз, як несанкціонований доступ, атаки на канал зв’язку та потенційні вразливості у захищених маршрутах. Це дозволяє переконатися, що реалізовані засоби захисту функціонують належним чином і захищають дані від зовнішніх загроз.

Крім того, аналіз продуктивності дає змогу оцінити, як шифрування та маршрутизація через проксі впливають на швидкість обробки запитів і стабільність системи під навантаженням. Важливо забезпечити, щоб впроваджені механізми безпеки не спричиняли значного зниження, особливо за умови активного використання системи.

3.3.1 Мануальне тестування та огляд інтерфейсу

Мануальне тестування – це процес перевірки програмного забезпечення, який виконується вручну без використання автоматизованих інструментів. Воно дозволяє детально оцінити роботу системи з точки зору користувача, включаючи функціональність, зручність використання та стабільність роботи.

Мануальне тестування проводилося для перевірки базових функцій застосунків, зокрема, входу в систему, створення, редагування та видалення об’єктів ситсеми. Крім цього, оцінювалася загальна стабільність роботи застосунків, відгук інтерфейсу на дії користувача. Результати тестування занесено у таблицю 3.1.

Таблиця 3.1 – Результати тестування інтерфейсу

№	**Опис тест-кейсу**	**Очікуваний результат**	**Фактичний результат**	**Оцінк**
1	**2**	**3**	**4**	**5**
1	авторизація	Успішний перехід до інтерфейсу	Аналогічний результат	пройдено
2	Перехід на авторизовану сторінку, без доступу до неї	Переведення на сторінку логіна	Аналогічний результат	пройдено
3	Вихід з інтерфейсу системи	Переведення на сторінку логіна	Аналогічний результат	пройдено

Продовження таблиці 3.1

1	2	3	4	5
4	Створення нового змагання	Додано до бази та оновлено список	Аналогічний результат	пройдено
5	Редагування змагання	Оновлено в базі та оновлено список	Аналогічний результат	пройдено
6	Видалення змагання	Видалено з бази та оновлено список	Аналогічний результат	пройдено
7	Відображення тільки активних змагань	Після переключення режиму, в списку тільки активні змагання	Аналогічний результат	пройдено
8	Додавання дистанції	Додано до бази та оновлено список	Аналогічний результат	пройдено
9	Видалення дистанції	Видалено з бази та оновлено список	Аналогічний результат	пройдено
10	Фільтрація дистанцій по змаганню	Оновлено список дистанцій, де відображені тільки дистанції змагання	Аналогічний результат	пройдено
11	Додавання нормативу	Додано до бази та оновлено список	Аналогічний результат	пройдено
12	Редагування нормативу	Оновлено в базі та оновлено список	Аналогічний результат	пройдено
13	Видалення нормативу	Видалено з бази та оновлено список	Аналогічний результат	пройдено
14	Додавання тренера	Додавано до бази та надано доступ системи за вказаними даними, оновлено список	Аналогічний результат	пройдено
15	Редагування тренера	Оновлено в базі та змінені дані для доступу в систему, список оновлено	Аналогічний результат	пройдено
16	Видалення тренера	Видалено з бази, закрито доступ до системи та оновлено список	Аналогічний результат	пройдено

Продовження таблиці 3.1

1	2	3	4	5
17	Додавання учасника до команди тренера	Учасника додано до команди та внесено у заявку на змагання, список оновлено	Аналогічний результат	пройдено
18	Редагування розряду учасника	Учаснику оновлено розряд та колонка оновлена у списку	Аналогічний результат	пройдено
19	Додавання існуючого учасника до змагання	Учасника внесено у заявку на змагання.	Аналогічний результат	пройдено
20	Видалення учасника з заявки на змагання	Учасника видалено з заявки на змагання та оновлено список заявок	Аналогічний результат	пройдено
21	Редагування особистих даних учасника	Особисті дані змінені у базі та рядок учасника оновлено	Аналогічний результат	пройдено
22	Фільтрація протоколів по змаганню	Протокол успішно відфільтровано по змаганню	Аналогічний результат	пройдено
23	Зміна результату запливу у протоколі змагань	Протокол успішно оновлений з зміненими результатами запливу	Аналогічний результат	пройдено

3.3.2 Тестування безпеки даних під час передачі за допомогою REST API

Тестування безпеки системи було спрямоване на перевірку стійкості передачі даних між клієнтом і сервером із фокусом на захист від несанкціонованого доступу, збереження конфіденційності й цілісності даних. Тестування охопило реалізацію гібридного шифрування, де AES забезпечує шифрування даних, а RSA – передачу ключів, що відповідає вимогам безпеки для обраної архітектури.

OWASP ZAP використовувався для автоматизованого тестування безпеки, що включало сканування REST API та виявлення можливих вразливостей у налаштуваннях передачі даних [37]. Він дозволив провести аналіз на стійкість до атак і перевірити можливі витоки інформації.

Перед автоматизованим тестуванням, виконана перевірка коректності роботи передачі даних з використанням шифрування. Спочатку через інтерфейс користувача заповнюються необхідні дані (рис. 3.3).

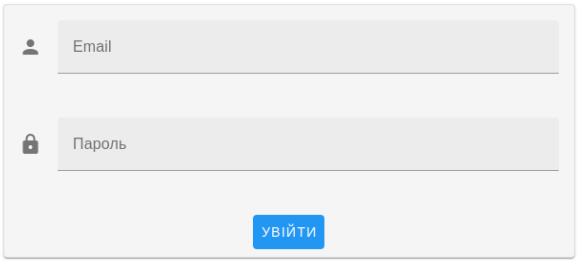

Рисунок 3.3 – Заповнення користувачем даних

Після натискання на кнопку увійти, перевіряється, що дані відправляються на проксі-сервер та хост відправлення є вірним (рис. 3.4).

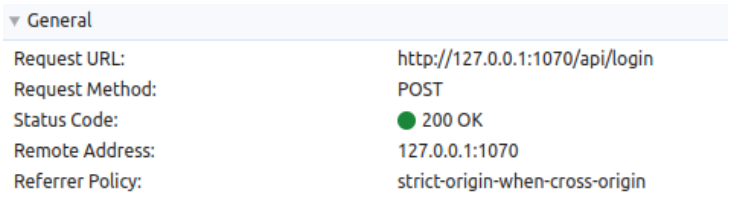

Рисунок 3.4 – Відправка на проксі-сервер

Сформовані JSON дані повністю співпадають з даними, що було введено у формі спочатку (рис. 3.5).

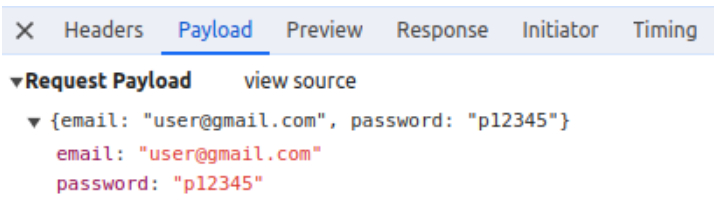

Рисунок 3.5 – Сформовані дані для проксі-сервера

Дані шифруються на стороні Electron за допомогою алгоритму AES та публічного RSA ключа, симетричний ключ передається разом із запитом. У консолі виведено зашифроване тіло запиту, заголовки, необхідні для шифрування та розшифрування, а також створену випадкову IV послідовність. Оброблені дані відповідають патерну та відправлені на сервер (рис. 3.6).

Рисунок 3.6 – Зашифровані дані на стороні Electron

Після відправлення зашифрованих даних вони надійшли на сервер, де розшифровані за допомогою приватного RSA ключа. На боці сервера виконана перевірка заголовків та обробка зашифрованого контенту. Сформована відповідь співпадає патерном зашифрованої відповіді (рис. 3.7).

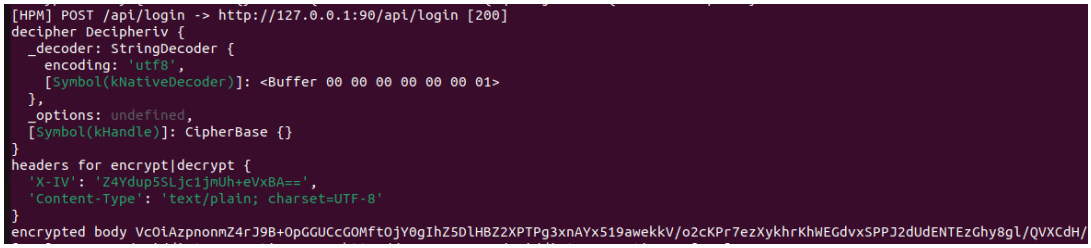

Рисунок 3.7 – Зашифрована відповідь від сервера

Electron приймає відповідь, розшифровує і відправляє клієнту (рис. 3.8).

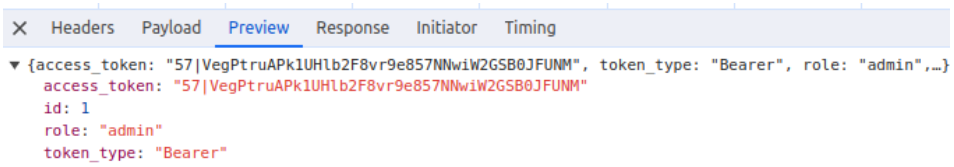

Рисунок 3.8 – Розшифрована відповідь від сервера

Результати вище підтверджують, що шифрування та розшифрування даних працює коректно на всіх етапах передачі, що свідчить про правильність роботи компонента.

Наступним кроком є тестування безпеки передачі даних для десктопного та вебзастосунка, воно було використано з OWASP ZAP, що дозволило виявити наявні вразливості та оцінити рівень захисту системи [38]. Результати для кожного застосунку відображені у відповідних звітах, які підтверджують наявність певних ризиків, що потребують уваги.

У десктопному застосунку виявлено кілька важливих аспектів (рис. 3.9). Середній рівень ризику стосується відсутності заголовків політики безпеки вмісту, використання символів-джокерів у директивах CSP та відсутності заголовка захисту від кліків. Ці вразливості можуть створювати можливості для атак, таких як кліковий джекінг і витік даних через неповну політику безпеки. Низький рівень ризику включає розкриття інформації через заголовок «X-Powered-By», відсутність заголовка «X-Content-Type-Options», що може призвести до атаки типу «MIME-sniffing», а також розкриття «Unix» міток часу.

Name	Risk Level	Number of Instances
CSP: Wildcard Directive	Medium	2
Content Security Policy (CSP) Header Not Set	Medium	1
Missing Anti-clickjacking Header	Medium	1
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	6
Timestamp Disclosure - Unix	Low	1
X-Content-Type-Options Header Missing	Low	4
Information Disclosure - Suspicious Comments	Informational	5
Modern Web Application	Informational	1

Рисунок 3.9 – Результати тестування десктоп застосунку з OWASP ZAP

У вебзастосунку було виявлено більше загроз (рис. 3.10), серед яких високий рівень ризику, що стосується потенційного витоку метаданих хмари. Це є серйозною загрозою, яка може призвести до несанкціонованого доступу до конфіденційних даних. Середні рівні ризику включають ті самі проблеми,

що й у десктопному застосунку, а саме відсутність заголовків CSP, використання символів-джокерів у директивах CSP та відсутність заголовка захисту від кліків. Додатково було виявлено неправильну конфігурацію міждоменних запитів, що може створювати можливості для атак через сторонні домени.

Name	Risk Level	Number of Instances
Cloud Metadata Potentially Exposed	High	1
CSP: Wildcard Directive	Medium	2
Content Security Policy (CSP) Header Not Set	Medium	1
Cross-Domain Misconfiguration	Medium	6
Missing Anti-clickjacking Header	Medium	1
Timestamp Disclosure - Unix	Low	1
X-Content-Type-Options Header Missing	Low	4
Information Disclosure - Suspicious Comments	Informational	5
Modern Web Application	Informational	1

Рисунок 3.10 – Результати тестування вебзастосунку з OWASP ZAP

Загальний аналіз показав, що десктопний застосунок мав кращий результат у порівнянні з вебзастосунком, оскільки у нього не було виявлено критичних загроз, подібних до витоку метаданих, як у випадку з вебзастосунком. Проте обидва типи програм потребують подальшої роботи над покращенням налаштувань безпеки заголовків, політики безпеки вмісту та захисту від клікових атак.

3.3.3 Тестування продуктивності

Тестування продуктивності було проведено для оцінки впливу шифрування на час відповіді системи. Порівнювалися два типи реалізацій: вебзастосунок без шифрування та десктопний застосунок з активованим шифруванням. Отримані результати наведені у таблиці 3.2, яка включає порівняння часу виконання для кожного запиту.

Таблиця 3.2 – Результати тестування часу виконання запитів

№	Запит	Тип запиту	Час виконання без шифрування	Час виконання з шифруванням
1	**2**	**3**	**4**	**5**
1	/login	POST	135 мс	254 мс
2	/competitions	POST	98 мс	152 мс
3	/competitions/{id}	PUT	106 мс	194 мс
4	/competitions	GET	92 мс	150 мс
5	/competitions/{id}	DELETE	88 мс	140 мс
6	/competition-type	POST	98 мс	157 мс
7	/competition-type/{id}	PUT	95 мс	144 мс
8	/competition-type/{id}	GET	92 мс	142 мс
9	/competition-type/{id}	DELETE	151 мс	141 мс
10	/competition-type	GET	91 мс	146 мс
11	/register	POST	148 мс	196 мс
12	/coaches/	POST	88 мс	141 мс
13	/coaches/{id}	PUT	98 мс	148 мс
14	/coaches/{id}	GET	165 мс	171 мс
15	/coaches/{id}	DELETE	91 мс	140 мс
16	/coaches	GET	89 мс	150 мс
17	/participants	POST	93 мс	203 мс
18	/heat	POST	120 мс	184 мс
19	/participants/{id}	GET	89 мс	135 мс
20	/participants/{id}	PUT	89 мс	156 мс
21	/heat	GET	148 мс	152 мс
22	/teams	GET	139 мс	146 мс
23	/heat/{id}	PUT	106 мс	157 мс
24	/heat/{id}	GET	90 мс	143 мс

Продовження таблиці 3.2

1	2	3	4	5
25	/heat/{id}	DELETE	91 мс	143 мс
26	/distance/{id}	GET	106 мс	166 мс
27	/distance/{id}	DELETE	98 мс	118 мс

На основі зібраних даних із таблиці, що містить час виконання запитів із шифруванням та без нього, було проведено порівняльний аналіз. Результати демонструють, що у випадку використання шифрування середній час відповіді збільшується, що очікувано, оскільки додаткові криптографічні операції вимагають часу на обробку [39, 40].

Для отримання різниці часу виконання формалізовано процес для розрахунку відсоткового збільшення часу виконання запитів:

$$R = \frac{\sum T_{re} - \sum T_r}{\sum T_r} \times 100\%, \quad (3.1)$$

де R – відсоткова різниця часу;

T_{re} – час виконання запиту з шифруванням;

T_r – час виконання запиту без шифрування.

На основі обчислень загальний час виконання всіх запитів без шифрування становив 2894 мс, тоді як час виконання з шифруванням склав 4269 мс. Відповідно, підставивши ці значення у (3.1) для розрахунку відсоткового збільшення часу виконання, було отримано 47.5%.

Це означає, що загальний час виконання запитів збільшився на 47.5% при використанні шифрування. Хоча цей приріст часу є помітним, він не був критичним для роботи системи та залишився в межах прийнятного діапазону. Це підтверджує, що використання шифрування для забезпечення безпеки є доцільним рішенням, яке не спричиняє значного зниження продуктивності системи.

ВИСНОВКИ

У рамках кваліфікаційної роботи було проведено дослідження компонентних моделей для реалізації проєктів, спрямованих на підвищення безпеки даних у сучасних інформаційних системах. Наукова новизна роботи полягає в застосуванні альтернативного підходу до розробки застосунків, що передбачає використання гібридного шифрування через проксі-сервер для забезпечення безпечної передачі даних та зберігання інформації.

Були реалізовані механізми гібридного шифрування AES та RSA, що дозволило підвищити рівень захищеності даних під час їх передачі. Також був налаштований проксі-сервер для посилення безпеки під час комунікації між клієнтом та сервером. Застосування цих інструментів допомогло зменшити ризики несанкціонованого доступу до інформації, забезпечивши високий рівень конфіденційності та цілісності даних.

Тестування, проведене з використанням спеціалізованих інструментів для аналізу вразливостей, підтвердило підвищення безпеки при використання компонента шифрування та компонента проксі-сервера. Була проаналізована продуктивність системи в умовах активного шифрування, і хоча час виконання операцій зріс, це не призвело до критичного зниження швидкості роботи. Порівняння вебзастосунку і десктоп застосунку продемонструвало перевагу десктоп версії у сфері безпеки.

Таким чином, виконане дослідження показало, що використання компонентних моделей дозволяє створювати гнучкі та безпечні інформаційні системи, адаптовані для різних умов використання. Впроваджені рішення можуть бути ефективно застосовані в проєктах, де питання безпеки даних мають ключове значення, забезпечуючи захищеність інформації.

Результати дослідження апробовано у вигляді тез доповіді під час XI Міжнародної науково-практичної конференція «Modern generation: current problems, experience, development prospects» [41].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Dubnitskiy V., Kobylin A., Kobylin O., and Kushneruk Y. (2021) Excel–орієнтована процедура для обчислення значень спеціальних функцій з інтервальним аргументом, заданим в гіперболічній формі, *Advanced Information Systems*, vol. 5, no. 4, pp. 116-123.

2. Dyadun S., Bodyanskiy Y.V., Korobchynskyi K., and Kobylin O. (2022) Methods For Increasing The Reliability Of The Functioning Of Pipeline System, *ProfIT AI* , vol. 3348, pp. 104-109.

3. Бутенко, П. В. (2024) Обробка рукописного тексту з зображень. *Радіоелектроніка та молодь у XXI столітті. Т. 7: Конференція "Комп'ютерний зір, системний аналіз та математичне моделювання": матеріали 28-го Міжнар. молодіж. форуму, 16–18 квітня 2024 р.*, pp.21-23 .

4. Бобейко, К. С., & Кобилін, О. А. (2024) Дослідження методів оптимізації процесу розробки музичного застосунку за допомогою аналізу даних. *The 20th International scientific and practical conference "Trends in the development of quality training of future specialists"(May 21–24, 2024) Oslo, Norway. International Science Group. 2024,* pp. 351-353.

5. Tvoroshenko, I. (2019) Development of models of spatial analysis of status of interactive processes of complex systems.

6. Скляренко, О., Савченко, Я., Литвиненко, Л., & Сушинський, О. (2024) Архітектурні підходи до розробки масштабованих веб-застосунків. *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»*, *4*(24), pp. 341-350.

7. Ваврук, Є. Я. (2019) Компонентно-орієнтований підхід в програмуванні Java-додатків.

8. Дубинчак, М. В., Крупельницький, Л. В., & Городецька, О. С. (2024) Json web tokens як інструмент для безпечної авто-ризації у веб-додатках: аналіз та перспективи.

9. Jin, W., & Kim, D. (2020) Interworking Proxy Based on OCF for Connecting Web Services and IoT Networks, *Journal of Communications*, *15*(2), pp. 192-197.

10. Машталір, С. В., & Ніколенко, О. В. (2023) Data preprocessing and tokenization techniques for technical Ukrainian texts, *Прикладні аспекти інформаційних технологій*, *6*(3), pp. 318-326.

11. Яцишин, В. В., and Хацюр, В. В. (2020) Переваги компонентно-орієнтованого програмування. *Матеріали VIII науково-технічної конференції «Інформаційні моделі, системи та технології»*, pp. 124-125.

12. Ревенчук, І. А., and Стешко, В. Ю. (2022) Architectural solutions and optimization methods to improve the performance of node. js and vue. js applications, *Біоніка інтелекту*, *1*(98), pp. 64-69.

13. Hassan, W. H. (2019) Current research on Internet of Things security: A survey. *Computer networks*, *148*, pp. 283-294.

14. Dyadun, S., Yakovlev, S., & Kobylin, O. (2022) Mathematical Modeling of Steady Flow Distribution in Water Supply Networks with Pumping Stations and Regulating Capacitances, *ProfIT AI* , pp. 78-83.

15. Dubnitskiy, V., Kobylin, A., Kobylin, O., Kushneruk, Y., and Sheviakov, I. (2022) Обчислення значень функцій комплексної змінної з інтервальним аргументом, визначеним в гіперболічній формі. *Advanced Information Systems*, *6*(3), pp. 83-91.

16. Dubnitskiy, V., Kobylin, A., Kobylin, O., Kushneruk, Y., and Khodyrev, A. (2023). Обчислення значень функції Харрінгтона (функції бажаності) при інтервальному визначенні її аргументів. *Advanced Information Systems*, *7*(1), 71- 81.

17. Кобилін, О. А., Путятіна, О. Є., & Гарячий, М. В. (2020). Filtration of parameters of the Heston model. *Системи обробки інформації*, 4(163), pp. 48- 55.

18. Tvoroshenko, I., Gorokhovatskyi, V., Kobylin, O., & Tvoroshenko, A. (2023). Application of deep learning methods for recognizing and classifying culinary dishes in images.

19. Бутенко П.В. Розробка застосунку оптичного розпізнавання символів для отримання тексту з фотографій: кваліфікаційна робота першого (бакалаврського) рівня вищої освіти: 122 Комп'ютерні науки. Харків, 2023. 74 с.

20. Sood, R., and Kaur, H. (2023). A literature review on rsa, des and aes encryption algorithms. *Emerging Trends in Engineering and Management*, pp. 57- 63.

21. Hamza, A., and Kumar, B. (2020). A review paper on DES, AES, RSA encryption standards. *2020 9th International Conference System Modeling and Advancement in Research Trends (SMART)*, pp. 333-338.

22. Husni, M., Ciptaningtyas, H. T., Suadi, W., Ijtihadie, R. M., Anggoro, R., Salam, M. F., & Arifiani, S. (2020). Security audit in cloud-based server by using encrypted data AES-256 and SHA-256. *IOP Conference Series: Materials Science and Engineering*, vol. 830, no. 3, pp. 1-6.

23. Abdul Hussien, F. T., Rahma, A. M. S., & Abdul Wahab, H. B. (2021). A Secure Environment Using a New Lightweight AES Encryption Algorithm for E-Commerce Websites. *Security and Communication Networks*, 2023, pp. 1-15

24. Hoobi, M. M., Sulaiman, S. S., & AbdulMunem, I. A. (2020). Enhanced Multistage RSA Encryption Model. *IOP Conference Series: Materials Science and Engineering* , vol. 928, no. 3, pp. 1-8.

25. Dubnitskiy, V., Kobylin, A., & Kobylin, O. (2021). Виконання на мобільних пристроях арифметичних операцій з використанням аксіом класичного та нестандартного інтервального аналізу. *Advanced Information Systems*, *5*(3), pp. 128-136.

26. Кобилін, О. А., & Путятіна, О. Є. (2024). Real-time image denoising corrupted by shot-noise. *Системи обробки інформації*, 1(176), pp. 46-51.

27. Chavan, P. R., & Pawar, S. (2021). Comparison Study Between Performance of Laravel and Other PHP Frameworks. *International Journal of Research in Engineering, Science and Management*, *4*(10), pp. 27-29.

28. Ariyanto, Y., Rachmad, M. F. F., & Puspitasari, D. (2024). Laravel framework and native PHP: Comparison in the crea-tion of rest API. *Matrix: Jurnal Manajemen Teknologi Dan Informatika*, *14*(2), pp. 66-73.

29. Bielak, K., Borek, B., & Plechawska-Wójcik, M. (2022). Web application performance analysis using Angular, React and Vue. js frameworks. *Journal of Computer Sciences Institute*, vol. *23*, pp. 77-83.

30. Rapley, A., Bellekens, X., Shepherd, L. A., & McLean, C. (2018). Mayall: a framework for desktop JavaScript auditing and post-exploitation analysis. *Informatics*, vol. 5, no. 4, pp. 46.

31. Моруга, Д. І., & Ревенчук, І. А. (2024). Research of methods for development of graphical user interface in cross-platform applications. In *VII International scientific and practical conference «Scientific Research: Theoretical Foundations and Practical Applications»(January 24-26, 2024) Vienna, Austria, International Scientific Unity. 2024*, pp. 165-169.

32. Супрун, А. Є. (2024) Розробка веб-застосунку «Соціальна мережа для публікації творчих робіе». *28-й Міжнародний молодіжний форум «Радіоелектроніка і молодь у XXI столітті». Зб. матеріалів форуму. Т. 7. Харків: ХНУРЕ. 2024*, pp. 114-115.

33. Kredpattanakul, K., & Limpiyakorn, Y. (2019). Transforming javascript-based web application to cross-platform desktop with electron. *Information Science and Applications 2018: ICISA 2018*, pp. 571-579.

34. Saundariya, K., Abirami, M., Senthil, K. R., Prabakaran, D., Srimathi, B., and Nagarajan, G. (2021). Webapp service for booking handyman using mongodb, express JS, react JS, node JS. *2021 3rd International Conference on Signal Processing and Communication (ICPSC)*, pp. 180-183.

35. Shukla, A. (2023). Modern JavaScript Frameworks and JavaScript's Future as a Full-Stack Programming Language. *Journal of Artificial Intelligence & Cloud Computing. SRC/JAICC-156*, 144, pp. 2-5.

36. Tvoroshenko, I., & Kharchenko, A. (2021). Some aspects of modern development for sign language recognition systems.

37. Tvoroshenko, I. S., & Maksimenko, H. (2021). To the question of analysis of existing mechanisms of web application testing.

38. Ehsan, A., Abuhaliqa, M. A. M., Catal, C., & Mishra, D. (2022). RESTful API testing methodologies: Rationale, challenges, and solution directions. *Applied Sciences, 12*(9), pp. 2-14.

39. Yakovleva, O., Kovtunenko, A., Liubchenko, V., Honcharenko, V., and Kobylin, O. (2023). Face Detection for Video Surveillance-based Security System. *COLINS, 3*, pp. 69-86.

40. Tvoroshenko, I., & Gorokhovatskyi, V. (2022). The Application of Hybrid Intelligence Systems for Dynamic Data Analysis, *International Journal of Engineering and Information Systems*, 6(2), pp. 40-48.

41. Бутенко П. В. (2024) Компонентний підхід у захисті інформаційних систем, *Proceedings of the XI International Scientific and Practical Conference. Seville, Spain. 2024*. pp. 39-42.