Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

Магістерська кваліфікаційна робота на тему:
«Метод та засіб захищеного обміну корпоративною інформацією розробників програмного забезпечення»

Виконав: студент 2 курсу групи 2БС-22м
спеціальності 125 Кібербезпека
_____ Олексій ПАЛІЙ

Керівник: к. т. н., доцент каф. ЗІ
_____ Юрій БАРИШЕВ
«13» грудня 2023 р.

Опонент: к. т. н., доцент каф. ПЗ
_____ Денис КАТЄЛЬНІКОВ
«13» грудня 2023 р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«14» 12 2023 р.

Вінниця ВНТУ – 2023 року

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти II (магістерський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ,
д. т. н., проф.
____Володимир ЛУЖЕЦЬКИЙ
«_19_» ____09_____2023 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Палію Олексію Миколайовичу

1. Тема роботи: «Метод та засіб захищеного обміну корпоративною інформацією розробників програмного забезпечення»
керівник роботи: Баришев Юрій Володимирович, к. т. н., доцент кафедри ЗІ, затверджені наказом ректора ВНТУ від 18 вересня 2023 №247.

2. Строк подання студентом роботи: 13 грудня 2023р.

3. Вихідні дані до роботи:
   - мова програмування – об'єктно-орієнтована;
   - тип програми – хмарне сховище;
   - підтримка різних типів файлів;
   - шифрування даних;
   - статичне тестування безпеки розробленого застосунку.
4. Зміст текстової частини: Вступ. 1. Аналіз відомих рішень. 2. Метод захищеного обміну корпоративною інформацією розробників програмного забезпечення. 3. Розробка засобу. 4. Тестування засобу. 5. Економічна частина. Висновки. Перелік використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: Актуальність, мета та задачі МКР (плакат А4). Наукова новизна та практична цінність (плакат А4). Метод шифрування даних (плакат А4). Модель розмежування прав доступу (плакат А4). Архітектура клієнтської частини (плакат А4). Архітектура серверної частини (плакат А4). Алгоритми шифрування (плакат А4). Алгоритм автентифікації/реєстрації (плакат

А4). Результати блокового тестування (плакат А4). Результати інтеграційного тестування (плакат А4). Результати статичного тестування безпеки (плакат А4).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Юрій БАРИШЕВ, к. т. н., доц. каф. ЗІ	01.09	15.09
2	Юрій БАРИШЕВ, к. т. н., доц. каф. ЗІ	01.09	10.10
3	Юрій БАРИШЕВ, к. т. н., доц. каф. ЗІ	01.09	16.11
4	Юрій БАРИШЕВ, к. т. н., доц. каф. ЗІ	01.09	07.12
5	Ольга РАТУШНЯК, к. т. н., доц. каф. ЕПВМ	01.09	16.11

7. Дата видачі завдання 1 вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	01.09.2023 – 10.09.2023	
2	Аналіз літературних джерел за напрямком магістерської кваліфікаційної роботи	10.09.2023 – 15.09.2023	
3	Науково-технічне обґрунтування	16.09.2023 – 22.09.2023	
4	Розробка технічного завдання	23.09.2023 – 29.09.2023	
5	Розробка рішень	30.09.2023 – 12.10.2023	
6	Практична реалізація, моделювання, експериментування, результати	14.10.2023 – 10.11.2023	
7	Розробка розділу економічного обґрунтування доцільного розробки	11.11.2023 – 17.11.2023	
8	Аналіз виконання ТЗ, висновки	18.11.2023 – 24.11.2023	
9	Оформлення пояснювальної записки	25.11.2023 – 30.11.2023	
10	Попередній захист та доопрацювання МКР	28.11.2023 – 10.12.2023	
11	Перевірка магістерської роботи на наявність плагіату	02.12.2023 – 10.12.2023	
12	Представлення МКР до захисту	11.12.2023 – 14.12.2023	
13	Захист МКР	14.12.2023 – 21.12.2023	

Студент _____ Олексій ПАЛІЙ

Керівник роботи _____ Юрій БАРИШЕВ

АНОТАЦІЯ

Магістерська кваліфікаційна робота складається з 139 сторінок формату А4, на яких є 24 рисунка, 9 таблиць, 27 формул, список використаних джерел містить 32 найменування.

Магістерська кваліфікаційна робота присвячена розробці методу та засобу обміну корпоративною інформацією розробників програмного забезпечення. У роботі проаналізовано сучасні засоби обміну корпоративною інформацією, а також основні загрози кібербезпеці для таких застосунків. У межах роботи виконано теоретико-множинний опис методу обміну, описано метод шифрування та модель розмежування прав доступу. Спроектовано архітектуру клієнтської та серверної частини застосунку, а також описано основні алгоритми роботи. Після розробки схем функціонування програмного засобу в цілому і алгоритмів його окремих складових здійснено програмну реалізацію. Проведено автоматизоване блокове та інтеграційне тестування, статичне тестування безпеки та тестування застосунку загалом.

Ключові слова: корпоративна інформація, інформаційна безпека, розмежування прав доступу, захист корпоративної інформації, автентифікація, верифікація

ABSTRACT

The master's qualification work consists of 139 pages of A4 format, which contains 24 figures, 9 tables, 27 formulas, the list of references contains 32 items.

The master's qualification work is devoted to the development of methods and tools for exchanging corporate information of software developers. The work analyses modern means of exchanging corporate information, as well as the main cybersecurity threats to such applications. The paper provides a set-theoretic description of the exchange method, describes the encryption method and the model of access rights differentiation. The architecture of the client and server parts of the application is designed, and the basic operation algorithms are described. After developing the schemes of functioning of the software tool as a whole and the algorithms of its individual components, the software implementation was carried out. Automated unit and integration testing, static security testing, and testing of the application as a whole were performed.

Keywords: corporate information, information security, separation of access rights, protection of corporate information, authentication, verification

ЗМІСТ

ВСТУП

У сучасному світі інформаційних технологій, де обсяг корпоративної інформації зростає на зачіпку, а технології надають можливості для її миттєвого обміну та збереження, питання безпеки та конфіденційності інформації виявляються на першому плані. Особливо актуальним стає завдання забезпечення безпеки обміну корпоративною інформацією, що стосується не лише великих корпорацій та урядових структур, але й невеликих підприємств, а також індивідуальних розробників програмного забезпечення [1].

Світові тенденції свідчать про постійне зростання кількості інформації, яка обмінюється між організаціями та користувачами. З цим зростанням обсягів інформації збільшується ризик її незаконного доступу, перехоплення, витоку або пошкодження. Ця ситуація ставить підприємства та розробників програмного забезпечення перед нагальною необхідністю забезпечити стійкий захист обміну корпоративною інформацією [1].

Розробники програмного забезпечення стають центральною постаттю в реалізації безпеки обміну інформацією. Вони відповідають за створення програм, які обробляють, передають та зберігають важливі дані. У своїй роботі вони не тільки повинні розуміти технічні аспекти безпеки, але і розробляти стратегії та заходи для ефективного захисту корпоративної інформації від зовнішніх загроз і внутрішніх порушень безпеки [1-2].

Розробники програмного забезпечення зустрічаються з низкою викликів, пов'язаних з безпекою, таких як захист даних від несанкціонованого доступу, забезпечення конфіденційності під час передачі інформації через мережі, а також забезпечення цілісності даних. Однак разом з викликами приходить і можливість впроваджувати нові методи та технології для захисту корпоративної інформації [2], тому актуально розробляти засіб захисту обміну корпоративною інформацією розробників програмного забезпечення.

Дослідження, присвячене методам та засобам захищеного обміну корпоративною інформацією розробниками програмного забезпечення, представляє істотний прорив у сфері інформаційної безпеки та управління ризиками в контексті розробки ПЗ. Воно вирішує два ключові завдання. По-перше, це інтеграція безпеки в процес розробки програмного забезпечення з самого початку. Це означає, що заходи безпеки не розглядаються як післямовлення, але стають невід'ємною частиною розробки. Це робить процес більш ефективним і гарантує відсутність слабких місць у безпеці.

Захищений обмін корпоративною інформацією має значення в ряді практичних аспектів. В першу чергу, він допомагає ефективно захищати корпоративні дані від несанкціонованого доступу та витоку інформації. Це допомагає компаніям зберегти конфіденційність і цілісність своїх даних. Крім того, цей підхід може суттєво знизити витрати на реагування на інциденти безпеки, оскільки більшість проблем виявляються та усуваються ще до того, як вони можуть спричинити серйозний збиток. Окрім цього, впровадження засобів захищеного обміну сприяє підвищенню довіри клієнтів та партнерів до організації.

Тема дослідження взаємодіє з іншими галузями досліджень, такими як кібербезпека в розробці програмного забезпечення, безпека даних та захист від витоку інформації, а також технології блокчейну та шифрування. Вона поєднує найкращі практики та методи з цих галузей, створюючи комплексний підхід до безпеки в розробці ПЗ.

Об’єктом дослідження є процес розробки прикладних програм.

Предметом дослідження є методи та засоби захисту комерційної інформації.

Метою даної роботи є покращення захисту інформації у хмарних сервісах, шляхом впровадження механізмів захисту від несанкціонованого доступу до цих даних.

Для досягнення мети потрібно розв’язати такі задачі:

– проаналізувати проблеми хмарних середовищ;

– створити модель розмежування прав доступу;

– розробити архітектуру клієнтських та серверних частин;

– розробити алгоритми модуля шифрування;

– розробити алгоритми роботи засобу.

Наукова новизна роботи полягає в такому:

– удосконалено метод захищеного обміну корпоративною інформацією розробників програмного забезпечення, який на відміну від відомих використовує передавання та зберігання всієї інформації незалежно від її виду в захищеному вигляді на стороні сервера, що дозволило врахувати виробничі процеси розробки програми, зберігаючи захищеність корпоративної інформації;

– отримала подальший розвиток модель розмежування прав доступу, яка на відмінну від відомих, створює попередні налаштування правил розмежування прав доступу у вигляді ролей з подальшим використанням моделі Гаррісона-Руззо-Ульмана, що дозволило розмежовувати права доступу до інформаційних ресурсів в момент їх створення.

Практична цінність програми полягає в такому:

– розроблено програмний засіб для захищеного обміну корпоративною інформацією;

– набори сценаріїв для блокового та інтеграційного тестування засобів обміну корпоративною інформацією;

– модуль статичного тестування безпеки застосунків, розроблених мовою JavaScript.

Результати отримані у магістерській кваліфікаційній роботі були представлені на таких конференціях:

– LI Науково-технічна конференція ІТКІ 2022 року, Вінницького національного технічного університету [3];

– LI Науково-технічна конференція факультету МТ 2022 року, Вінницького національного технічного університету [4];

– Молодь у науці: дослідження, проблеми, перспективи 2024 року, Вінницького національного технічного університету [5].

1 АНАЛІЗ ВІДОМИХ РІШЕНЬ

1.1 Аналіз засобів обміну корпоративною інформацією

В сучасному бізнес-середовищі обмін корпоративною інформацією є ключовим аспектом успішного функціонування підприємств. Зростання обсягів даних та їх значущість роблять актуальним завдання аналізу та вдосконалення засобів обміну корпоративною інформацією.

Засоби обміну корпоративною інформацією можуть бути надзвичайно корисними для ефективної співпраці та обміну даними в корпоративному середовищі.

GitHub - це платформа для розробки програмного забезпечення та керування версіями коду [6]. Основні характеристики:

– Система керування версіями (VCS) - GitHub використовує Git для збереження та відстеження змін у коді [7]. Це дозволяє розробникам працювати над проектами, зберігаючи історію змін.

– Спільна робота - розробники можуть створювати репозиторії для проектів, де зберігається код та файли. Запити на злиття дозволяють команді обговорювати та об'єднувати зміни.

– Хмарне сховище - репозиторії зберігаються в хмарі, що дозволяє забезпечити доступ до них з будь-якого пристрою.

– Інструменти для розробки -включають редагування коду, інструменти для автоматизації інтеграції та неперервної доставки (CI/CD).

– Командна робота - розробники можуть спільно працювати над проектами, обговорюючи код та надаючи права доступу.

– Інтеграція з іншими сервісами - GitHub інтегрується з іншими інструментами розробки.

– Безпека та контроль доступу - GitHub надає інструменти для забезпечення безпеки та контролю доступу до репозиторіїв.

GitHub є незамінним інструментом для розробників і команд, спрощуючи керування кодом та спільну роботу над проектами.

Google Drive - це хмарне сховище та офісний пакет інструментів для зберігання, редагування та спільної роботи над документами [8]. Основні характеристики:

– Зберігання в хмарі - Google Drive дозволяє користувачам зберігати файли та дані в хмарному сховищі, доступному з будь-якого пристрою з інтернет-з'єднанням.

– Офісний пакет - включає в себе Google Docs (текстовий редактор), Google Sheets (таблиці) і Google Slides (презентації) для створення та редагування документів.

– Спільна робота - користувачі можуть спільно працювати над документами, надавати доступ і редагувати файли разом з колегами, а також залишати коментарі.

– Спільний доступ - можливість надавати доступ до файлів та папок і встановлювати рівні прав доступу.

– Інтеграція з Gmail - інтеграція з Gmail дозволяє прикріплювати файли з Google Drive до листів безпосередньо з поштової скриньки.

– Автоматичне збереження та синхронізація - зміни в документах автоматично зберігаються та синхронізуються, забезпечуючи безперервний доступ до останньої версії.

– Завдання та нагадування - можливість створювати завдання та нагадування в Google Keep та Google Tasks, інтегрованих із Google Drive.

– Інтеграція з іншими сервісами - Google Drive інтегрується з іншими Google-сервісами (Google Photos, Google Calendar) та багатьма іншими додатками і платформами.

Google Drive є потужним інструментом для зберігання та спільної роботи над документами та файлами, спрощуючи роботу команд і користувачів над проектами та завданнями.

Slack - це комунікаційна платформа для команд, яка допомагає спростити спільну роботу та обмін повідомленнями в організаціях [9]. Основні характеристики:

– Чат та повідомлення - Slack надає можливість обмінюватися текстовими повідомленнями та документами в режимі реального часу між користувачами та групами.

– Канали (Channels) - користувачі можуть створювати та приєднуватися до різних каналів для групування обговорень за темами чи проектами.

– Інтеграція із зовнішніми сервісами - Slack інтегрується з багатьма іншими інструментами і сервісами, такими як Google Drive, GitHub, Trello тощо, щоб забезпечити зручну спільну роботу та обмін даними.

– Відеоконференції та голосовий чат - Slack має вбудовані можливості відеоконференцій та голосового чату, що полегшують віддалену спільну роботу та комунікацію.

– Повідомлення та завдання - користувачі можуть встановлювати нагадування, створювати завдання та отримувати сповіщення для ефективного керування робочими процесами.

– Спільна робота в реальному часі - Slack дозволяє більш ефективно співпрацювати в реальному часі, вирішувати завдання та швидко вирішувати проблеми.

– Інструменти для аналізу та звітності - Slack надає інструменти для аналізу активності та спільної роботи користувачів.

– Захист та безпека даних - Slack забезпечує рівень захисту та безпеки для конфіденційної інформації.

Slack є важливим інструментом для комунікації та спільної роботи в організаціях, полегшуючи обмін інформацією та співпрацю команд.

Microsoft Teams - це інтегрована платформа для спільної роботи та комунікації в організаціях, яка включає в себе різноманітні інструменти та можливості [10]. Основні характеристики:

– Чат та повідомлення - Microsoft Teams дозволяє користувачам обмінюватися текстовими повідомленнями, файлами та документами в режимі реального часу.

– Канали (Channels) - користувачі можуть створювати канали для організації обговорень за темами чи проектами, подібно до Slack.

– Відеоконференції та голосовий чат - в Microsoft Teams вбудовані можливості відеоконференцій, голосового чату та екранних спільних робіт, що полегшують віддалену спільну роботу та комунікацію.

– Спільна робота в реальному часі - користувачі можуть спільно працювати над документами та таблицями Microsoft Office (Word, Excel, PowerPoint) в режимі реального часу безпосередньо в Teams.

– Інтеграція з іншими сервісами Microsoft - Teams інтегрується з іншими продуктами Microsoft, такими як SharePoint, OneDrive, Outlook, та іншими, для покращення продуктивності.

– Завдання та нагадування - в Teams можна створювати завдання, призначати їх учасникам та встановлювати нагадування.

– Спільна робота з іншими користувачами - з Teams можна спільно редагувати документи, ділитися файлами та отримувати сповіщення про зміни.

– Захист та безпека даних - Microsoft Teams забезпечує високий рівень захисту даних та відповідає стандартам безпеки.

– Аналітика та звітність - Teams надає інструменти для аналізу та звітності про користувачів та активність.

Microsoft Teams є потужним інструментом для спільної роботи та комунікації в організаціях, особливо для тих, хто використовує продукти Microsoft.

Dropbox Business - це хмарне сховище та інструменти спільної роботи, спеціально призначені для корпоративних користувачів [11]. Основні характеристики:

– Зберігання в хмарі - Dropbox Business дозволяє користувачам зберігати файли та дані в безпечному та доступному хмарному сховищі.

– Спільна робота - користувачі можуть спільно працювати над файлами та документами, ділитися ними з колегами та налаштовувати рівні доступу.

– Версіонування та відновлення - Dropbox зберігає історію змін файлів, що дозволяє відновлювати попередні версії та відновлювати файли у випадку помилок.

– Інтеграція з іншими сервісами - Dropbox інтегрується з різними іншими інструментами, такими як Google Workspace, Slack, Trello та іншими.

– Мобільний доступ - користувачі можуть отримати доступ до своїх файлів та даних через мобільні пристрої, що полегшує роботу на віддалених місцях.

– Автоматична синхронізація - зміни у файлах автоматично синхронізуються між різними пристроями.

– Захист даних - Dropbox Business забезпечує високий рівень захисту та шифрування для зберігання даних.

– Запити на завантаження файлів - можливість надсилати запити на завантаження файлів від зовнішніх користувачів, що полегшує збір даних від клієнтів чи партнерів.

– Аналітика та звітність - Dropbox Business надає інструменти для аналізу користувачів та активності файлів.

Dropbox Business є інструментом для зберігання та спільної роботи з файлами в корпоративному середовищі, спрощуючи обмін та співпрацю між користувачами та командами.

SharePoint - це платформа для спільної роботи та управління даними в корпоративному середовищі, розроблена Microsoft [12]. Основні характеристики:

– Зберігання та управління документами - SharePoint дозволяє користувачам зберігати та організовувати документи, файли та дані в централізованому хмарному сховищі.

– Спільна робота - користувачі можуть спільно працювати над документами та файлами, ділитися ними з колегами та встановлювати рівні доступу.

– Версіонування та контроль змін - SharePoint зберігає історію змін файлів, дозволяючи відновлювати попередні версії та контролювати доступ до них.

– Інтеграція з Microsoft 365 - SharePoint інтегрується з іншими продуктами Microsoft, такими як Word, Excel, PowerPoint, Outlook, що спрощує роботу з документами та комунікацію.

– Спеціальні сайти та портали - SharePoint дозволяє створювати спеціальні сайти та портали для організації проектів, бізнес-процесів та звітності.

– Мобільний доступ - користувачі можуть отримати доступ до даних та файлів через мобільні пристрої, що полегшує роботу на віддалених місцях.

– Шаблони та робочі процеси - SharePoint має набір шаблонів та інструменти для створення робочих процесів та автоматизації завдань.

– Захист даних та безпека - SharePoint надає рівень захисту даних та шифрування для забезпечення конфіденційності інформації.

– Аналітика та звітність - SharePoint надає інструменти для аналізу користувачів та активності файлів.

– Завдання та сповіщення - можливість створювати завдання та сповіщення користувачам для організації спільної роботи та слідкування за завданнями.

SharePoint є потужним інструментом для спільної роботи, управління документами та організації даних в корпоративному середовищі, особливо для користувачів, які використовують продукти Microsoft.

Trello - це інтерактивна платформа для керування завданнями та спільної роботи, яка базується на концепції дошок та карток [13]. Основні характеристики:

– Картки та дошки - Trello організовує роботу за допомогою карток, які розташовані на дошках. Картки відображають завдання, проекти або ідеї.

– Списки та колонки - картки розміщуються в списках або колонках на дошці, що дозволяє структурувати завдання.

– Спільна робота - користувачі можуть ділитися дошками з колегами та спільно працювати над картками.

– Призначення та терміни - кожну картку можна призначити конкретному користувачу та встановити терміни виконання завдання.

– Етікетки та фільтри - етікетки дозволяють позначати картки за темами або пріоритетами, а фільтри допомагають швидко знаходити потрібні завдання.

– Коментарі та обговорення - картки підтримують коментарі, що дозволяє користувачам обговорювати деталі завдань.

– Інтеграція з іншими сервісами - Trello інтегрується з багатьма іншими інструментами, такими як Google Drive, Slack, інші сервіси для зручної роботи.

– Діаграми Ганта - Trello може бути інтегрованим з діаграмами Ганта для керування проектами та розкладами.

– Завдання та дедлайни - Trello дозволяє створювати завдання та встановлювати дедлайни для кожної картки.

Trello - це простий та ефективний інструмент для керування завданнями та спільної роботи, особливо підходить для проектного управління та організації робочих процесів.

Після проведення ретельного аналізу різних інструментів для обміну корпоративною інформацією, ми підготували таблицю порівнянь цих засобів (табл. 1.1). В ході цього дослідження було виявлено розмаїття інструментів, які задовольняють різні потреби організацій на сучасному ринку.

GitHub відмінно підходить для розробки програмного забезпечення, Google Drive виявився ідеальним для спільної роботи над документами, Slack та Microsoft Teams дозволяють ефективно комунікувати та спільно працювати в командах, Dropbox Business забезпечує надійне зберігання та спільну роботу з файлами, а SharePoint і Trello ідеально підходять для управління завданнями та проектами. Вибір конкретного інструменту повинен визначатися виходячи з особливостей та потреб конкретної організації.

Таблиця 1.1 – Порівняння хмарних середовищ

Характеристика	GitHub	GDrive	Slack	Microsoft Teams	Dropbox	SharePoint	Trello
Система керування версіями	Використовує Git	Не має, але має версії файлів	Не має VCS	Не має VCS	Не має VCS	Не має VCS	Не має VCS
Хмарне сховище	Так	Так	Ні	Так	Так	Так	Ні
Інтеграція з іншими сервісами	Так	Так	Так	Так	Так	Так	Так
Інструменти для розробки	Редагування коду, CI/CD	Ні	Так	Ні	Ні	Ні	Ні
Командна робота	Так	Так	Ні	Так	Так	Так	Так
Завдання та нагадування	Ні	Так	Так	Так	Так	Так	Так
Відеоконференції та голосовий чат	Ні	Ні	Так	Так	Так	Ні	Ні
Аналітика та звітність	Ні	Ні	Так	Так	Так	Так	Ні
Захист та безпека даних	Так	Так	Так	Так	Так	Так	Ні
Спільна робота в реальному часі	Ні	Ні	Так	Так	Так	Ні	Так
Версіонування та відновлення	Ні	Ні	Ні	Ні	Ні	Так	Ні

1.2 Аналіз загроз для засобів обміну корпоративною інформацією

Спільний обмін корпоративною інформацією стає все важливішим для організацій будь-якого розміру. Ця тенденція підвищує вимоги до забезпечення безпеки та конфіденційності обмінюваної інформації компаніями. У цьому розділі ми розглянемо різні інструменти обміну інформацією та проведемо докладний аналіз потенційних загроз і вразливостей, які становлять ризики для безпеки даних під час їх використання.

Загрози безпеки для GitHub, платформи для керування версіями коду та роботи над програмним забезпеченням, включають наступні аспекти:

– Несанкціонований доступ до коду - загроза полягає в тому, що несанкціоновані користувачі можуть спробувати отримати доступ до приватних репозиторіїв або коду. Для запобігання цьому, GitHub пропонує авторизацію, обмеження доступу та можливість двофакторної автентифікації [14].

– Вразливості в коді - якщо розробник не виявить та не виправить вразливості в коді, це може призвести до атак та порушення безпеки застосунку або системи [7].

– Атаки на репозиторії - GitHub піддається ризику атак, таких як DDoS (розподілені атаки на відмову в обслуговуванні) або спроби злому. Спільність розробників повинна бути готовою реагувати на такі загрози.

– Підробка коду - атаки, які полягають у підробці коду або комітів, можуть завдати шкоди довірі та безпеці проекту. Сам GitHub не завжди перевіряє вміст комітів, тому це може бути важливою відповідальністю для розробників [14].

– Безпека облікових записів - вразливості в паролях або несанкціоновані доступи до облікових записів розробників можуть призвести до порушення безпеки. Рекомендується використовувати сильні паролі та двофакторну авутентифікацію.

– Незахищені API ключі - помилково розкриті або незахищені API ключі можуть призвести до небезпеки для безпеки застосунку.

GitHub надає інструменти та можливості для забезпечення безпеки, але важливою є активна участь розробників у виявленні та виправленні вразливостей. Перед

використанням GitHub для управління кодом, слід вивчити та впровадити найкращі практики забезпечення.

Загрози безпеки для Google Drive, подібно до багатьох хмарних служб зберігання даних, включають наступні аспекти:

– Несанкціонований доступ - найпоширенішою загрозою є несанкціонований доступ до даних. Це може статися, якщо обліковий запис користувача стає жертвою вторгнення або якщо паролі недостатньо захищені. Щоб запобігти цій загрозі, рекомендується використовувати сильні паролі та включити двофакторну аутентифікацію.

– Фішинг та соціальна інженерія: - атаки фішингу можуть спонукати користувачів розкрити свої облікові дані. Користувачі повинні бути обережними щодо надісланих листів і посилань [15].

– Віруси та зловмисний код - завантажені файли можуть містити віруси або зловмисний код. Google Drive виявляє більшість зловмисних файлів, але ця загроза не повністю виключена.

– Неналежна обробка даних - несправедливі адміністратори або помилки в налаштуваннях можуть призвести до неналежної обробки даних. Важливо правильно налаштувати права доступу та контролювати адміністраторські дії [3].

– Втрата даних - випадкова видалення або небажані модифікації файлів можуть призвести до втрати даних. Регулярні резервні копії та контроль версій допомагають запобігти цій загрозі [15].

– Спільний доступ - при наданні доступу до файлів іншим користувачам, існує ризик, що дані можуть бути недоліково захищені або поширені несанкціоновано. Важливо встановлювати права доступу з урахуванням конфіденційності даних.

– Аналіз безпеки з боку виробника - Google Drive забезпечує власні заходи безпеки, але користувачі повинні бути готові до того, що виробник може мати доступ до даних для аналізу безпеки та виявлення порушень.

Збереження безпеки в Google Drive вимагає ретельного керування правами доступу, використання сильних паролів, регулярних аудитів безпеки та уважності користувачів під час обробки даних.

Загрози безпеки для Slack, популярної платформи для комунікації та спільної роботи, включають наступні аспекти:

– Несанкціонований доступ - зловмисники можуть намагатися отримати доступ до Slack-команд або каналів, щоб використовувати конфіденційну інформацію. Важливо використовувати міцні паролі та двофакторну аутентифікацію [16].

– Фішинг - атаки фішингу можуть використовувати соціальну інженерію, щоб спонукати користувачів надсилати свої облікові дані. Важливо навчати персонал розпізнавати фішингові атаки.

– Віруси та зловмисний код - файли, передані через Slack, можуть містити віруси або зловмисний код, який може шкодити комп'ютерам або мережам [16].

– Підробка повідомлень - зловмисники можуть підробити повідомлення або ідентифікаційні дані користувачів для здійснення атак.

– Втрата конфіденційності - важлива інформація може бути надто відкритою, якщо не встановлені відповідні обмеження доступу або якщо користувачі надто необачні у спілкуванні.

– Аналіз безпеки з боку виробника - Slack проводить аналіз з метою виявлення аномалій і загроз безпеці, але користувачі повинні розуміти, що певна кількість інформації може бути доступна адміністраторам [16].

– Спільний доступ - надання спільного доступу до повідомлень та файлів може призвести до незаконного поширення інформації.

Забезпечення безпеки в Slack вимагає налаштування прав доступу, навчання користувачів з питань безпеки, використання антивірусного програмного забезпечення та регулярного аудиту безпеки.

Загрози безпеки для Dropbox Business, хмарної служби для зберігання та обміну даними, включають наступні аспекти:

– Несанкціонований доступ - наявність облікового запису або надто спростого пароля може призвести до несанкціонованого доступу до зберіганих даних в Dropbox.

– Віруси та зловмисний код - файли, які завантажуються та зберігаються в Dropbox, можуть містити віруси або зловмисний код, які можуть шкодити комп'ютерам та мережам [17].

– Фішинг - фішингові атаки можуть спонукати користувачів надсилати свої облікові дані через спеціально сформовані повідомлення або посилання.

– Недостатня налаштувана безпека - неналежна настройка прав доступу та контролю може призвести до незаконного доступу до конфіденційної інформації [17].

– Спільний доступ до даних - надання спільного доступу до файлів та папок може призвести до незаконного поширення інформації [17].

– Неналежна обробка даних - помилки в налаштуваннях адміністраторів або несвідома обробка даних можуть призвести до незаконного розкриття інформації.

– Втрата даних - випадкова видалення або небажані модифікації файлів можуть призвести до втрати даних [4].

– Аналіз безпеки з боку виробника - Dropbox проводить аналіз безпеки для виявлення аномалій і загроз безпеки, але користувачі повинні розуміти, що певна кількість інформації може бути доступна адміністраторам.

Забезпечення безпеки в Dropbox вимагає правильної настройки прав доступу, використання сильних паролів, регулярних аудитів безпеки та навчання користувачів з питань безпеки.

Загрози безпеки для SharePoint, платформи для спільної роботи та обміну документами в межах організації, включають наступні аспекти:

– Несанкціонований доступ - несправедливий доступ до документів та ресурсів може виникнути через недоліки в налаштуваннях, облікові записи користувачів, або недоліки в безпеці облікових записів [18].

– Віруси та зловмисний код - документи, завантажені на платформу, можуть містити віруси або зловмисний код, які можуть поширитися в межах організації.

– Фішинг - атаки фішингу можуть спонукати користувачів надсилати облікові дані через спеціально сформовані повідомлення або посилання.

– Втрата конфіденційності - помилкові права доступу та обробка документів можуть призвести до незаконного розкриття конфіденційної інформації.

– Підробка документів - зловмисники можуть спробувати підробити документи або ідентифікаційні дані користувачів для використання в атаках.

– Аналіз безпеки з боку виробника - Microsoft, яка розробляє SharePoint, виконує аналіз безпеки для виявлення аномалій та загроз безпеці [18].

Забезпечення безпеки в SharePoint вимагає правильної настройки прав доступу, використання сильних паролів, регулярних аудитів безпеки та навчання користувачів з питань безпеки. Регулярні оновлення та моніторинг допомагають забезпечити найвищий рівень безпеки для вашої організації.

Загрози безпеки для Trello, популярної платформи для організації та спільної роботи над проектами, включають наступні аспекти:

– Несанкціонований доступ - несправедливий доступ до дошок та проектів може виникнути, якщо користувачі не налаштують належні правила доступу або використовуватимуть недоліки в паролях [19].

– Фішинг - фішингові атаки можуть спробувати використовувати соціальну інженерію, щоб спонукати користувачів надсилати свої облікові дані через підроблені повідомлення або посилання.

– Віруси та зловмисний код - файли, завантажені на дошки, можуть містити віруси або зловмисний код, які можуть шкодити комп'ютерам та мережам.

– Втрата конфіденційності - публічні або неправильно налаштовані дошки можуть призвести до витоку конфіденційної інформації [19].

– Підробка проектів та завдань - зловмисники можуть підробити проекти та завдання для внесення недобросовісної інформації або атак [19].

– Застосунки і розширення - деякі сторонні додатки та розширення можуть мати доступ до даних Trello, і це може створювати ризики для безпеки.

– Аналіз безпеки з боку виробника - Trello проводить аналіз безпеки для виявлення аномалій та загроз безпеці, але користувачі повинні розуміти, що певна кількість інформації може бути доступна адміністраторам [19].

Забезпечення безпеки в Trello вимагає належної конфігурації прав доступу, навчання користувачів з питань безпеки, регулярних аудитів безпеки та обачності щодо зовнішніх додатків і розширень. Також важливо встановлювати конфіденційність та контролювати доступ до проектів та завдань, особливо тих, що містять конфіденційну інформацію.

Аналіз загроз безпеки для різних інструментів обміну корпоративною інформацією демонструє серйозний характер викликів, які стоять перед організаціями у сфері збереження та обміну даними. Несанкціонований доступ, фішинг, віруси та незаконне розкриття інформації стають актуальними загрозами для безпеки. Однак застосування надійних практик безпеки, включаючи двофакторну аутентифікацію, аудити, налаштування прав доступу та навчання персоналу, може допомогти зменшити ризики. Для забезпечення найвищого рівня безпеки важливо постійно моніторити і адаптувати стратегії безпеки, враховуючи змінюючийся характер загроз і розвиток технологій.

1.3 Постановка задачі

Аналіз виконаний в попередніх підрозділах, показав що розробка нового хмарного середовища з можливостями перегляду коду, схожими на функціонал, який притаманний платформі GitHub, є вельми актуальним завданням в контексті спільної роботи та розробки програмного забезпечення. Завдання проекту полягає в створенні інтерактивної онлайн-платформи, де розробники можуть спільно працювати над кодом, аудитори можуть переглядати та коментувати код, а також долучатися до процесу рецензування. Серед основних завдань цього проекту включається розробка інтерфейсу для зручного відображення коду, можливість відстеження змін та версій коду, інтеграція з системами контролю версій, встановлення механізмів безпеки для

запобігання несанкціонованому доступу та забезпечення конфіденційності даних. Основними користувачами цієї платформи будуть розробники, команди програмістів, аудитори коду та проекти з відкритим вихідним кодом, які шукають зручний спосіб спільно працювати над програмними проектами та аналізувати код.

1.4 Висновки з розділу

Підсумовуючи розділ, присвячений хмарним платформам, їх безпеці та постановці завдання для розробки нового хмарного середовища з можливостями перегляду коду, варто відзначити, що в сучасних організаціях зберігання та обмін корпоративною інформацією стають все більш важливими завданнями. Ростуча кількість загроз, таких як несанкціонований доступ, фішинг та віруси, підкреслює важливість ретельного аналізу і забезпечення безпеки у хмарному середовищі. Хмарні платформи стають все більш популярними для спільної роботи над програмними проектами, і відповідно до цього зростає необхідність розвивати нові стратегії безпеки та інструменти для захисту даних. Поставлення завдання на створення нової хмарної платформи для спільної роботи над кодом визначається як ключовий крок у напрямку забезпечення безпеки та зручності для розробників та команд, які працюють над різними аспектами програмних проектів. Вирішення цієї задачі має на меті покращення процесів розробки та аудиту коду в хмарному середовищі, забезпечуючи високий рівень захисту від потенційних загроз і забезпечуючи ефективну співпрацю між учасниками команди.

2 МЕТОД ЗАХИЩЕННОГО ОБМІНУ КОРПОРАТИВНОЮ ІНФОРМАЦІЄЮ РОЗРОБНИКІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Математичний опис процесу обміну корпоративною інформацією

Математичний опис процесу обміну корпоративною інформацією відіграє важливу роль у сучасному бізнесі. Досягнення цілей корпорації та оптимізація бізнес-процесів вимагають вивчення обсягів даних, їхньої передачі та обробки. Цей розділ присвячений застосуванню математичних моделей, алгоритмів та методів для аналізу, збереження та забезпечення безпеки обміну корпоративною інформацією.

У математичному описі фігурують наступні ключові елементи: Входи, позначено як I; виходи, позначено як O; внутрішні стани, позначено як IS; внутрішні перетворення, позначено як IT. Для доведення використання цих ключових елементів та процесів в обміні корпоративною інформацію потрібно описати кожен із них та розглянути їх взаємозв’язок з точки зору теорії множини.

Зважаючи на детальні вхідні дані для обміну корпоративною інформацією з точки зору теорії множин, потрібно детальніше розглянути кожен з вказаних входів:

– Сирий код (RC) - це програмний код, який розробники створюють для розробки програм і послуг. Сирий код може бути написаний на різних мовах програмування і представляти різні складності і функціональності. Важливо забезпечити безпеку і цілісність сирого коду під час обміну ним між розробниками та командами.

– Проекти (P), які включають інформацію про розробку нових програм або послуг, включаючи вимоги, графіки, архітектуру, розклади і бюджети. Обмін проектами допомагає управляти розробкою та спільною роботою команд.

– Баг-репорти (BR) - це документи, які містять інформацію про виявлені помилки, проблеми та недоліки в програмному забезпеченні. Обмін баг-репортами допомагає виправити помилки і покращити якість продукту.

– Документи (D) -це корпоративна документація, яка включає в себе специфікації, технічні описи, інструкції, стандарти, політики та інші документи, необхідні для розробки, тестування та експлуатації продукту. Обмін документами важливий для розуміння та дотримання правил та стандартів.

– Звіти з код-рев'ю (CR) – це звіти, отримані у процесі взаємної перевірки програмного коду розробниками для забезпечення якості та безпеки. В процесі код-рев'ю обмінюються кодом, коментарями та відгуками. Це важливий етап для виявлення та виправлення помилок та вдосконалення коду.

Дані автентифікації користувачів (UAD) - інформація, необхідна для перевірки особистості користувачів, включає логін і пароль, біометричні дані, ключі або інші фактори автентифікації.

Розглянувши можливі входи, отримано таку множину:

$$\{RC, P, BR, D, CR, UAD\} = I, \tag{2.1}$$

де RC – сирий код, P - проекти, BR – баг-репорти, D - документи, CR – звіти з код-рев'ю, UAD – дані автентифікації користувачів.

Для кожного з цих входів важливо встановити механізми та процедури забезпечення конфіденційності, цілісності та доступності інформації під час обміну. Також необхідно враховувати внутрішні стани, які впливають на управління та захист корпоративною інформацією під час обміну.

Внутрішні стани в контексті обміну корпоративною інформацією важливі для ефективного управління і забезпечення безпеки інформації. Виконаний у попередньому підрозділі аналіз засобів дозволив визначити такі внутрішні стани:

– Правила розмежування прав доступу (ACR) - це набір правил і політик, які визначають, хто має право доступу до конкретних ресурсів і які дії вони можуть виконувати. Правила розмежування прав доступу регулюють, які користувачі або ролі мають доступ до певних даних і як вони можуть їх використовувати.

– Стан заповненості носіїв даних (DSU) - це інформація про те, наскільки заповнені носії даних, такі як сервери або сховища. Важливо контролювати заповненість для уникнення перевантаження, забезпечення долі інформації та вчасного розширення сховищ даних, якщо це необхідно.

– Еталонні значення факторів автентифікації (AFS) - це загешовані значення показників факторів автентифікації, необхідних для підтвердження особистості користувачів. Це може включати вимоги до паролів, біометричних даних, одноразових кодів тощо. Еталонні значення допомагають визначити, які фактори автентифікації прийнятні та безпечні для використання в організації.

– Журнали подій (EL) - це системи журналювання, які відстежують всі події і дії, пов'язані з обміном інформацією. Журнали подій дозволяють вести аудит інформації, виявляти несанкціонований доступ і інші безпекові порушення, а також забезпечують можливість відновлення даних у разі інцидентів.

– Конфігураційні параметри (CS) - це налаштування системи та програмного забезпечення, які визначають, як вони повинні працювати в контексті обміну інформацією. Ці параметри можуть включати параметри безпеки, мережеві налаштування, параметри шифрування і інше.

– Механізми резервного копіювання (BM) - це системи та процедури для регулярного створення резервних копій інформації та систем для відновлення даних в разі втрати або пошкодження. Резервне копіювання є важливою складовою внутрішніх станів для забезпечення надійності та доступності інформації.

Розглянувши можливі внутрішні стани, отримано таку множину:

$$\{ACR, DSU, AFS, EL, CS, BM\} = IS, \tag{2.2}$$

де ACR – правила розмежування прав доступу, DSU – стан заповненності носіїв даних, AFS – еталонні значення факторів автентифікації, EL – журнал подій, CS – конфігураційні параметри, BM – механізми резервного копіювання.

Ці внутрішні стани важливі для безпеки, моніторингу та ефективного управління обміном корпоративною інформацією. Вони допомагають забезпечити цілісність та доступність даних, виявляти безпекові загрози та реагувати на них, а також забезпечити дотримання політик та стандартів безпеки в організації.

Внутрішні перетворення у процесі обміну корпоративною інформацією включають різні операції та процедури, що застосовуються до вхідних даних для забезпечення безпеки та цілісності інформації під час її обміну. Розглянемо детальніше ці внутрішні перетворення:

– Шифрування (E), яку використовується для захисту конфіденційності даних під час їх передавання. Дані перетворюються в нечитабельний формат (шифрований текст), який може бути розшифрований лише за допомогою відповідного ключа. Шифрування дозволяє забезпечити конфіденційність даних, які передаються мережею.

– Автентифікація (A) використовується для перевірки особистості користувачів та систем. Це може включати використання паролів, біометричних даних або інших факторів автентифікації. Автентифікація дозволяє переконатися, що тільки правильні користувачі мають доступ до інформації та ресурсів.

– Підписи (S): Цінна інформація може бути підписана цифровим підписом для підтвердження її автентичності та цілісності. Цифровий підпис вказує, що дані не були змінені під час передачі та що вони були створені відомим джерелом.

– Гешування (H) використовується для створення фіксованої довжини геш-значень з вхідних даних. Геш-значення служать для перевірки цілісності даних, оскільки будь-яка зміна вхідних даних призведе до зміни геш-значення.

– Розшифрування (D) використовується для відновлення читабельного тексту зі шифрованого тексту. Це проводиться за допомогою відповідного ключа шифрування.

Розглянувши можливі внутрішні перетворення, отримано таку множину:

$$\{E, A, S, H, D\} = IT, \quad (2.3)$$

де E – шифрування, A – автентифікація, S – підписи, H – хешування, D - розшифрування.

Внутрішні перетворення важливі для забезпечення безпеки та цілісності інформації під час її обміну. Вони допомагають зменшити ризики, пов'язані з втратою даних, несанкціонованим доступом та іншими загрозами для безпеки.

Виходи являються собі ті ж самі Входи, але окрім даних автентифікації користувачів. Тобто, отримано таку множину:

$$\{RC, P, BR, D, CR\} = O, \quad (2.4)$$

де RC – сирий код, P - проекти, BR – баг-репорти, D - документи, CR – код- рев'ю.

Виходи в процесі обміну корпоративною інформацією відіграють критичну роль у забезпеченні ефективності та безпеки бізнес-процесів. Організації повинні ретельно планувати та контролювати цей аспект, щоб забезпечити надійний обмін інформацією та використання її відповідно до своїх потреб і цілей.

Розглянувши всі ключові аспекти, отримано такий математичний опис обміну корпоративною інформацією:

$$\textit{Обмін корпоративною інформацією} = I \times IS \rightarrow^{IT} O, \quad (2.5)$$

де I – входи, IS – внутрішні стани, IT – внутрішні перетворення, O – виходи.

Використовуючи математичний опис, є можливість систематично розглянути процес обміну корпоративною інформацією, від вхідних даних до виходів. Ця формула допомагає підкреслити важливість правильних внутрішніх станів та внутрішніх перетворень для забезпечення безпеки та цілісності даних під час їх обміну, що дозволяє формалізувати об’єкт досліджень.

2.2 Узагальнений опис методу

Метод обміну корпоративною інформацією - це складний процес передачі та обробки даних в організаційному контексті. Він включає в себе багато аспектів, які спільно забезпечують дієвий обмін та управління важливою інформацією в корпоративному середовищі. Тому перед розробкою методу важливо проаналізувати ці аспекти.

Однією з основних складових цього методу є комунікація. Внутрішня та зовнішня комунікація є ключовими для передачі інформації між різними стейкхолдерами. Вона може включати в себе різні види спілкування, такі як електронна пошта, телефонні дзвінки, відеоконференції, чати, зустрічі, а також використання спеціалізованих платформ для спілкування та обміну інформацією.

Забезпечення безпеки інформації - це інший важливий аспект. Корпоративна інформація може бути конфіденційною і чутливою, тому важливо застосовувати заходи для захисту її від несанкціонованого доступу та витоку [20]. Це включає в себе використання шифрування, автентифікації, встановлення прав доступу, а також моніторинг інцидентів для виявлення порушень безпеки [20].

Управління даними - це інший аспект методу обміну інформацією. Це включає в себе зберігання, організацію, індексацію та каталогізацію даних, щоб легко знаходити інформацію та забезпечувати її якість і цілісність. Крім того, це передбачає створення резервних копій для запобігання втратам даних в разі аварій або непередбачених ситуацій.

Інструменти для керування проектами і ресурсами також грають важливу роль у методі обміну корпоративною інформацією. Вони допомагають організувати робочі процеси, планувати завдання, розподіляти ресурси та відстежувати прогрес проектів. Це допомагає підвищити ефективність роботи і забезпечити вчасне виконання завдань.

Особливий акцент в методі обміну корпоративною інформацією також робиться на забезпеченні спільного доступу до даних та спільної роботи над проектами. Корпоративні портали, хмарні платформи та спеціалізовані інструменти для співпраці

можуть бути використані для об'єднання співробітників та спрощення спільної роботи над завданнями та проектами.

Розглянувши важливі ключові методу, можна його формалізувати як послідовність таких кроків:

Крок 1: Завантаження файлів і проектів:

– Користувач відкриває застосунок для обміну корпоративною інформацією на своєму персональному комп'ютері (ПК).

– Користувач обирає файли або проекти, які він бажає завантажити до застосунку.

Крок 2: Шифрування на ПК:

– Застосунок на ПК користувача виконує процес шифрування обраних файлів і проектів. Цей процес забезпечує безпеку інформації під час її передачі на сервер.

– Шифрування перетворює звичайні файли в незрозумілу форму, яку не можна легко прочитати без відповідного ключа.

Крок 3: Передавання на сервер:

– Зашифровані файли і проекти, тепер захищені від несанкціонованого доступу, передаються на сервер обміну корпоративною інформацією.

– Сервер приймає ці файли та зберігає їх в зашифрованому стані на безпечному зберіганні.

Крок 4: Зберігання на сервері:

– Файли і проекти залишаються на сервері, де вони знаходяться в зашифрованому вигляді, що гарантує конфіденційність та безпеку інформації.

– Сервер може забезпечувати резервне копіювання та інші засоби забезпечення цілісності даних.

Крок 5: Розшифрування:

– Користувач, який має необхідний доступ і права, входить до застосунку та обирає файли для розшифрування.

– Застосунок виконує процес розшифрування зашифрованих файлів з використанням відповідного ключа.

– Результатом розшифрування є отримання оригінальних файлів і проектів, які можна використовувати і редагувати на ПК користувача.

Таким чином, запропонований метод забезпечує безпеку та конфіденційність корпоративної інформації під час обміну нею через застосунок, дозволяючи контролювати доступ до даних та зберігати їх в захищеному стані. Це дозволить зменшити поверхню вразливостей, зокрема, нівелюючи загрози викрадання інформації з каналів передавання інформації.

2.3 Метод шифрування даних

Зважаючи на вимоги щодо забезпечення безпеки обміну корпоративною інформацією, метод гібридного шифрування стає ключовим елементом в рамках реалізації методу для застосунку обміну корпоративною інформацією. В даному підрозділі розглядається метод шифрування, де симетричний ключ генерується сервером та використовується для шифрування та розшифрування даних на комп'ютерах користувачів, а також де пара публічного та приватного ключа використовується для захисту самого симетричного ключа. Цей метод дозволяє забезпечити високий рівень безпеки та конфіденційності даних при обміні корпоративною інформацією. Метод шифрування передбачає низку кроків:

Крок 1. Генерування ключів сервером: Перший крок полягає в тому, щоб сервер згенерував симетричний ключ, який буде використовуватися для шифрування та розшифрування фактичних даних. Важливою є надійність цього кроку, оскільки симетричний ключ буде використовуватися для надійного шифрування інформації.

Крок 2. Зашифрування симетричного ключа публічним ключем користувача: Симетричний ключ, згенерований сервером, далі шифрується публічним ключем користувача. Це забезпечує безпеку передачі симетричного ключа через небезпечні канали, оскільки лише приватний ключ користувача може розшифрувати цей ключ. За допомогою асиметричного шифрування, наприклад, RSA, симетричний ключ

зашифровується так, щоб лише приватний ключ користувача міг розшифрувати його. Цей процес можна відобразити у формулі:

$$EncryptedKey = AsymmetricEncrypt\,(SymmetricKey, PublicKey), \quad (2.6)$$

де EncryptedKey – зашифрований симетричний ключ, AsymmetricEncrypt – функція шифрування, SymmetricKey – симетричний ключ, що був використаний для шифрування фактичних даних, PublicKey – публічний ключ одержувача.

Крок 3. Відправлення зашифрованого симетричного ключа та даних: Зашифрований симетричний ключ та фактичні дані відправляються на сервер. Це може відбуватися через захищені канали передачі даних, такі як HTTPS, для забезпечення додаткової безпеки.

Крок 4. Збереження симетричного ключа на сервері: Сервер зберігає зашифрований симетричний ключ у безпечному місці, де він залишається захищеним від несанкціонованого доступу.

Крок 5. Розшифрування симетричного ключа на комп'ютері користувача: Користувач використовує свій приватний ключ для розшифрування симетричного ключа на своєму комп'ютері. Цей крок гарантує, що лише власник приватного ключа може отримати доступ до симетричного ключа. Цей процес можна відобразити у формулі:

$$SymmetricKey = AsymmetricDecrypt(EncryptedKey, PrivateKey), \quad (2.7)$$

де EncryptedKey – зашифрований симетричний ключ, AsymmetricDecrypt – функція розшифрування, SymmetricKey – симетричний ключ, PrivateKey – приватний ключ одержувача.

Крок 6. Розшифрування та робота з даними: Одержавши симетричний ключ, користувач може розшифрувати фактичні дані і працювати з ними на своєму комп'ютері. Це забезпечує надійний обмін і конфіденційність даних.

Крок 7. Збереження ключів і захист від несанкціонованого доступу. Важливим є збереження і захист публічних та приватних ключів користувачів і сервера для забезпечення безпеки обміну даними.

Також потрібно визначити який саме симетричний шифр слід використовувати. Для цього було відібрано декілька шифрів: AES, Aria, Camellia і Nockcon [21]. Всі вони є симетричними та використовуються для шифрування файлів. Результатом порівняння цих шифрів є таблиця 2.1.

Таблиця 2.1 – Порівняння симетричних шифрів

	AES	Aria	Camellia	Nockcon
Безпека	Висока	Висока	Висока	Потенційно менш вивчена
Довжина ключа	128, 192, 256 біт	128, 192, 256 біт	128, 192, 256 біт	Залежить від реалізації
Ефективність	Висока	Висока	Висока	Залежить від реалізації
Підтримка	Широка (бібліотеки, апаратне обладнання)	Обмежена	Обмежена	Обмежена
Реалізація	Легка	Залежить від реалізації	Залежить від реалізації	Залежить від реалізації

Як видно з таблиці 2.1, AES виділяється серед інших симетричних шифрів завдяки високому рівню безпеки, широкій підтримці в бібліотеках та апаратному обладнанні, що робить його надійним вибором для різноманітних платформ. Завдяки стандартній довжині ключа в 128, 192 та 256 біт, AES пропонує гнучкість у виборі рівня безпеки в залежності від конкретних потреб. Його швидка та ефективна

реалізація дозволяє ефективно шифрувати великі обсяги даних. Загалом, ці характеристики роблять AES оптимальним вибором для шифрування файлів у багатьох сценаріях.

Такий метод гарантує безпеку обміну корпоративною інформацією, дозволяючи зберігати симетричний ключ на сервері і використовувати пару публічного та приватного ключа для захисту цього ключа під час передачі через небезпечні мережі.

2.4 Модель розмежування прав доступу

В цьому підрозділі буде дослідження та реалізовано дискреційну модель розмежування прав доступу в системі обміну корпоративною інформацією, використовуючи дискреційну модель Гаррісона-Руззо-Ульмана (HRU). Ця область зосереджена на створенні програмного забезпечення та алгоритмів, що регулюють, які користувачі мають доступ до певних ресурсів та функціональностей в інформаційних системах [22]. Розмежування прав доступу грає важливу роль у забезпеченні безпеки та конфіденційності даних, а також у забезпеченні оптимального використання ресурсів. Але так як у рамках обміну корпоративною інформацією, у деяких користувачів обов’язково мають бути певні права доступу, без можливості їх зміни, як наприклад у Tech Lead можливість читати звіти з код-рев'ю, вирішено розробити гібридну модель, яка буде тримати в собі переваги дискреційну, надаючи можливість власнику інформаційного ресурсу надавати права доступу за його бажанням, так і вже заготовленні ролі [5].

Для створення матриці доступу потрібно визначити права доступу, які кожен з суб'єктів має до кожного з об'єктів. В даному випадку (табл. 2.2) використано такі права доступу:

– R (читання) - дозволено читати інформацію з об'єкта.

– W (запис) - дозволено змінювати інформацію в об'єкті.

– X (виконання) - дозволено виконувати дії на об'єкті (наприклад, виконувати код).

– - (відмова в доступі) - немає доступу до об'єкта за замовчуванням.

– η (заборона доступу) - заборонено доступ до об'єкту.

Таблиця 2.2 – Матриця доступу

Об'єкти / Суб'єкти	Сирий код	Проекти	Баг-репорти	Документи	Звіти з код-рев'ю
Розробник	R	R	-	-	-
HR	-	-	-	R	-
Tech Lead	R	R	R	-	R
QA	-	-	RW	-	-
Не Команда	ηRWX	ηRWX	-	-	-

Як видно з таблиці 2.2, у цій матриці:

– Розробники мають право на читання (R) сирого коду та читання (R) проектів, відмова в доступі (-) до баг-репортів, документів та звітів з код-рев'ю.

– HR мають відмову в доступі (-) до сирого коду, проектів, баг-репортів та звітів з код рев'ю, право на читання (R) документів.

– Tech Lead має право на читання (R) сирого коду, проектів, баг-репортів та звітів з код рев'ю, відмову в доступі (-) до документів.

– QA мають відмову в доступі (-) до сирого коду, проектів, документів та звітів з код рев'ю, право на читання та запис (RW) баг-репортів.

– Не Команда має заборону в доступі на читання, запис і виконання (ηRWX) до сирого коду та проектів, відмову в доступі (-) до баг-репортів, документів та звітів з код рев'ю.

Ця матриця дозволяє чітко визначити, які права доступу мають різні суб'єкти до різних об'єктів в системі обміну корпоративною інформацією і є важливою для управління доступом та безпекою в організації.

2.5 Висновки з розділу

У даному розділі було розглянуто важливі аспекти пов'язані з обміном корпоративною інформацією, включаючи математичний опис процесу обміну, узагальнений опис методів, методи шифрування даних і модель розмежування прав доступу Математичний опис процесу обміну корпоративною інформацією вказує на важливі математичні аспекти, які лежать в основі процесу обміну інформацією в організації. Це може включати формалізовані алгоритми, рівняння та математичні моделі, що використовуються для забезпечення безпеки та ефективності обміну даними. Узагальнений опис методів вказує на різноманітні підходи і методи, які можуть бути використані для обміну корпоративною інформацією. Це може включати технології, алгоритми, принципи та стратегії, які допомагають забезпечити надійність і конфіденційність даних під час обміну. Метод шифрування даних є ключовим аспектом забезпечення безпеки корпоративного обміну інформацією. Шифрування дозволяє перетворити дані в нечитабельний вигляд, який може бути розшифрований лише з використанням правильного ключа. Це допомагає захистити дані від несанкціонованого доступу та забезпечити їх конфіденційність. Модель розмежування прав доступу визначає, які користувачі або системи мають доступ до конкретних ресурсів або інформації. Ця модель встановлює правила та обмеження, що регулюють, як і коли користувачі можуть отримати доступ до ресурсів, і допомагає управляти безпекою та контролем в організації. Узагальнюючи, цей розділ надає глибоке розуміння процесу обміну корпоративною інформацією, включаючи математичний та методологічний аспекти, які допомагають забезпечити безпеку, надійність і конфіденційність даних в організації.

3 РОЗРОБКА ЗАСОБУ

3.1 Узагальнена архітектура

Для реалізації серверної частини потрібно реалізувати механізм взаємодії всіх основних модулів між собою. Для цього вирішено використовувати за приклад архітектуру «EventDriven» [23]. Основою процесу є процедури «Накопичення даних шини подій» та «Прослуховування шини подій». Ці процедури реалізують можливість спілкування між сервісами серверної частини, що дозволяє оптимізувати роботи сервісів, шляхом спілкування за допомогою подій і повідомлень. На кожній ітерації роботи процедури «Накопичення даних шини подій», інформація про подію записується у логи для подальшого відслідковування можливих проблем. При отриманні запиту з клієнтської частини потрібна процедура «Обробка запиту», яка буде обробляти запит та готувати його для відправлення у шину подій. Для авторизації користувача потрібна процедура «Авторизація користувача», активація якої буде відбуватись з отриманням події процедурою «Прослуховування шини подій», її подальше передання в процедуру «Оброблення подій та надсилання повідомлень». Ця процедура потрібна для контролю навантаження сервісів, і буде працювати як черга по принципу FIFO (First In – First Out) [24]. Сама ж процедура «Авторизація користувача» повинна взаємодіяти з процедурою «Відправлення запитів в базу даних» для отримання даних про користувача, який створив запит. Також потрібні процедури «Взаємодії з файлами» та «Визначення прав доступу користувача» для відповідних їм отримання файлів проекту та визначення прав доступу користувача, щоб знати до яких файлів у користувача є доступ. Проаналізувавши потрібні алгоритми для роботи серверної частини, а також відповідні їм процедури, вирішено створити архітектуру серверної частини для подальшої розробки, архітектуру зображено на рисунку 3.1.

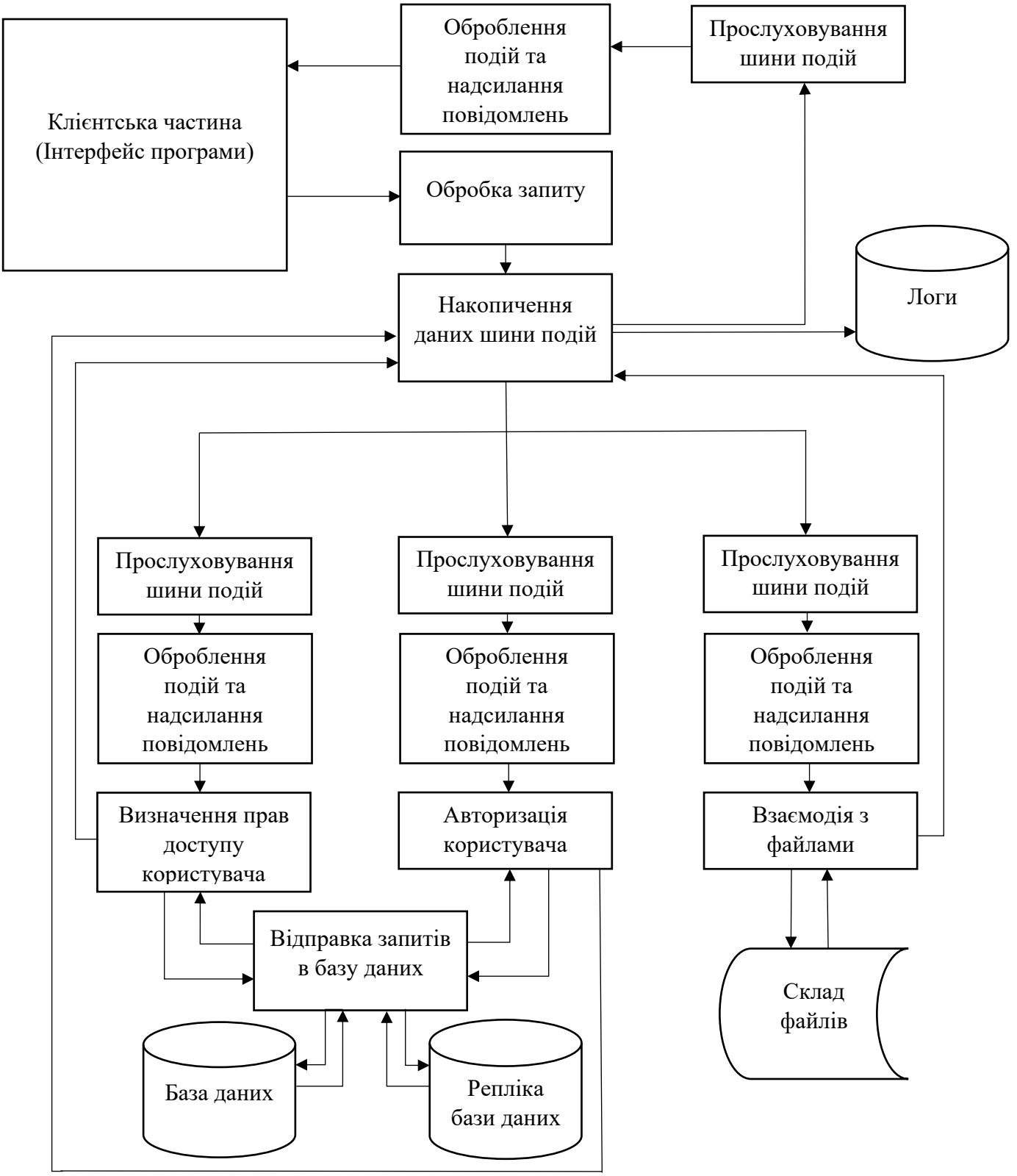


Рисунок 3.1 - Архітектура серверної частини

Як видно з рисунку 3.1, ключовою процедурою є накопичення даних шини подій, оскільки включає в себе зв’язки з усіма ключовими процедурами . Отримуючи інформацію з інших блоків, процедура накопичення шини подій відправляє подію у всі її слухачі, тобто у всі процедури «Прослуховування шини подій». Отримуючи подію яка призначалась для цієї течії роботи, вона активується та передає подію далі.

Для реалізації клієнтської частини потрібно реалізувати механізми роботи з серверною частиною, збереження частини даних на клієнтській частині та відображення інформації користувачеві. Для цього потрібно реалізувати процедури «Відправка події» та «Обробка відповіді», які і будуть здійснювати спілкування з серверною частиною. Процедури «Обробка відповіді» повинна поміщати дані у клієнтську бази даних для, наприклад, збереження файлів проекту, щоб уникнути великої кількості запитів у серверну частину. Також для відображення отримання інформації користувачеві потрібно реалізувати процедури «Оновлення інтерфейсу програми», яка, на основі даних у базі даних, буде відображати їх користувачеві. Для обробки дій користувача потрібно реалізувати процедури «Обробка дій користувача», яка, в залежності від дій користувача, буде робити запити у серверну частину, або відображення вже отриманні раніше дані. Проаналізувавши потрібні алгоритми для роботи клієнтської частини, а також відповідні їм процедури, вирішено створити архітектуру клієнтської частини для подальшої розробки, архітектуру зображено на рисунку 3.2.

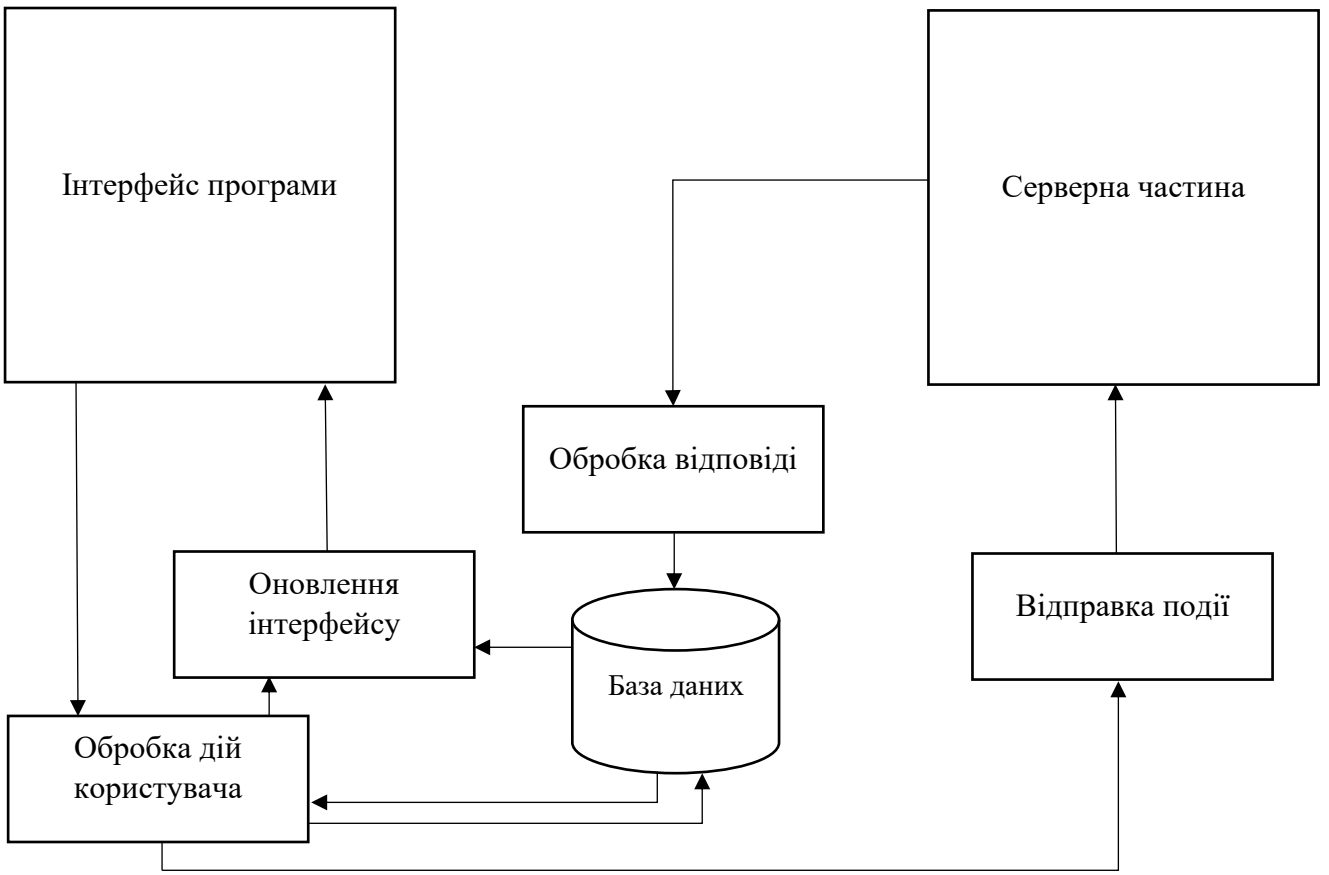


Рисунок 3.2 – Архітектура клієнтської частини

Як видно з рисунку 3.2, ключовими процедурами є оновлення інтерфейсу програми та обробка дій користувача, які виконують всю взаємодію з користувачем, відправляючи запиту у серверну частину або базу даних та оновлюючи інтерфейс користувача новою інформацією.

У результаті реалізації архітектури серверної та клієнтської частини засобу обміну корпоративною інформацією вдалося створити добре структуровану та ефективну систему. Серверна частина забезпечує надійне зберігання та обробку даних, здатну витримувати великі обсяги інформації, тоді як клієнтська частина надає зручний інтерфейс для користувачів, сприяючи зручному обміну даними в корпоративному середовищі. Ця архітектура допомагає підвищити продуктивність та ефективність обміну інформацією в організації, забезпечуючи надійність та безпеку даних.

3.2 Алгоритм шифрування

Основним механізмом роботи є алгоритм шифрування, який буде відповідати за безпечну передачу файлів від клієнтів на сервер, та навпаки. Зашифрування та розшифрування повинні відбуватись на клієнтській частині для забезпечення більшої безпеки при передачі даних (рис. 3.3). Для симетричного шифрування обрано AES [25].

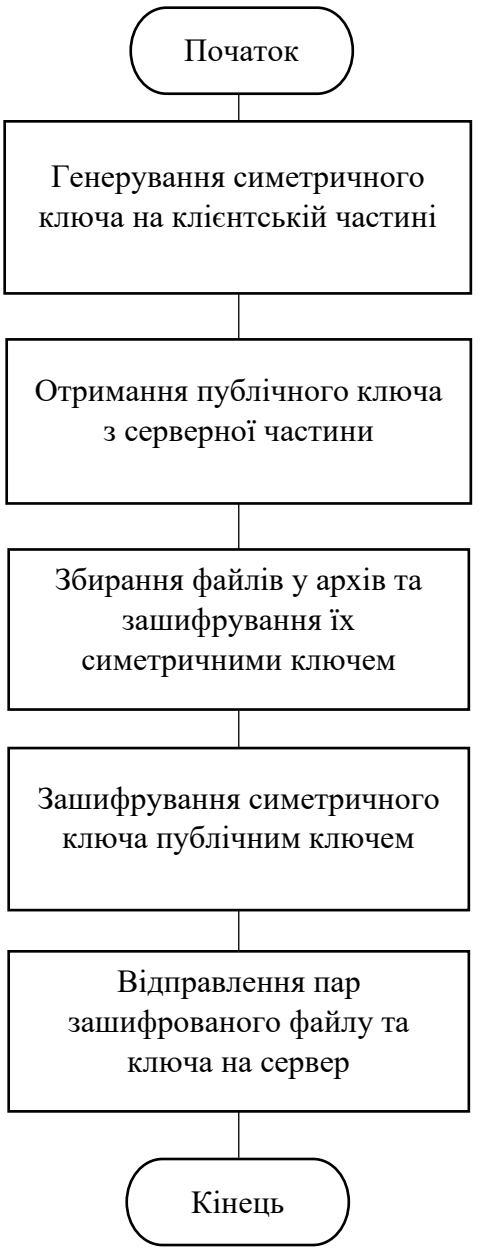


Рисунок 3.3 – Алгоритм зашифрування

Як видно з схеми, все шифрування відбувається на клієнтській частині. Сервер виконує роль тільки зберігача інформації та файлів. Для розшифрування алгоритм повинен включати в себе отримання даних з серверу та їх розшифрування (рис. 3.4).

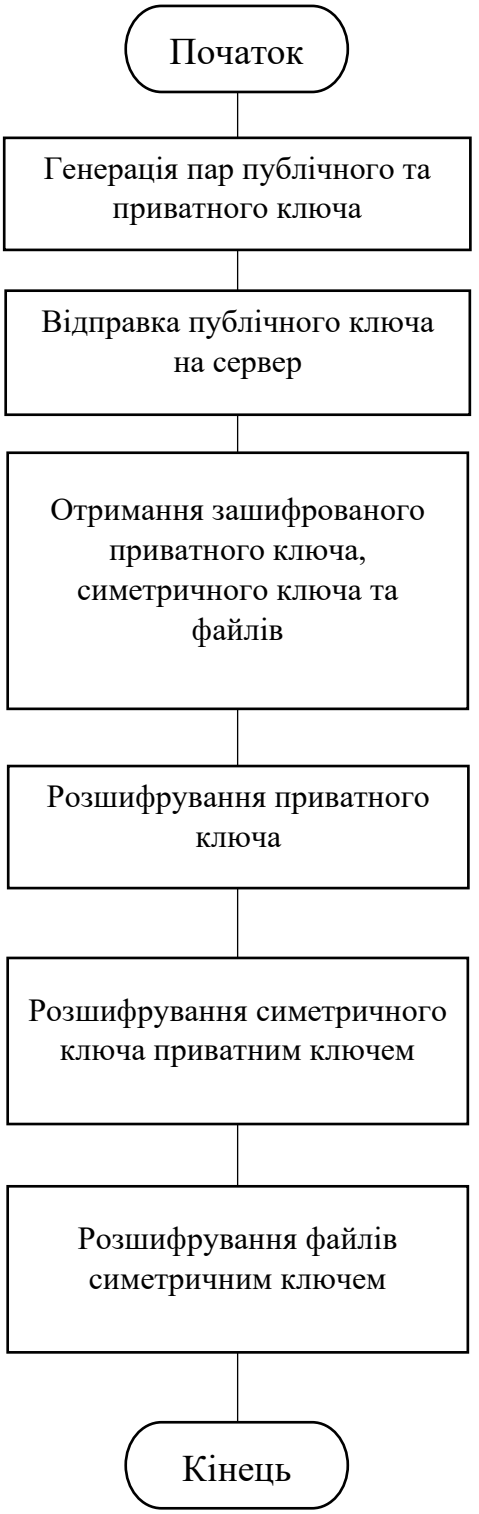


Рисунок 3.4 - Алгоритм розшифрування

Впровадження алгоритмів шифрування стало ключовим компонентом безпеки в системі. Реалізовані алгоритми гарантують конфіденційність та цілісність корпоративної інформації, забезпечуючи високий рівень захисту від несанкціонованого доступу. Ці заходи забезпечують відповідність стандартам безпеки та сприяють довірі користувачів до системи обміну даними, роблячи її надійною та відповідальною в роботі з конфіденційною інформацією.

3.3 Алгоритм реєстрації та автентифікації користувачів

Алгоритм реєстрації/автентифікації користувача включає в себе роботу модулю автентифікації користувачів. Спочатку користувач вибирає потрібний йому режим роботи (реєстрація, авторизація), потім перевіряється вибраний режим роботи і від цього залежить яка із процедур (реєстрації, чи автентифікації) далі буде працювати. Загальний алгоритм реєстрації/автентифікації зображено на рисунку 3.5

Рисунок 3.5 - Алгоритм реєстрації/автентифікації користувачів

При авторизації (рис. 3.6): користувач надає логін та свій симетричний ключ, який перевіряється і передається у модуль автентифікації користувачів, звідки очікується відповідь з проектами та файлами користувача, яка також буде підтверджуватись за допомогою перевірки належності окремих змінних до відповідних їх дійсним типам даних.

Рисунок 3.6 - Процедура автентифікації користувача

Процес реєстрації аналогічний авторизації (рис. 3.7), але з отриманням мнемонічного коду, який потрібен для відновлення доступу у разі втрати симетричного ключа.

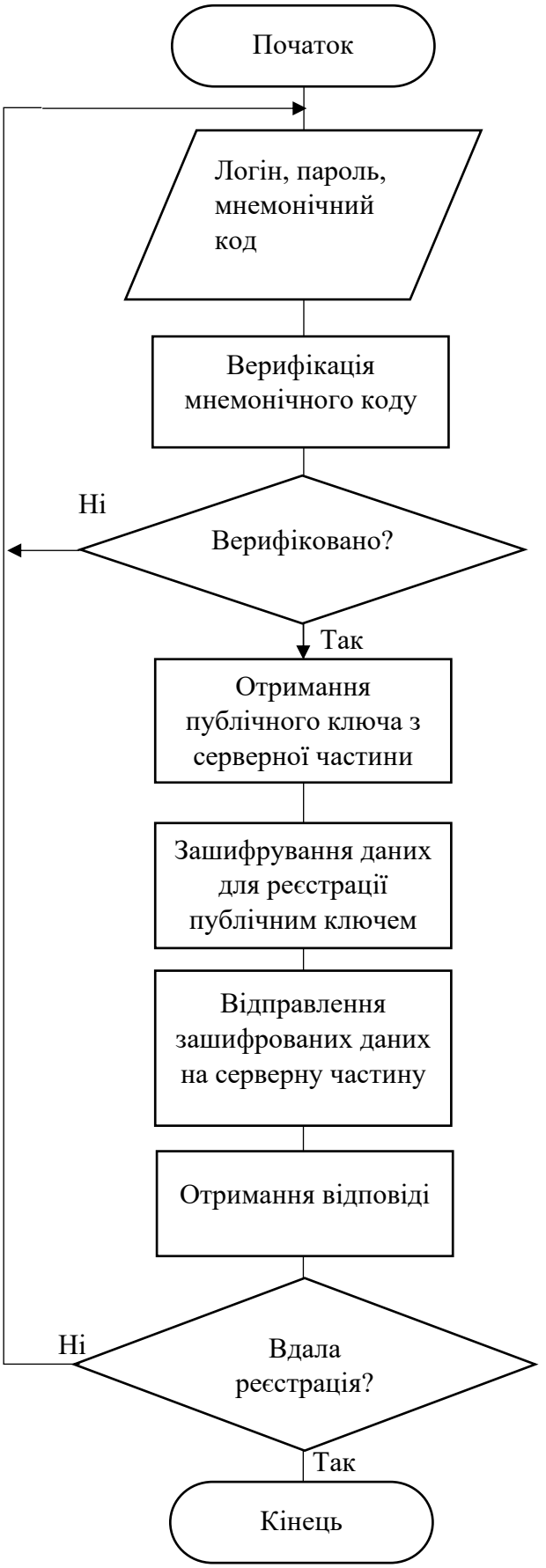


Рисунок 3.7 - Схеми процедури реєстрації користувача

Також потрібно описати процедуру валідації логіну та симетричного ключа (рис 3.8), яка буде включати в себе зашифрування логіну та ключа та відправка їх на сервер для подальшої валідації.

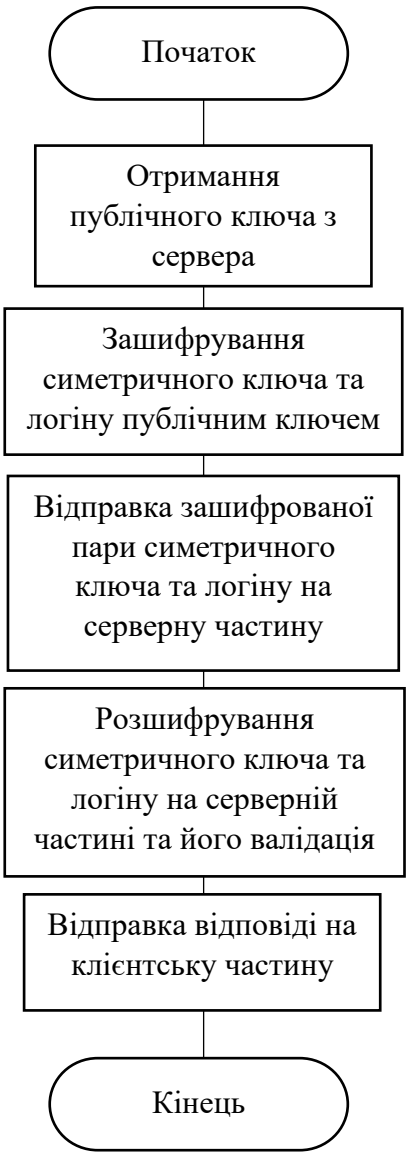


Рисунок 3.8 - Схеми процедури валідації логіну та симетричного ключа

Оскільки процедура автентифікації включає в себе отримання файлів користувача після успішної автентифікації, то така процедура має включати в себе відправлення запиту на їх отримання та розшифрування таких файлів на клієнтській частині (рис. 3.9).

Рисунок 3.9 - Схема процедури отримання файлів

Реалізація алгоритмів реєстрації та автентифікації користувачів створила систему з високим рівнем безпеки та зручності використання. Процедура реєстрації впроваджує ефективні методи валідації логінів та симетричних ключів, спрощуючи приєднання користувачів. Механізм автентифікації гарантує високий рівень безпеки через перевірку ідентичності користувача. Отримання файлів з серверу забезпечує швидкий та безпечний обмін корпоративною інформацією.

3.4 Алгоритм розмежування прав доступу

Системи, які оброблюють чи зберігають конфіденційну інформацію, повинні ефективно керувати доступом користувачів до різних ресурсів та функцій. Одним із ключових етапів забезпечення безпеки є розмежування прав доступу, що визначає, які конкретні дії або ресурси може використовувати кожен користувач. На шляху вдосконалення цього процесу виникає необхідність у чіткому алгоритмі, який забезпечить ефективне управління правами доступу. Нижче (рис. 3.10) наведена схема алгоритму розмежування прав доступу, що дозволяє системі визначати, які дії та ресурси доступні конкретному користувачеві, забезпечуючи при цьому високий рівень безпеки та ефективності.

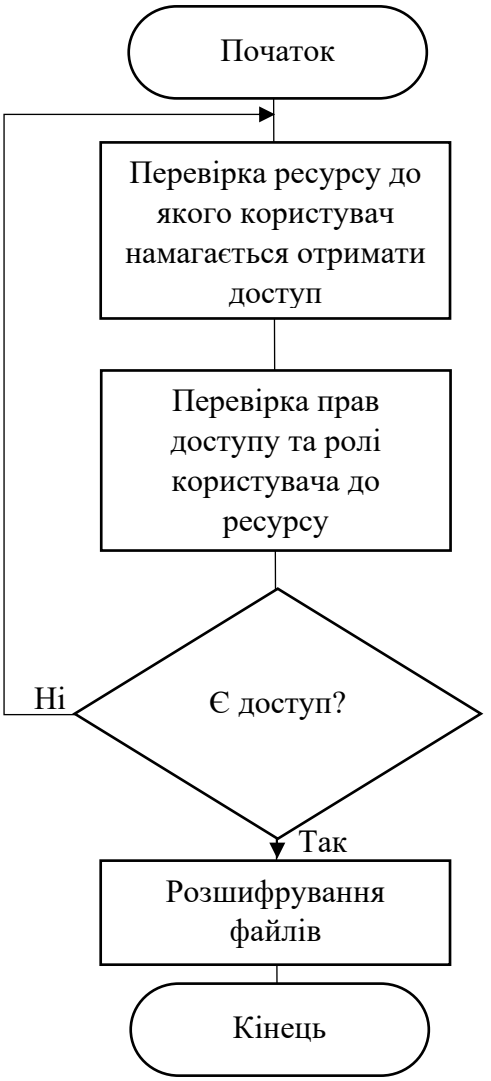


Рисунок 3.10 – Алгоритм розмежування прав доступу

Процес надання прав вимагає систематичного та безпечного підходу, оскільки неправильні або неконтрольовані зміни можуть призвести до порушення безпеки та конфіденційності даних. У даному контексті розроблено схему алгоритму надання доступу користувачеві іншим користувачем (рис. 3.11). Цей алгоритм дозволяє системі ефективно визначати нові рівні доступу, враховуючи права та запобігаючи можливим конфліктам. Наведена схема служить основою для розробки прозорого та безпечного механізму управління доступом.

Рисунок 3.11 – Алгоритм надання прав доступу

Алгоритм розмежування прав доступу та надання доступу користувачам виявився ефективним, створюючи систему, яка добре керує правами користувачів. Його застосування дозволяє точно визначити рівень доступу для кожного користувача, гарантуючи адаптовану та безпечну роботу з корпоративною інформацією. Цей підхід забезпечує гнучкість та конфіденційність у керуванні доступом, що робить систему відмінним інструментом для користувачів у взаємодії з даними.

3.5 Висновки з розділу

В процесі реалізації алгоритмів розмежування прав доступу була впроваджена гнучка система, що дозволяє точно визначити рівень доступу для кожного користувача в залежності від його функціональних потреб та ролі в організації. Це забезпечує ефективне керування конфіденційністю та безпекою корпоративних даних. Алгоритми шифрування виграють ключову роль у забезпеченні безпеки обміну інформацією. Застосовані шифри гарантують, що дані залишаються конфіденційними та невразливими перед потенційними загрозами безпеки. Це створює надійну основу для безпечного обміну даними в корпоративному середовищі. Процедури реєстрації та автентифікації забезпечують не лише легкість введення нових користувачів у систему, але й високий рівень захисту облікових записів. Використані алгоритми дозволяють перевіряти ідентичність користувачів та уникнути несанкціонованого доступу. Ці алгоритми разом створюють інтегровану та надійну інфраструктуру для обміну корпоративною інформацією, де ефективне керування правами доступу, шифруванням та процедурами автентифікації грають важливу роль у забезпеченні безпеки та ефективності обробки даних.

4 ТЕСТУВАННЯ ЗАСОБУ

4.1 Обґрунтування вибору інструментів розробки та тестування

Основним вибором для розробки є вибір мови програмування. Для цього необхідно проаналізувати сучасні мови програмування.

JavaScript - це високорівнева, інтерпретована мова програмування, яка використовується для створення динамічних веб-сайтів [26]. Вона є однією з основних технологій для реалізації клієнтської логіки у веб-додатках, додаючи можливість взаємодії та динаміки на стороні користувача. JavaScript виконується в середовищі веб-переглядача (або на інших платформах, таких як Node.js для серверної розробки).

До основних переваг JavaScript можна віднести:

– Широке поширення - JavaScript є однією з найпоширеніших мов програмування в інтернет-розробці. Майже кожен сучасний веб-переглядач підтримує JavaScript, що робить його доступним для користувачів у будь-якому веб-середовищі.

– Асинхронна природа - JavaScript має асинхронну модель виконання, що дозволяє ефективно обробляти операції вводу-виводу без блокування інших операцій. Це важливо для розробки веб-застосунків, де часто виникають асинхронні запити, такі як отримання даних з сервера [26].

– Багатофункціональність на клієнті - використання JavaScript на клієнті дозволяє створювати багатофункціональні та інтерактивні інтерфейси для користувачів, що є важливим для корпоративних застосунків з багатьма функціональними вимогами.

– Спільна логіка - використання однієї мови (JavaScript) для клієнтської та серверної частин дозволяє спільно використовувати деяку логіку між клієнтом та сервером. Наприклад, обрання однієї мови дозволяє переосмислити валідацію даних або форматування на обох сторонах.

– Екосистема та бібліотеки - широка екосистема JavaScript має багато готових бібліотек та фреймворків, які полегшують розробку та підтримку застосунків.

Загалом, використання JavaScript у клієнт-серверній архітектурі дозволяє створювати потужні та ефективні веб-застосунки, що можуть ефективно обмінювати корпоративною інформацією та забезпечувати високий рівень інтерактивності для користувачів.

Порівняння JavaScript з C++:

– C++ компілюється заздалегідь [27], JavaScript інтерпретуються, тобто, додатки написані на JavaScript працюють повільніше, але підтримують кроссплатформеність.

– C++ є статично типізованою, JavaScript – динамічно типізованою.

– C++ зазвичай використовується для програм, які потребують високої продуктивності, наприклад операційні системи [27].

– C++ дозволяє взяти контроль на потоками, тоді як JavaScript дозволяє виконувати декілька завдань [27], розбиваючі завдання на асинхронні функції, які викликаються, коли дані готові, що є важливими оскільки у роботі потрібно очікувати дані від інших модулів, не продовжуючи виконання додатку далі.

Порівняння JavaScript з Java:

– Java є сильно типізованою мовою, і змінні повинні бути оголошені спочатку для використання в програмі. У Java тип змінної перевіряється під час компіляції [28], натомість JavaScript є динамічно типізованою.

– Java - це об'єктно-орієнтована мова програмування, а JavaScript - це мова сценаріїв на основі об'єктів, що, враховуючи, спроектовані у другому розділі моделі ігрових процесів та робота гри з технологіями блокчейну, є перевагою мови JavaScript.

– Об’єкти Java засновані на класах [28], а об'єкти JavaScript [26] засновані на прототипах.

– Java є окремою мовою, а JavaScript міститься на веб-сторінці та інтегрується з її вмістом HTML

Вибір React та Nest.js для розробки засобу обміну корпоративною інформацією може бути обґрунтований кількома вагомими причинами:

Ефективна реактивність та компонентний підхід (React):

– Компонентна архітектура - React базується на компонентній архітектурі, що дозволяє розбити інтерфейс на невеликі, самодостатні компоненти. Це полегшує розробку, підтримку та розширення корпоративного інтерфейсу [29].

– Реактивна підтримка: React дозволяє ефективно взаємодіяти з користувачем, оновлюючи інтерфейс лише при необхідності. Це може забезпечити більш зручне та продуктивне використання корпоративного засобу обміну інформацією.

Масштабованість та серверна частина (Nest.js):

– Node.js підтримка: Nest.js базується на Node.js, що забезпечує високу швидкість та ефективність роботи серверної частини. Це особливо важливо при обробці великого обсягу корпоративної інформації.

– Масштабованість: Nest.js надає вам можливість побудувати масштабовані застосунки, що дозволяє легко розширювати ваші системи для вирішення зростаючих потреб корпоративного середовища [30].

– Типізація та модульність: Nest.js використовує TypeScript, що сприяє високій читабельності коду та допомагає у виявленні помилок на етапі розробки. Модульна структура також полегшує підтримку та розширення серверної частини.

Можливості розширення та спільна робота:

– Екосистема та спільнота: React та Nest.js мають великі та активні спільноти розробників, що означає, що ви матимете доступ до широкого спектру бібліотек, модулів та ресурсів, які полегшать вам розробку та підтримку [30].

– API для зв'язку з іншими системами: Nest.js, як серверний фреймворк, може легко інтегруватися з іншими корпоративними системами через стандартні API та протоколи обміну даними.

Враховуючи всі вищенаведені аргументи, можна висновувати що мова програмування JavaScript, бібліотека React та фреймворк Nest.js є найбільш доцільними до використання для реалізації поставленого завдання.

4.2 Блокове тестування

Розвиток програмного забезпечення вимагає високої якості коду та надійності розроблених програмних продуктів. Одним із ефективних методів забезпечення цих вимог є використання тестування на рівні окремих компонентів програмного забезпечення, відомого як unit-тестування. Даний підрозділ присвячений проведенню unit-тестування розробленого застосунку з метою перевірки коректності та надійності його функціоналу.

Оскільки застосунок розроблений на основі клієнт-серверної архітектури, доцільно виконувати тестування на кожній із сторін по черзі. Для клієнтської частини було вирішено взяти основні функції, які відповідають за працездатність застосунку, такі як: GetKeys (метод для отримання ключів з серверної частини), Login/Register (методи реєстрації та аутентифікації користувача), UploadFiles/GetFiles (методи отримання та вивантаження файлів).

У межах блокового тестування методу GetKeys відбувається перевірка того, що він повинен повертати ключі та зберігати їх у локальному сховищі, результати тестування зображено на рисунку 4.1.

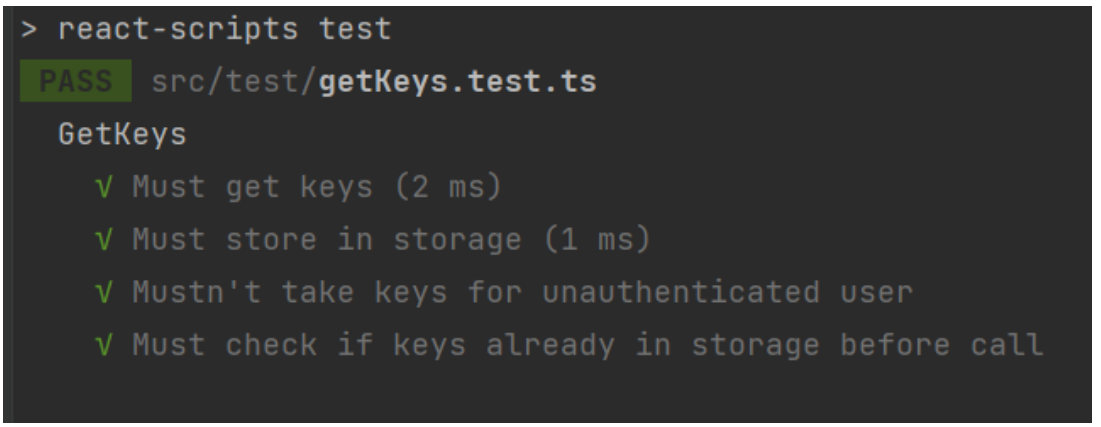


Рисунок 4.1 – Результат тестування методу GetKeys

Судячи з результатів тестування, метод працює правильно. У межах блокового тестування методів Login/Register відбувається перевірка того, що вони повинні: реєструвати користувача; видавати помилку якщо при реєстрації вказані невірний/вже

зареєстрований логін; перевіряти валідність мнемонічного виразу; проводити авторизацію вже зареєстрованим користувачам; видавати помилку, якщо дані некоректні. Результати цього тестування зображено на рисунку 4.2.

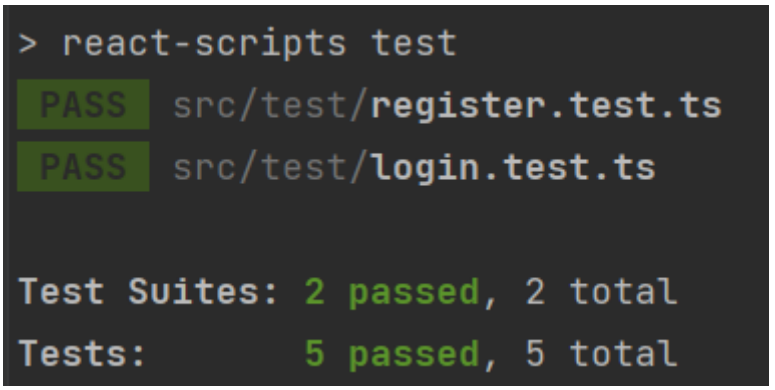


Рисунок 4.2 – Результат тестування методів Login/Register

Судячи з результатів тестування, методи працюють правильно. У межах блокового тестування методів UploadFiles/GetFiles відбувається перевірка того, що методи повинні: вивантажували файли; видавати помилку, якщо вивантажити файли намагається не автентифікований користувач; видавати помилку, якщо завантажити файли намагається не автентифікований користувач; завантажувати файли. Результати тестування наведено на рисунку 4.3.

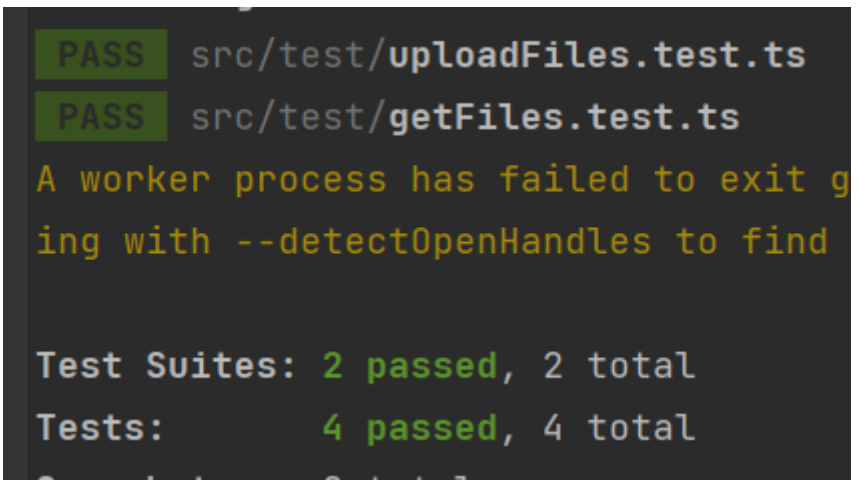


Рисунок 4.3 – Результат тестування методу GetKeys

Судячи з результатів тестування, методи працюють правильно. Для серверної частини потрібно протестувати такі ж самі методи, тобто їх серверну імплементацію. На рисунку 4.4 зображено результат тестування цих методів.

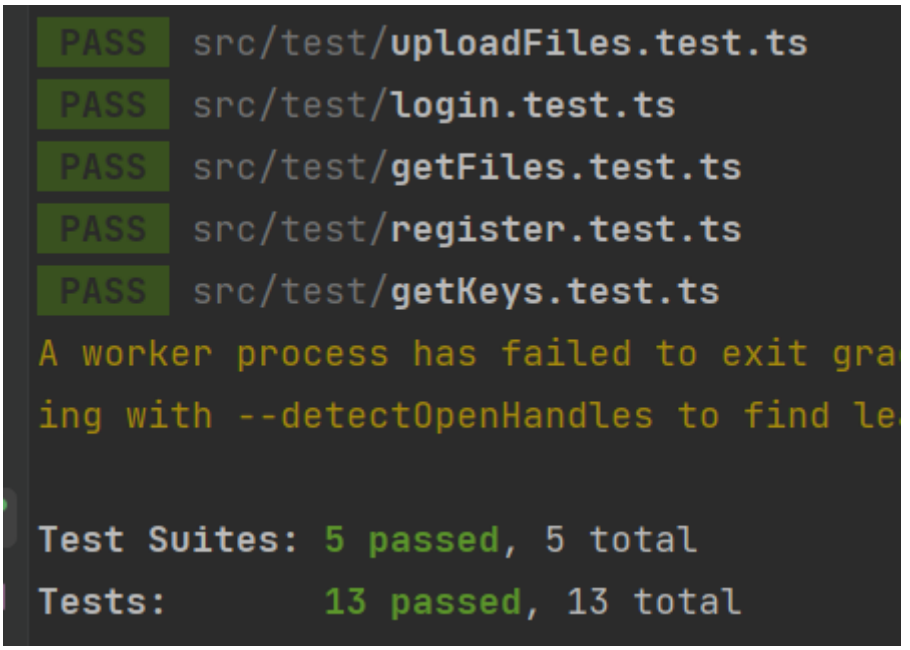


Рисунок 4.4 – Результат тестування методів GetKeys, Login, Register, UploadFiles, GetFiles на серверній частині

Судячи з результатів тестування, методи працюють правильно, а це значить що методи працюють правильно на клієнтській та серверній частинах.

Проведене блокове тестування підтвердило високу ефективність та надійність методів GetKeys, Login, Register, UploadFiles, GetFiles. Система готова до роботи в реальних умовах, відповідаючи всім функціональним вимогам.

4.3 Інтеграційне тестування

Для досягнення коректності роботи програмного забезпечення є важливо перевірити не лише функціональність окремих компонентів, а й їх взаємодію та інтеграцію. Інтеграція є ключовим етапом в життєвому циклі програмного продукту, адже вона спрямована на впевненість у правильності та ефективності взаємодії різних

частин системи. У даному підрозділі магістерської роботи висвітлюється процес та результати інтеграційного тестування розробленого застосунку.

Оскільки у минулому підрозділі проводилось тестування окремих функцій, без їх прямої взаємодії на клієнтській та серверних частинах, доцільно тепер виконати тестування їх взаємодії у реальному часі. Для перевірки будуть використовуватись ті ж самі умови, які було зображено у минулому розділі (див. рис. 4.1-4.4). Результати інтеграційного тестування зображено на рисунку 4.5.

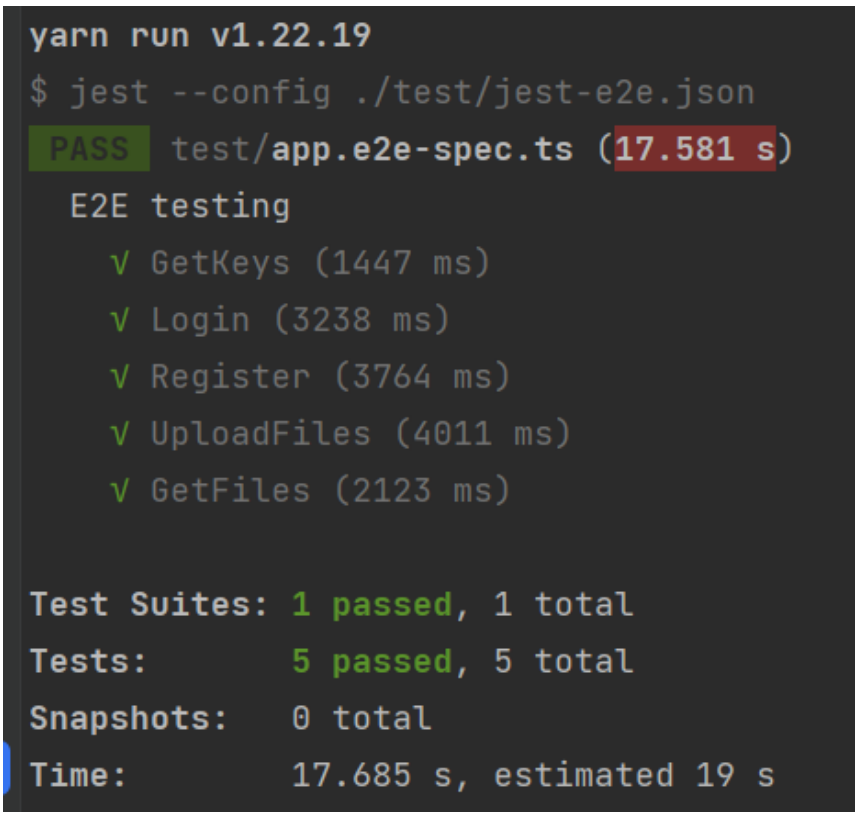


Рисунок 4.5 – Результат інтеграційного тестування

Судячи з результатів тестування, взаємодія методів на клієнтській та серверних частинах працює правильно.

Протестувавши взаємодію між методами, які реалізовані на основі алгоритмів описаних у третьому розділі, перевіривши їх на коректність значень, що повертається, можна вважати що інтеграція розроблених методів пройшло вдало, оскільки усі тести було пройдено успішно.

4.4 Статичне тестування безпеки

Однією з ефективних стратегій гарантування безпеки програмного продукту є використання статичного тестування безпеки.

У межах тестування безпеки, було вирішено обрати три інструмента для проведення цього тестування, а саме: Внутрішній метод пакет менеджеру, Snyk та SonarQube [31]. Вирішено почати з внутрішнього методу пакет менеджеру, а саме методу audit у пакет менеджеру yarn. Результат тестування зображено на рисунку 4.6.

```
PS C:\Users\alexe\Desktop\graynode\graynode-backend> yarn audit --groups dependencies
yarn audit v1.22.19
0 vulnerabilities found - Packages audited: 248
Done in 0.79s.
```

Рисунок 4.6 – Результат статичного тестування безпеки за допомогою «yarn audit»

Судячи з результатів тестування за допомогою «yarn audit» проблем у застосунку немає. Наступне тестування відбувалось за допомогою платформи Snyk, яка пропонує проводити тестування на їх серверах. Результат тестування зображено на рисунку 4.7.

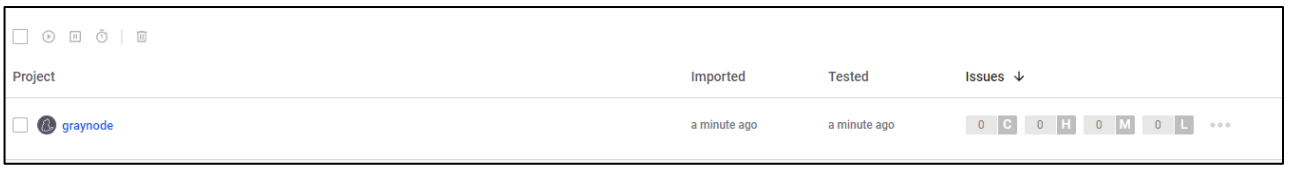


Рисунок 4.6 – Результат статичного тестування безпеки за допомогою Snyk

Судячи з результатів тестування за допомогою платформи Snyk проблем у застосунку немає. Наступне тестування відбувалось за допомогою інструменту SonarQube, який інтегрується у GitLab і виконує тестування коли з’являються нові комміти в запиті на злиття (Merge Request). Результат тестування зображено на рисунку 4.7.

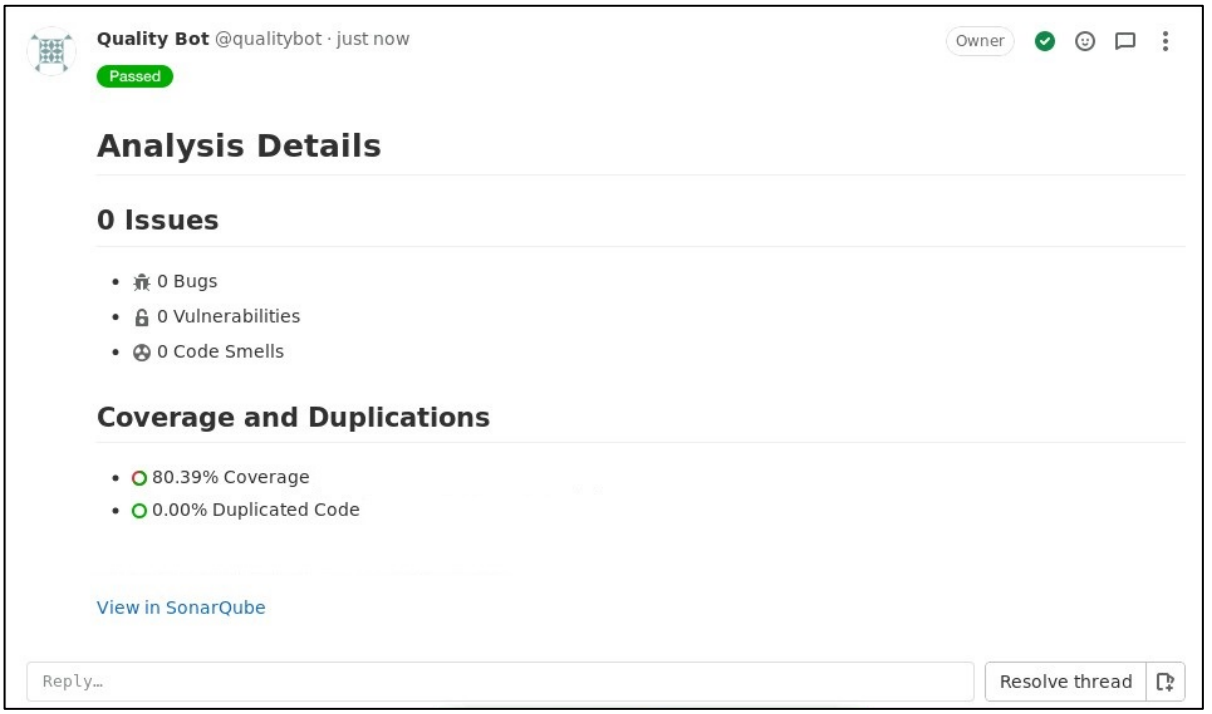


Рисунок 4.7 – Результат статичного тестування безпеки за допомогою SonarQube

Судячи з результатів тестування за допомогою SonarQube проблем у застосунку не виявлено.

Наведені результати тестування дозволяють стверджувати про успішно проведене статичне тестування безпеки, завдяки якому було встановлено високий рівень захищеності системи. Те, що не було виявлено жодних загроз чи вразливостей, що свідчить про коректне застосування заходів безпеки.

4.5 Тестування всього застосунку

Після завершення інтеграції та тестувань безпеки, доцільно виконати тестування всієї системи. У межах тестування застосунку, відбувається перевірка алгоритмів авторизації при правильно введених даних користувача. Для тестування було створено тестового користувача. Результат авторизації зображено на рисунку 4.8.

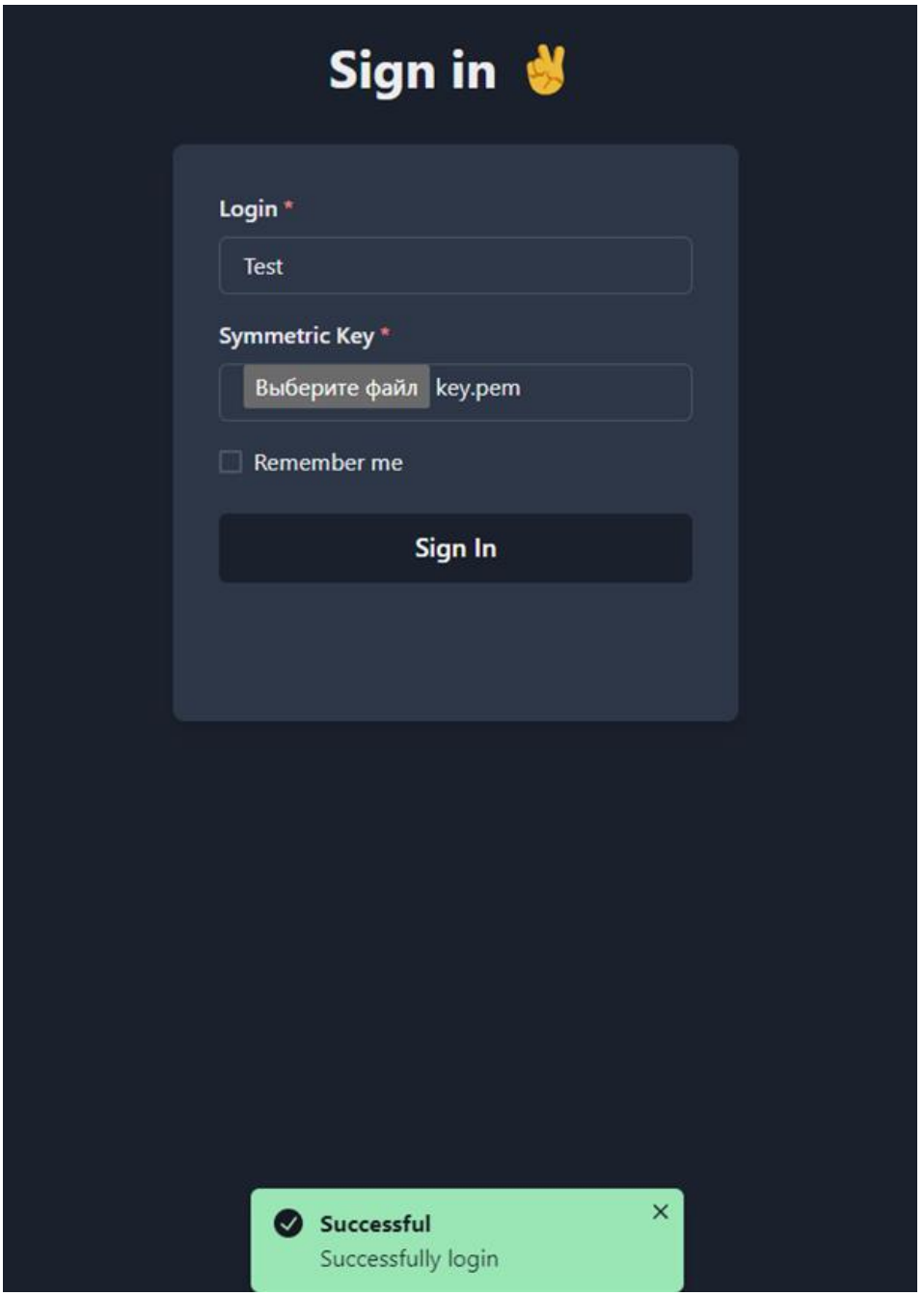


Рисунок 4.8 – Результат тестування алгоритму авторизації

Наступне тестування, яке необхідно провести, це тестування можливість вивантаження файлів для авторизованого користувача. Результати тестування зображено на рисунку 4.9.

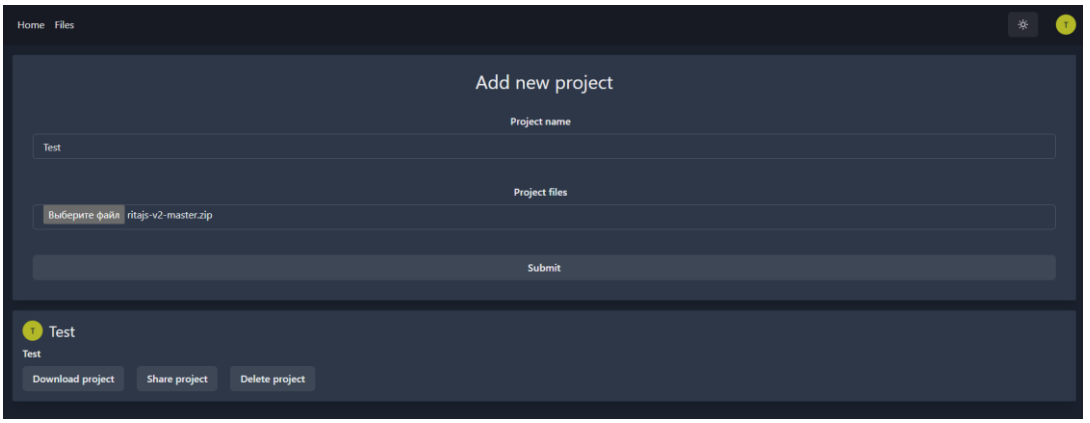


Рисунок 4.9 – Результат тестування можливості завантаження файлів

Як видно з результатів, наведених на рисунку 4.9, тестовий користувач зміг завантажити файл, і він одразу з’явився у списку проектів користувача, що свідчи про те що завантаження працює правильно. Тепер потрібно перевірити роботу цього алгоритму для неавторизованого користувача. Результат тестування зображено на рисунку 4.10.

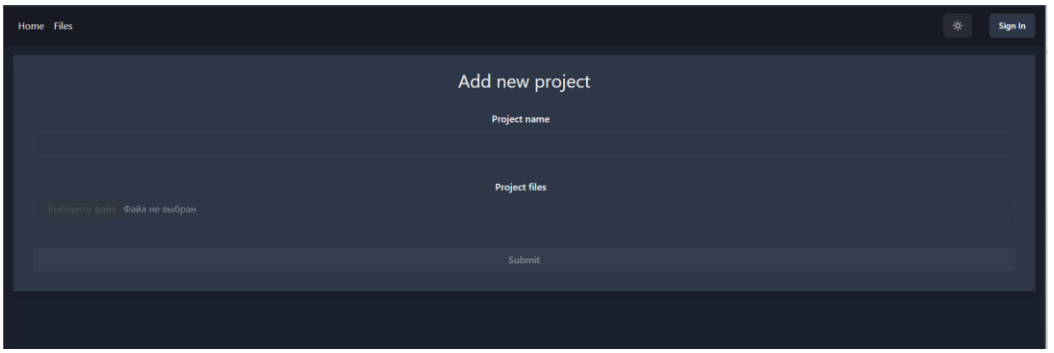


Рисунок 4.10 – Результат тестування можливості завантаження файлів для неавторизованого користувача

Як видно з результатів (рис. 4.10), неавторизований користувач немає змоги завантажити проект, такий ж захист запроваджено на серверній частині, де для завантаження потрібен спеціальний токен, який клієнтська частина отримує при авторизації. Наступне тестування це тестування вивантаження проекту власник інформаційного ресурсу, або іншими користувачами, яким він надав доступ. Результати тестування зображено на рисунках 4.11-4.13.

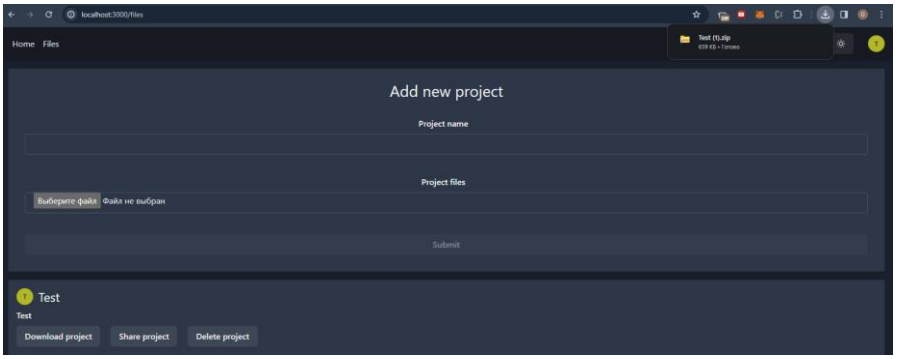


Рисунок 4.11 – Результат тестування можливості завантаження файлів для власника інформаційного ресурсу

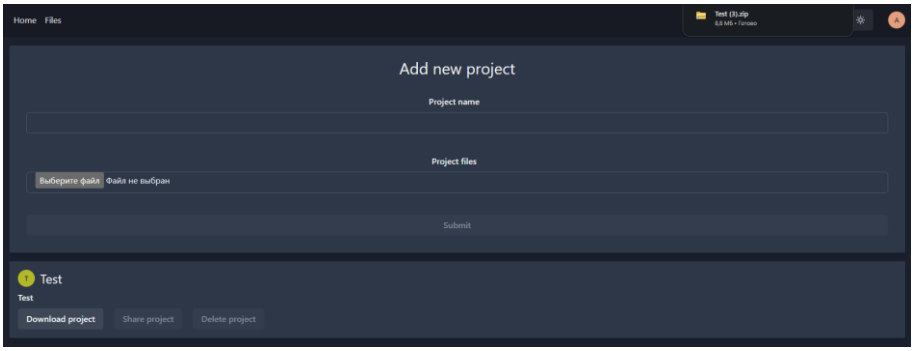


Рисунок 4.12 – Результат тестування можливості завантаження файлів для користувача з доступом

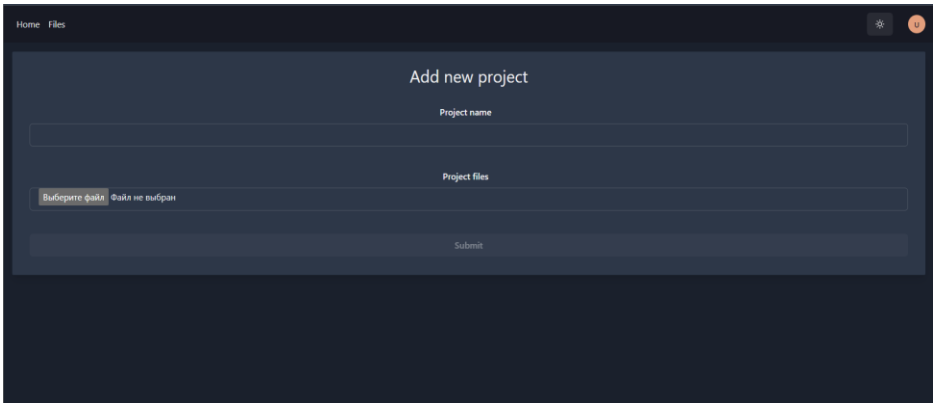


Рисунок 4.13 – Результат тестування можливості завантаження файлів для користувача без доступу

Як видно з результату тестування можливості завантаження файлів для власника інформаційного ресурсу (рис. 4.11), у нього є можливість проводити будь-які

маніпуляції з ресурсом, такі як завантаження, делегування доступами та видалення проекту, що свідчить про правильність роботи алгоритмів. Також тестування можливості завантаження файлів для користувача з доступом (рис. 4.12) показало що у користувача, якому власник ресурсу надав доступ, є можливість його завантажити, натомість немає можливості видати доступ ще комусь, або видалити проект. І останнє тестування можливості завантаження файлів для користувача без доступу (рис. 4.13) показало що користувач, якому не видавали доступ до ресурсу, в списку своїх ресурсів навіть його немає. Для тестування було використано три різних облікових записи.

У результаті успішного тестування застосунку було виявлено відсутність будь-яких загроз або проблем, що свідчить про високий рівень стабільності та надійності продукту. Всі функціональності працюють належним чином, а користувачі можуть безперешкодно скористатися всіма можливостями програми. Це підтверджує високий рівень якості розробки та ефективність проведеного тестування.

4.6 Висновки з розділу

Проведений аналіз засобів та мов програмування дозволив обґрунтувати вибір технологій для розробки, а саме React та Nest.js. React був обраний для клієнтської частини завдяки своїй гнучкості, ефективності та активній спільноті розробників. Nest.js відзначається чіткою структурою, підтримкою TypeScript та здатністю забезпечити захист та функціональність серверної частини. Обрана комбінація технологій не лише відповідає вимогам проекту, але й забезпечує швидку розробку, підтримку та масштабованість системи. У процесі розробки застосунку було проведено тестування на різних рівнях, включати блокове, інтеграційне тестування, статичне тестування безпеки та тестування всього застосунку. Результати тестування свідчать про коректність як прийнятих технічний рішень, так і їх реалізації. В цілому, вибір технологій та успішне проведене тестування підтверджують успішну реалізацію проекту з високим ступенем якості та ефективності.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Метод та засіб захищеного обміну корпоративною інформацією розробників програмного забезпечення» є покращення захисту інформації у хмарних сервісах, шляхом впровадження механізмів захисту від несанкціонованого доступу до цих даних.

Для проведення комерційного та технологічного аудиту залучаємо 3-х незалежних експертів.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, а результати зведено до таблиці 5.1 [32].

Таблиця 5.1 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	Ілля Тріщенко Middle+ FullStack Developer. ТОВ «СМІСС»	Влад Сірий Middle Backend Developer. ТОВ «СМІСС»	Влад Селезньов Team Lead. ТОВ «СМІСС»
	Бали, виставлені експертами:		
1. Технічна здійсненність концепції	4	3	4
2. Ринкові переваги (наявність аналогів)	3	4	4
3. Ринкові переваги (ціна продукту)	3	3	3
4. Ринкові переваги (технічні властивості)	2	3	3
5. Ринкові переваги (експлуатаційні витрати)	4	3	4
6. Ринкові перспективи (розмір ринку)	3	3	3
7. Ринкові перспективи (конкуренція)	1	2	2
8. Практична здійсненність (наявність фахівців)	4	3	4

9. Практична здійсненність (наявність фінансів)	3	4	3
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	4	4	4
Сума балів	$СБ_1$=39	$СБ_2$=40	$СБ_3$=42
Середньоарифметична сума балів $СБ_c$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{39+40+42}{3} = 40.3$		

За результатами розрахунків, наведених в таблиці 5.1, зробимо висновок про те, що науково-технічний рівень і рівень комерційного потенціалу розробки - високий.

5.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Метод та засіб захищеного обміну корпоративною інформацією розробників програмного забезпечення», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [32]:

$$З_о = \sum_{i=1}^{k} \frac{М_{пі} \cdot t_і}{Т_р}, \quad (5.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

$М_{пі}$ – місячний посадовий оклад конкретного дослідника, грн;

$t_і$ – число днів роботи конкретного дослідника, дн.;

$Т_р$ – середнє число робочих днів в місяці, $Т_р$=21 дні.

$$З_о = 29000 \cdot 20 / 21 = 26364 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.2 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	29000	1318,2	20	26364
Програміст	15000	681,8	45	30682
Всього				57045

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Метод та засіб захищеного обміну корпоративною інформацією розробників програмного забезпечення» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} С_і \cdot t_і, \quad (5.2)$$

де $С_і$ – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

$t_і$ – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду $С_і$ можна визначити за формулою:

$$С_і = \frac{М_М \cdot К_і \cdot К_с}{Т_р \cdot t_{зм}}, \quad (5.3)$$

де $М_М$ – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6500 грн;

$К_і$ – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [32];

$К_с$ – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$Т_р$ – середнє число робочих днів в місяці, приблизно $Т_р$ = 21 дн;

$t_{зм}$ – тривалість зміни, год.

$С_1 = 6700{,}00 \cdot 1 \cdot 1{,}65 / (21 \cdot 8) = 65{,}8$ грн.
$З_{р1} = 65{,}8 \cdot 1 = 65{,}8$ грн.

Таблиця 5.3 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
1.Підготовчі	16	1	65,8	1052,9
2.Створення архітектури	40	3	88,8	3553,4
3.Вдосконалення алгоритму	40	5	111,9	4474,6
4.Розробка застосунку	40	2	72,4	2895,4
5.Тестування	16	4	59,8	957,1
Всього				12933,4

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \qquad (5.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод}$ = (57045+ 12933,4) · 11 / 100% = 7697,67 грн.

5.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%}, \quad (5.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

$Зн$ = (57045+ 12933,4+7697,67) · 22 / 100% = 17088,84 грн.

5.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою «Метод та засіб захищеного обміну корпоративною інформацією розробників програмного забезпечення».

Витрати на матеріали ($М$), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (5.6)$$

де $Н_j$ – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу j-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 … 1,15*);

$В_j$ – маса відходів j-го найменування, кг;

$Ц_{вj}$ – вартість відходів j-го найменування, грн/кг.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.4 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Вартість витраченого матеріалу, грн
Папір канцелярський офісний (А4)	175	1	175
Папір для заміток (А5)	115	1	115
Ручка	10	1	10
Флешка	250	1	250
Всього			550
З врахуванням коефіцієнта транспортування			605

5.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Метод та засіб захищеного обміну корпоративною інформацією розробників ПЗ».

Витрати на комплектуючі К, що були використані під час виконання даного етапу роботи, розраховуються за формулою:

$$К = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i \text{ грн.}, \quad (5.7)$$

де $Н_i$ – кількість комплектуючих *i*-го виду, шт.;

$Ц_i$ – ціна комплектуючих *i*-го виду, грн.;

$К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15);

n – кількість видів комплектуючих.

Зроблені розрахунки бажано звести до таблиці:

Таблиця 5.5 – Витрати на комлектуючі

Найменування комплектувальних	Кількість	Ціна за штуку, грн.	Сума, грн.
Intel Xeon E5-2620, 64 ГБ оперативної пам'яті, 1 ТБ SSD	2	100000	200000
PostgreSQL	1	15000	15000
Мереже обладнання	1	10000	10000
JetBrains WebStorm	3	1000	3000
SSL-сертифікат	1	2000	2000
DataStorage	4	60000	240000
Всього з врахування коефіцієнт транспортних витрат			517000

5.2.5 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (5.8)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (62000 \cdot 1) / (2 \cdot 12) = 5166,67$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 5.6 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
MacBook Pro 14 2021	62000	2	2	5166,67
Робоче місце розробника ПЗ	190000	20	2	1583,33
Всього				6750,00

5.2.6 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію ($B_е$) розраховуємо за формулою:

$$B_е = \sum_{i=1}^{n} \frac{W_{уі} \cdot t_i \cdot Ц_е \cdot K_{впі}}{\eta_i}, \quad (5.9)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_е$ = 7,5 грн;

$К_{вп i}$ – коефіцієнт, що враховує використання потужності, $К_{вп i} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$В_е = 0,25 \cdot 310,0 \cdot 7,5 \cdot 0,5 / 0,8 = 363,28$ грн.

5.2.7 Службові відрядження

До статті «Службові відрядження» дослідної роботи на тему «Метод та засіб захищеного обміну корпоративною інформацією розробників програмного забезпечення» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \qquad (5.10)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$В_{св} = (57045 + 12933,4) \cdot 20 / 100\% = 13989,42$ грн.

5.2.8 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \qquad (5.11)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = (57045+ 12933,4) · 50 / 100% = 34989,42 грн.

5.2.9 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \qquad (5.12)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$В_{нзв}$ = (57045+ 12933,4) · 100 / 100% = 69978,85 грн.

Витрати на проведення науково-дослідної роботи на тему «Метод та засіб захищеного обміну корпоративною інформацією розробників програмного забезпечення» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + B_{спец} + B_{прг} + A_{обл} + B_е + B_{св} + B_{сп} + I_в + B_{нзв}. \quad (4.18)$$

$B_{заг}$=57045+12933,4+7697,67+17088,84+605+517000+6750,00+363,28+13989,42+34989,42 +69978,85 = 738447,68 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (5.13)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,7.

ЗВ = 738447,68 / 0,7 = 1054925,25 грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою «Метод та засіб захищеного обміну корпоративною інформацією розробників програмного забезпечення» передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку основу майбутнього економічного ефекту будуть формувати:

ΔN – збільшення кількості споживачів яким надається відповідна інформаційна послуга у періоди часу, що аналізуються;

N – кількість споживачів яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки, приймемо 1 особа

$Ц_б$ – вартість послуги у році до впровадження інформаційної системи, приймемо 12000,00 грн;

$\pm\Delta Ц_о$ – зміна вартості послуги від впровадження результатів, приймемо зростання на 500,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [32]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (5.14)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2021 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0{,}8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо ρ =40%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 = (1 \cdot 500 + 12000 \cdot 800) \cdot 0{,}83 \cdot 0{,}4 \cdot (1\text{-}0{,}18/100\%) = 1708350{,}4$ грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 = (1 \cdot 500 + 12000 \cdot (800 + 1000)) \cdot 0{,}83 \cdot 0{,}4 \cdot (1\text{-}0{,}18/100\%) = 3844096{,}3$ грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 = (1 \cdot 500 + 12000 \cdot (800 + 1000 + 900)) \cdot 0{,}83 \cdot 0{,}4 \cdot (1\text{-}0{,}18/100\%) = 5765894{,}4$ грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.15)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =18%;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = 1708350{,}4 /(1+0{,}18)^1+3844096{,}3 /(1+0{,}18)^2+5765894{,}4/(1+0{,}18)^3=$$
$$= 7445400{,}80 \text{ грн.}$$

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \quad (5.16)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 1054925,25 грн.

$PV = k_{інв} \cdot ЗВ = 2 \cdot 1054925{,}25 = 21009850{,}51$ грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV, \quad (5.17)$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 7445400,80 грн;

PV – теперішня вартість початкових інвестицій, 21009850,51 грн.

$E_{абс} = ПП - PV$ = 7445400,80 - 21009850,51 = 5335550,30 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.18)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1$ = (1+5335550,30 /21009850,51)$^{1/3}$ – 1 = 0,82.

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (5.19)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,1;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,25.

$\tau_{мін}$ = 0,1+0,25 = 0,35 < 0,82 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Інформаційна технологія онтологічного моделювання бази знань з організації бібліотеки» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \tag{5.20}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 0,82 = 1,2 р.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.4 Висновки з розділу

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Метод та засіб захищеного обміну корпоративною інформацією розробників програмного забезпечення» становить 40 балів, що, свідчить про комерційну важливість проведення даних досліджень оскільки рівень комерційного потенціалу розробки високий. Також термін окупності становить 1,2 роки, що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок. Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Метод та засіб захищеного обміну корпоративною інформацією розробників програмного забезпечення».

ВИСНОВКИ

Результатом виконання даної магістерської кваліфікаційної роботи є метод та програмний засіб обміну корпоративною інформацією розробників програмного забезпечення.

Аналіз сучасних засобів обміну корпоративною інформацією поміж розробників програмного забезпечення, дозволив визначити актуальність розробки засобу, який враховує бізнес-процеси життєвого циклу розробки програм і при цьому захищає корпоративні дані від атак з боку адміністраторів серверів. Завдяки цьому аналізу також було визначено кардинальні відмінності між цими засобами та структуровано основні етапи майбутньої розробки застосунку. Аналіз основних відомих загроз кібербезпеці хмарних застосункам дозволив спрогнозувати вектори потенційних атак та визначити методи захисту.

Проведення математичного опису процесу обміну корпоративною інформацією дозволило формалізувати інформацію, яку необхідно зберігати, описати можливі внутрішні стани та перетворення. Завдяки цьому визначено основні процеси, які будуть відбуватись в засобі обміну корпоративною інформацією та які потрібно врахувати під час розробки методу. Також розроблено узагальнений метод обміну корпоративною інформацією та метод шифрування даних. Завдяки формалізації методу, визначено вимоги до вибору шифрів. Створено модель розмежування прав доступу, яка є варіантом поєднання рольової та дискреційної, завдяки чому відбувається розмежування прав доступу користувачів за замовчуванням до щойно створених ресурсів. Це дозволяє розробникам скоротити час на створення правил розмежування прав доступу, що особливо важливо з урахуванням великої кількості файлів у кодовій базі сучасних проектів.

Створення архітектури серверної та клієнтських частин, з урахуванням раніше визначених загроз, дозволило виконати декомпозицію задачі розробки програмного засобу обміну корпоративною інформацією та визначити його основі алгоритми. Для наочності, було розбито ці алгоритми на окремі процедури для більш гнучкої інтеграції їх між собою.

Для тестування розроблених методів на основі раніше спроектованих алгоритмів, використовувались бібліотеки, які дозволяють автоматизувати цей процес, за допомогою перевірки значень, що повертаються. Були розроблені сценарії блокового та інтеграційного тестування, які дозволити довести коректність прийнятий рішень, крім того ці тести можна використовувати для проведення тестування інших застосунків цього типу. Проведене статичне тестування безпеки за допомогою сторонніх інструментів не виявило загроз безпеці застосунку. Для тестування готового застосунку використовувалось ручне тестування, оскільки це імітує дії реального користувача. Протестувавши застосунок щодо наявності помилок у роботі алгоритмів та інтерфейсу, визначено, що застосунок розроблено коректно, враховуючи постановку задачі відносно питань кібербезпеки.

На основі проведених економічних досліджень виявлено, що рівень комерційного потенціалу розробки складає 40 балів, свідчачи про високу комерційну важливість цієї розробки. Термін окупності цієї розробки становить 1,2 роки, що менше тривалості 3 років, що свідчить про її комерційну привабливість і можливість залучення інвестицій для впровадження на ринок. Зазначені фактори роблять висновок про доцільність подальшого проведення науково-дослідної роботи за даною темою.

Отриманий програмний засіб доцільно використовувати для продуктових, аутсорс та аутстаф компаній розробки програмного забезпечення. Також застосунок буде корисним для компаній, які не фокусуються на розробці програмного забезпечення, але мають відділи, які цим займуться, наприклад як банки, енергетичні системи, лікарні тощо. Засіб може бути удосконалено завдяки інтеграції з системою керування версіями Git.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Dinesh Kumar Saini, Krishan Kumar, Punit Gupta. Security Issues in IoT and Cloud Computing Service Models with Suggested Solutions. April 2022. URL: https://www.hindawi.com/journals/scn/2022/4943225/ (accessed: 15.10.2023)
2. Krešimir Popović; Željko Hocenski. Cloud computing security issues and challenges. URL: https://ieeexplore.ieee.org/abstract/document/5533317 (accessed: 15.10.2023)
3. Олексій Палій. Аналіз загроз кібербезпеці комп'ютерних ігор жанру RPG. Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії ВНТУ. 2022 р. 2 с. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/15765/13281 (дата звернення: 15.10.2023).
4. Олексій Палій, Ігор Віщун. Рух квадрокоптеру за траєкторією, заданою контрольними точками. Науково-технічна конференція факультету машинобудування та транспорту ВНТУ. 2022 р. 3 с. URL: https://conferences.vntu.edu.ua/index.php/all-fmt/all-fmt-2022/paper/view/15370/12925 (дата звернення: 15.10.2023).
5. Олексій Палій. Використання гібридної моделі розмежування прав доступу у застосунках обміну корпоративною інформацією. Молодь у науці: дослідження, проблеми, перспективи. ВНТУ . 2023 р. 3 с. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/19525 (дата звернення: 28.11.2023).
6. Nakita Mc Cool. What is GitHub?. 2023. URL: https://codeinstitute.net/global/blog/github-might-benefit-using (accessed: 15.10.2023).
7. Ponuthorai P. Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development / Prem Ponuthorai, Jon Loeliger. – [S. l.] : O'Reilly Media, Incorporated, 2023.

8. Erica Mixon, Ivy Wigmore. What is Google Drive?. 2022. URL: https://www.techtarget.com/searchmobilecomputing/definition/Google-Drive (accessed: 15.10.2023)
9. Ellen Cushing, How Slack Upended the Workplace. November 2021. URL: https://www.theatlantic.com/magazine/archive/2021/11/slack-office-trouble/620173/ (accessed: 15.10.2023)
10. John Brandon. Why Microsoft Teams is so much better than zoom for collaboration. URL: https://www.forbes.com/sites/johnbbrandon/2021/01/17/why-microsoft-teams-is-so-much-better-than-zoom-and-slack-for-collaboration/?sh=5a64bc705cd7 (accessed: 15.10.2023)
11. Idilio Drago, Marco Mellia, Anna Sporotto, Ramin Sadre, Aiko Pras. Inside dropbox: understanding personal cloud storage services. November 2012 URL: https://dl.acm.org/doi/abs/10.1145/2398776.2398827 (accessed: 15.10.2023)
12. Hari Narayn. Get Online with SharePoint Online. September 2023 URL: https://link.springer.com/chapter/10.1007/978-1-4842-9726-1_1 (accessed: 15.10.2023)
13. Muhammed-Amr Abd El-Migid. A Trello power-up to capture and monitor emotions of Agile teams. URL: https://www.sciencedirect.com/science/article/abs/pii/S016412122100279X (accessed: 15.10.2023)
14. Noah Buhlmann, Mohammad Ghafari. How do developers deal with security issue reports on Github? April 2022. URL: https://dl.acm.org/doi/abs/10.1145/3477314.3507123 (accessed: 15.10.2023).
15. Darren Quick. Google Drive: Forensic analysis of data remnants. April 2021. URL: https://www.sciencedirect.com/science/article/abs/pii/S1084804513002051 (accessed: 15.10.2023)
16. Joel Witts. How Secure Is Slack For Your Business? 29 March 2023. URL: https://expertinsights.com/insights/how-secure-is-slack-for-your-business/ (accessed: 15.10.2022)

17. Jason Stagnitto. Dropbox Security 2023: The Good, the Bad and the Ugly. URL: https://www.cloudwards.net/dropbox-security/ (accessed: 15.10.2023)
18. Risks and Vulnerabilities: Is SharePoint Secure? URL: https://www.mrsharepoint.guru/is-sharepoint-secure/ (accessed: 15.10.2023)
19. Trello App Exposes Personally Identifiable Information of its Users. URL: https://cisomag.com/trello-app-exposes-personally-identifiable-information-of-its-users/ (accessed: 15.10.2023).
20. Лужецький В. А., Кожухівський А. Д, Войтович О. П Основи інформаційної безпеки : навчальний посібник Вінниця : ВНТУ, 2013. 221 с.
21. Chadi RIMAN, and Pierre E. ABI-CHAR, «Comparative Analysis of Block Cipher-BasedEncryption Algorithms: A Survey.» Information Security and Computer Fraud , vol. 3, no. 1 (2015): 1-7. doi:10.12691/iscf-3-1-1.
22. Баришев Ю. В., Каплун В. А., Неуйміна К. В. Дискреційна модель та метод розмежування прав доступу до розподілених інформаційних ресурсів / *Наукові праці ВНТУ*, № 2, 2017. 8 с . URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/26676/506-%d0%a2%d0%b5%d0%ba%d1%81%d1%82%20%d1%81%d1%82%d0%b0%d1%82%d1%82%d1%96-560-1-10-20170912.pdf?sequence=1&isAllowed=y . (дата звернення: 14.10.2023).
23. Oliveira Rocha H. F. Practical Event-Driven Microservices Architecture. Berkeley, CA : Apress, 2022. URL: https://doi.org/10.1007/978-1-4842-7468-2 (accessed: 25.11.2023).
24. Ski B. K. FIFO: First in, First Out. Argyll Productions, 2022.
25. Osuna A. IBM System Storage Data Encryption. Poughkeepsie, NY : IBM, International Technical Support Organization, 2010. 1008 p.
26. Crockford D. JavaScript: The good parts. Sebastopol, CA : O'Reilly, 2008. 153 p.
27. Stroustrup B. C++ Programming Language. Addison Wesley, 2013. 1368 p.
28. Holmes D., Gosling J., Arnold K. Java Programming Language. Pearson Education, Limited, 2021. 992 p.

29. Boduch A. React and React Native. Packt Publishing - ebooks Account, 2017. 500 p.
30. Dubois S., Georges A. Learn TypeScript 3 by Building Web Applications: Gain a Solid Understanding of TypeScript, Angular, Vue, React, and NestJS. Packt Publishing, Limited, 2019. 804 p.
31. Blokdyk G. Web Application Security Testing a Complete Guide - 2020 Edition. Emereo Pty Limited, 2020. 320 p.
32. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

ДОДАТКИ

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

УЗГОДЖЕНО

Директор ТОВ «СМІСС»

_____ Марина ВЕРТИКОВА
підпис

«__1__» _____листопада_____ 2023 р.

ЗАТВЕРДЖУЮ

Завідувач кафедри ЗІ

д. т. н., проф.

_____ Володимир ЛУЖЕЦЬКИЙ

«__01__» _____11_____ 2023 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

на тему: "Метод та засіб захищеного обміну корпоративною інформацією розробників програмного забезпечення"

08-20.МКР.021.00.000 ТЗ

Керівник магістерської кваліфікаційної роботи
к. т. н., доц.

_____ Юрій БАРИШЕВ

Вінниця 2023

1 Назва та область використання

Модель розмежування прав доступу
Область використання: захист комерційних даних від несанкціонованого копіювання
Розробка виконується на основі наказу ВНТУ №247 від 18.09.23.

2 Мета та призначення розробки

Підвищення ефективності захисту комерційних даних шляхом впровадження правильної та ефективної моделі розмежування прав доступу.

3 Джерела розробки

3.1 В. А. Лужецький, А. Д. Кожухівський, О. П. Войтович. Основи інформаційної безпеки : навчальний посібник / Вінниця: ВНТУ, 2013. 221 с

3.2 Ю.В. Баришев, В.А. Каплун, К.В. Неуйміна Дискреційна модель та метод розмежування прав доступу до розподілених інформаційних ресурсів / *Наукові праці ВНТУ*, 2017, № 2. Вінниця, 2017.

3.3 Dinesh Kumar Saini, Krishan Kumar, Punit Gupta. Security Issues in IoT and Cloud Computing Service Models with Suggested Solutions. April 2022. URL: https://www.hindawi.com/journals/scn/2022/4943225/ (accessed: 15.09.2023)

3.4 Prem Ponuthorai, Jon Loeliger. Version Control with Git: Powerful Tools and Techniques for Collaborative Software Developmen[S. l.] : O'Reilly Media, Incorporated, 2023.

4 Виконавці МКР

Студент групи 2БС-22м Палій Олексій Миколайович

5 Вимоги до програми

5.1 Вимоги до функціональних характеристик:
- програма повинна підтримувати оновлення інтерфейсу;
- можливість реєстрації та авторизації користувача;
- можливість завантаження та вивантаження файлів.
- можливість власнику інформаційного ресурсу розподіляти доступ до своїх ресурсів

5.2 Вимоги до нефункціональних характеристик:
- можливість верифікації даних;
- застосування симетричного шифрування з довжиною ключа не менше 256 біт для захисту файлів.

5.3 Вимоги до складу і параметрів технічних засобів:
- тактова частота процесора – від 1,8 ГГц;
- обсяг оперативної пам’яті – не менше 1 Гб.

5.4 Вимоги до інформаційної та програмної сумісності:
- програма вимагає наявність одного з браузерів на комп’ютері користувача (Safari, Mozilla Firefox, Google Chrome, Microsoft Edge)

6 Стадії та етапи розробки

№	Зміст	Початок - закінчення	Результат
1	Аналіз завдання. Вступ	01.09.2023 - 10.09.2023	Чернетка вступу
2	Аналіз літературних джерел за напрямком магістерської кваліфікаційної роботи	10.09.2023 - 15.09.2023	Чернего першого розділу
3	Науково-технічне обґрунтування	16.09.2023 - 22.09.2023	Чернетка першого розділу
4	Розробка технічного завдання	23.09.2023 - 29.09.2023	Технічне завдання
5	Розробка рішень	30.09.2023 - 12.10.2023	Чернетка другого розділку
6	Практична реалізація, моделювання, експериментування, результати	14.10.2023 - 10.11.2023	Чернетка третього та четвертих розділів
7	Розробка розділу економічного обґрунтування доцільності розробки	11.11.2023 - 17.11.2023	Чернетка економічного розділу
8	Аналіз виконання ТЗ, висновки	18.11.2023 - 24.11.2023	Чернетка висновків
9	Оформлення пояснювальної записки	25.11.2023 - 30.11.2023	Пояснювальна записка

7 Порядок контролю та приймання

7.1 До приймання МКР представляється:

- пояснювальна записка до МКР;
- програма для реалізації захисту;
- результати функціонального тестування програми;
- ілюстративні матеріали для захисту.

7.2 Попередній захист на кафедрі: 28 листопада – 1 грудня 2023 р.

7.3 Представлення МКР до захисту: 11 – 14 грудня 2023р.

7.4 Захист МКР: 14 – 21 грудня 2023.

Розробив студент групи 2БС-22м  Олексій ПАЛІЙ

ДОДАТОК Б

Текст програми серверної частини

main.ts:

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { ConfigService } from '@nestjs/config';
import { CookieEnum, EnvEnum } from '@common/enums';
import * as cookieParser from 'cookie-parser';
import { ValidationPipe } from '@nestjs/common';
import { DocumentBuilder, SwaggerModule } from '@nestjs/swagger';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.use(cookieParser());

  app.enableCors({
    origin: true,
    methods: 'GET,PATCH,PUT,POST,DELETE',
    credentials: true,
  });

  app.useGlobalPipes(
    new ValidationPipe({
      transform: true,
      whitelist: true,
      disableErrorMessages: false,
    }),
  );

  const options = new DocumentBuilder()
    .setTitle('Graynode - API')
    .setVersion('1.0')
    .addBearerAuth()
    .addCookieAuth(CookieEnum.REFRESH_TOKEN)
    .build();
  const document = SwaggerModule.createDocument(app, options);
  SwaggerModule.setup('api', app, document);

  const configService = new ConfigService();
  const port = parseInt(await configService.get(EnvEnum.PORT));

  await app.listen(port);
}
bootstrap();
```

app.module.ts:

```
import { Module } from '@nestjs/common';
import { UserModule } from './modules/user/user.module';
import { AuthModule } from './modules/auth/auth.module';
import { FileModule } from './modules/file/file.module';

@Module({
  imports: [UserModule, AuthModule, FileModule],
})
export class AppModule {}
```

database.module.ts:

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { EnvEnum } from '@common/enums';

@Module({
  imports: [
    TypeOrmModule.forRootAsync({
```

```
      inject: [ConfigService],
      useFactory: (configService: ConfigService) => ({
        type: 'postgres',
        host: configService.get<string>(EnvEnum.DATABASE_HOST),
        port: configService.get<number>(EnvEnum.DATABASE_PORT),
        username: configService.get<string>(EnvEnum.DATABASE_USER),
        password: configService.get<string>(EnvEnum.DATABASE_PASSWORD),
        database: configService.get<string>(EnvEnum.DATABASE_NAME),
        autoLoadEntities: true,
        synchronize: true,
      }),
    }),
    ConfigModule.forRoot({ isGlobal: true }),
  ],
})
export class DatabaseModule {}
```

file.entity.ts:

```
import {
  CreateDateColumn,
  Entity,
  Index,
  ManyToOne,
  PrimaryColumn,
  UpdateDateColumn,
} from 'typeorm';
import { User } from '@shared/entities/user.entity';

@Entity({ name: 'Files' })
export class File {
  @PrimaryColumn({ type: 'uuid' })
  id: string;

  @ManyToOne(() => User, (user) => user.sessions, { nullable: false })
  @Index('file-user-idx')
  user: User;

  @CreateDateColumn({
    type: 'timestamp',
    default: () => 'CURRENT_TIMESTAMP(6)',
  })
  createdAt: Date;

  @UpdateDateColumn({
    type: 'timestamp',
    default: () => 'CURRENT_TIMESTAMP(6)',
    onUpdate: 'CURRENT_TIMESTAMP(6)',
  })
  updatedAt: Date;
}
```

user.entity.ts:

```
import {
  Column,
  CreateDateColumn,
  Entity,
  Index,
  OneToMany,
  PrimaryGeneratedColumn,
  UpdateDateColumn,
} from 'typeorm';
import { File, UserSession } from '@shared/entities';

@Entity({ name: 'Users' })
export class User {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @Column({ unique: true })
  @Index('login-idx', { unique: true })
  login: string;

  @Column({ nullable: false })
  symmetricKey: string;
```

```
  @Column({ nullable: false })
  mnemonicPhrase: string;

  @OneToMany(() => UserSession, (userSession) => userSession.user)
  sessions: UserSession[];

  @OneToMany(() => File, (file) => file.user)
  files: File[];

  @CreateDateColumn({
    type: 'timestamp',
    default: () => 'CURRENT_TIMESTAMP(6)',
  })
  createdAt: Date;

  @UpdateDateColumn({
    type: 'timestamp',
    default: () => 'CURRENT_TIMESTAMP(6)',
    onUpdate: 'CURRENT_TIMESTAMP(6)',
  })
  updatedAt: Date;
}
```

userSession.entity.ts:

```
import {
  Column,
  CreateDateColumn,
  Entity,
  Index,
  ManyToOne,
  PrimaryGeneratedColumn,
  UpdateDateColumn,
} from 'typeorm';
import { User } from '@shared/entities/user.entity';
import { SessionTypeEnum } from '@common/enums';

@Entity({ name: 'UserSessions' })
export class UserSession {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @ManyToOne(() => User, (user) => user.sessions, { nullable: false })
  @Index('userSession-user-idx')
  user: User;

  @Column({ nullable: false })
  ipAddress: string;

  @Column({ nullable: false, type: 'enum', enum: SessionTypeEnum })
  sessionType: SessionTypeEnum;

  @Column({ nullable: false, default: true })
  isActive: boolean;

  @Column({ nullable: true })
  refreshToken: string | null;

  @CreateDateColumn({
    type: 'timestamp',
    default: () => 'CURRENT_TIMESTAMP(6)',
  })
  createdAt: Date;

  @UpdateDateColumn({
    type: 'timestamp',
    default: () => 'CURRENT_TIMESTAMP(6)',
    onUpdate: 'CURRENT_TIMESTAMP(6)',
  })
  updatedAt: Date;

  @Column({ type: 'timestamp', nullable: true })
  expiredAt: Date;
}
```

file.repository.ts:

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { File } from '@shared/entities';
import { AbstractRepository } from '@common/abstract';

@Injectable()
export class FileRepository extends AbstractRepository<File> {
  constructor(
    @InjectRepository(File) private readonly repository: Repository<File>,
  ) {
    super(repository);
  }
}
```

user.repository.ts:

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { User } from '@shared/entities';
import { AbstractRepository } from '@common/abstract';

@Injectable()
export class UserRepository extends AbstractRepository<User> {
  constructor(
    @InjectRepository(User) private readonly repository: Repository<User>,
  ) {
    super(repository);
  }
}
```

userSession.repository.ts:

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { UserSession } from '@shared/entities';
import { AbstractRepository } from '@common/abstract';

@Injectable()
export class UserSessionRepository extends AbstractRepository<UserSession> {
  constructor(
    @InjectRepository(UserSession)
    private readonly repository: Repository<UserSession>,
  ) {
    super(repository);
  }
}
```

shared.module.ts:

```
import { Module } from '@nestjs/common';
import { DatabaseModule } from './database/database.module';
import { File, User, UserSession } from '@shared/entities';
import { TypeOrmModule } from '@nestjs/typeorm';
import {
  FileRepository,
  UserRepository,
  UserSessionRepository,
} from '@shared/repositories';

const entities = [User, UserSession, File];
const repositories = [UserRepository, UserSessionRepository, FileRepository];

@Module({
  imports: [DatabaseModule, TypeOrmModule.forFeature(entities)],
  providers: [...repositories],
  exports: [...repositories],
})
export class SharedModule {}
```

abstract.repository.ts:

```
import { Injectable } from '@nestjs/common';
import { FindOptionsWhere, Repository } from 'typeorm';
import { PageType } from '@common/types';
```

```
@Injectable()
export class AbstractRepository<Entity> {
  constructor(protected readonly abstractRepository: Repository<Entity>) {}

  async findAll(relations = []): Promise<Entity[]> {
    return this.abstractRepository.find({ relations });
  }

  async paginate(
    take = 15,
    page = 1,
    relations = [],
  ): Promise<PageType<Entity> | null> {
    if (take < 1) return null;

    const [data, total] = await this.abstractRepository.findAndCount({
      take,
      skip: (page - 1) * take,
      relations,
    });

    return {
      data: data,
      meta: {
        total,
        page,
        lastPage: Math.ceil(total / take),
      },
    };
  }

  async create(data): Promise<Entity> {
    return this.abstractRepository.save(data);
  }

  async findOne(
    condition: FindOptionsWhere<Entity>,
    relations: string[] = [],
  ): Promise<Entity> {
    return this.abstractRepository.findOne({
      where: condition,
      relations,
    });
  }

  async update(id: string, data): Promise<any> {
    return this.abstractRepository.update(id, data);
  }

  async delete(id: string): Promise<any> {
    return this.abstractRepository.delete(id);
  }
}
```

tokensTTL.const.ts:

```
import { addWeeks } from 'date-fns';

export const TokensTTLConst = Object.freeze({
  accessTokenTTL: '30m',
  refreshTokenTTL: '14d',
  refreshTokenCookieTTL: addWeeks(new Date(), 2),
});
```

getUser.decorator.ts:

```
import { createParamDecorator, ExecutionContext } from '@nestjs/common';
import { JwtPayload } from '@common/types';

export const GetUser = createParamDecorator(
  (data: unknown, ctx: ExecutionContext): JwtPayload => {
    const request = ctx.switchToHttp().getRequest();
    return request.user;
  },
);
```

ipAddress.decorator.ts:

```
import { createParamDecorator, ExecutionContext } from '@nestjs/common';
import type { Request as RequestType } from 'express';

export const IpAddress = createParamDecorator(
  (data: unknown, ctx: ExecutionContext): string => {
    const request: RequestType = ctx.switchToHttp().getRequest();
    return request.socket.remoteAddress;
  },
);
```

cookie.enum.ts:

```
export enum CookieEnum {
  REFRESH_TOKEN = 'graynode-refreshToken',
}
```

env.enum.ts:

```
export enum EnvEnum {
  NODE_ENV = 'NODE_ENV',

  PORT = 'PORT',

  DATABASE_HOST = 'DATABASE_HOST',
  DATABASE_NAME = 'DATABASE_NAME',
  DATABASE_USER = 'DATABASE_USER',
  DATABASE_PASSWORD = 'DATABASE_PASSWORD',
  DATABASE_PORT = 'DATABASE_PORT',

  JWT_ACCESS_SECRET = 'JWT_ACCESS_SECRET',
  JWT_REFRESH_SECRET = 'JWT_REFRESH_SECRET',
}
```

sessionType.enum.ts:

```
export enum SessionTypeEnum {
  TEMPORARY = 'TEMPORARY',
  PERMANENT = 'PERMANENT',
}
```

response.factory.ts:

```
import { HttpStatus } from '@nestjs/common';

export class HttpFactory {
  static okResponse() {
    return {
      statusCode: HttpStatus.OK,
      message: 'Successfully',
    };
  }

  static createdResponse() {
    return {
      statusCode: HttpStatus.CREATED,
      message: 'Successfully created',
    };
  }
}
```

accessToken.guard.ts:

```
import { Injectable } from '@nestjs/common';
import { AuthGuard } from '@nestjs/passport';

@Injectable()
export class AccessTokenGuard extends AuthGuard('jwt-access-token') {}
```

refreshToken.guard.ts:

```
import { Injectable } from '@nestjs/common';
import { AuthGuard } from '@nestjs/passport';

@Injectable()
export class RefreshTokenGuard extends AuthGuard('jwt-refresh-token') {}
```

accessToken.strategy.ts:

```
import { Injectable } from '@nestjs/common';
import { PassportStrategy } from '@nestjs/passport';
import { ExtractJwt, Strategy } from 'passport-jwt';
import { JwtPayload } from '@common/types';
import { ConfigService } from '@nestjs/config';
import { EnvEnum } from '@common/enums';

@Injectable()
export class AccessTokenStrategy extends PassportStrategy(
  Strategy,
  'jwt-access-token',
) {
  constructor(private readonly configService: ConfigService) {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      secretOrKey: configService.get<string>(EnvEnum.JWT_ACCESS_SECRET),
    });
  }

  validate(payload: JwtPayload) {
    return payload;
  }
}
```

refreshToken.strategy.ts:

```
import { PassportStrategy } from '@nestjs/passport';
import { ExtractJwt, Strategy } from 'passport-jwt';
import { Request as RequestType } from 'express';
import { Injectable } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { CookieEnum, EnvEnum } from '@common/enums';

@Injectable()
export class RefreshTokenStrategy extends PassportStrategy(
  Strategy,
  'jwt-refresh-token',
) {
  constructor(private readonly configService: ConfigService) {
    super({
      jwtFromRequest: ExtractJwt.fromExtractors([
        (req: RequestType) => {
          const token = req?.cookies[CookieEnum.REFRESH_TOKEN];
          if (!token) {
            return null;
          }
          return token;
        },
      ]),
      secretOrKey: configService.get<string>(EnvEnum.JWT_REFRESH_SECRET),
      passReqToCallback: true,
    });
  }

  validate(req: RequestType, payload: any) {
    const refreshToken = req.cookies[CookieEnum.REFRESH_TOKEN];
    return { ...payload, refreshToken };
  }
}
```

auth.controller.ts:

```
import {
  Body,
  Controller,
  Get,
  HttpCode,
  HttpStatus,
  Post,
  Put,
  Res,
  UseGuards,
} from '@nestjs/common';
import type { Response as ResponseType } from 'express';
```

```
import { AuthService } from './auth.service';
import { AuthDto, CreateUserDto, UpdateUserDto } from '@common/dto';
import { AccessTokenGuard, RefreshTokenGuard } from '@common/guards';
import { CookieEnum, EnvEnum } from '@common/enums';
import { User } from '@shared/entities';
import { ApiBearerAuth, ApiCookieAuth, ApiTags } from '@nestjs/swagger';
import { GetUser, IpAddress } from '@common/decorators';
import { JwtPayload } from '@common/types';
import { TokensTTLConst } from '@common/consts';
import { ConfigService } from '@nestjs/config';

@ApiTags('Auth', 'User')
@Controller('auth')
export class AuthController {
  constructor(
    private readonly authService: AuthService,
    private readonly configService: ConfigService,
  ) {}

  @HttpCode(HttpStatus.CREATED)
  @Post('signUp')
  async signUp(@Body() body: CreateUserDto): Promise<User> {
    return this.authService.signUp(body);
  }

  @HttpCode(HttpStatus.OK)
  @Post('signIn')
  async signIn(
    @IpAddress() ipAddress: string,
    @Res({ passthrough: true }) res: ResponseType,
    @Body() body: AuthDto,
  ): Promise<{ accessToken: string }> {
    const { accessToken, refreshToken } = await this.authService.signIn({
      ipAddress,
      ...body,
    });
    if (body.isRemember) {
      this.addCookie(res, refreshToken);
    }
    return { accessToken };
  }

  @ApiBearerAuth()
  @UseGuards(AccessTokenGuard)
  @HttpCode(HttpStatus.OK)
  @Put('/userProfile')
  async updateUser(
    @GetUser() { sub: userId }: JwtPayload,
    @Res() res: ResponseType,
    @Body() body: UpdateUserDto,
  ) {
    const user = await this.authService.updateUser({ userId, ...body });
    await this.authService.logout(user.id);
    this.clearCookie(res);
    return res.json();
  }

  @ApiBearerAuth()
  @HttpCode(HttpStatus.OK)
  @UseGuards(AccessTokenGuard)
  @Get('logout')
  async logout(
    @GetUser() { sub: userId }: JwtPayload,
    @Res() res: ResponseType,
  ): Promise<ResponseType> {
    await this.authService.logout(userId);
    this.clearCookie(res);
    return res.json();
  }

  @ApiCookieAuth(CookieEnum.REFRESH_TOKEN)
  @HttpCode(HttpStatus.OK)
  @UseGuards(RefreshTokenGuard)
  @Get('refresh')
  async refresh(
    @IpAddress() ipAddress: string,
    @GetUser() { sub: userId, refreshToken }: JwtPayload,
```

```
    @Res({ passthrough: true }) res: ResponseType,
  ): Promise<{ accessToken: string }> {
    const { accessToken, refreshToken: newRefreshToken } =
      await this.authService.refreshTokens(userId, refreshToken, ipAddress);
    this.addCookie(res, newRefreshToken);
    return { accessToken };
  }

  private clearCookie(res: ResponseType) {
    res.clearCookie(CookieEnum.REFRESH_TOKEN);
  }

  private addCookie(res: ResponseType, refreshToken: string) {
    res.cookie(CookieEnum.REFRESH_TOKEN, refreshToken, {
      httpOnly: true,
      secure: this.configService.get<string>(EnvEnum.NODE_ENV) === 'production',
      expires: TokensTTLConst.refreshTokenCookieTTL,
    });
  }
}
```

auth.module.ts:

```
import { Module } from '@nestjs/common';
import { JwtModule } from '@nestjs/jwt';
import { AccessTokenStrategy } from './strategies/accessToken.strategy';
import { AuthService } from './auth.service';
import { RefreshTokenStrategy } from './strategies/refreshToken.strategy';
import { UserModule } from '../user/user.module';
import { UserSessionModule } from '../userSession/userSession.module';
import { AuthController } from './auth.controller';

@Module({
  imports: [UserModule, UserSessionModule, JwtModule.register({})],
  controllers: [AuthController],
  providers: [AuthService, AccessTokenStrategy, RefreshTokenStrategy],
})
export class AuthModule {}
```

auth.service.ts:

```
import {
  BadRequestException,
  ForbiddenException,
  Injectable,
  NotFoundException,
} from '@nestjs/common';
import * as argon2 from 'argon2';
import { JwtService } from '@nestjs/jwt';
import { ConfigService } from '@nestjs/config';
import { UserService } from '../user/user.service';
import { AuthDto, CreateUserDto, UpdateUserDto } from '@common/dto';
import { EnvEnum, SessionTypeEnum } from '@common/enums';
import { UserSessionService } from '../userSession/userSession.service';
import { User, UserSession } from '@shared/entities';
import { TokensType } from '@common/types';
import { TokensTTLConst } from '@common/consts';

@Injectable()
export class AuthService {
  constructor(
    private readonly userService: UserService,
    private readonly userSessionService: UserSessionService,
    private readonly jwtService: JwtService,
    private readonly configService: ConfigService,
  ) {}
  async signUp(createUserDto: CreateUserDto): Promise<User> {
    const userExists = await this.userService.findOneByLogin(
      createUserDto.login,
    );
    if (userExists) {
      throw new BadRequestException('User already exists');
    }

    const hashedKey = await this.hash(createUserDto.symmetricKey);
    const hashedMnemonic = await this.hash(createUserDto.mnemonicPhrase);
```

```
  return this.userService.create({
    ...createUserDto,
    symmetricKey: hashedKey,
    mnemonicPhrase: hashedMnemonic,
  });
}

async signIn(data: AuthDto): Promise<TokensType> {
  const user = await this.userService.findOneByLogin(data.login);
  if (!user) throw new NotFoundException('User does not exist');

  const isValidPassword = await this.verify(
    user.symmetricKey,
    data.symmetricKey,
  );
  if (!isValidPassword) throw new BadRequestException('Key is incorrect');

  if (data.isRemember) {
    const tokens = await this.getTokens(user.id, user.login);
    await this.createSession({
      userId: user.id,
      refreshToken: tokens.refreshToken,
      ipAddress: data.ipAddress,
      sessionType: SessionTypeEnum.PERMANENT,
    });
    return tokens;
  }
  await this.createSession({
    userId: user.id,
    ipAddress: data.ipAddress,
    sessionType: SessionTypeEnum.TEMPORARY,
  });
  return await this.getOnlyAccessToken(user.id, user.login);
}

async updateUser(payload: UpdateUserDto): Promise<User> {
  const data = { ...payload };
  if (data.login) {
    const userExists = await this.userService.findOneByLogin(data.login);
    if (userExists) {
      throw new BadRequestException('Username already exists');
    }
  }
  if (data.password) {
    data.password = await this.hash(data.password);
  }
  return this.userService.update(data);
}

async logout(userId: string): Promise<void> {
  const lastUserSession = await this.userSessionService.findOneByUserId(
    userId,
  );
  if (!lastUserSession) {
    throw new BadRequestException('User does not has any sessions');
  }

  await this.userSessionService.update(lastUserSession.id, {
    refreshToken: null,
    isActive: false,
    expiredAt: new Date(),
  });
}

private async createSession({
  userId,
  sessionType,
  refreshToken,
  ipAddress,
}: {
  userId: string;
  sessionType: SessionTypeEnum;
  refreshToken?: string;
  ipAddress: string;
}): Promise<UserSession> {
  const newRefreshToken = refreshToken ? await this.hash(refreshToken) : null;
```

```
  const lastSession = await this.userSessionService.findOneByUserId(userId);

  if (lastSession) {
    await this.userSessionService.update(lastSession.id, {
      isActive: false,
      refreshToken: null,
      expiredAt: new Date(),
    });
  }

  return this.userSessionService.create({
    user: { id: userId },
    ipAddress,
    sessionType,
    refreshToken: newRefreshToken,
  });
}

private async updateRefreshToken(
  sessionId: string,
  refreshToken: string,
  ipAddress: string,
): Promise<UserSession> {
  const hashedRefreshToken = await this.hash(refreshToken);
  return this.userSessionService.update(sessionId, {
    ipAddress,
    refreshToken: hashedRefreshToken,
  });
}

private async getOnlyAccessToken(
  userId: string,
  login: string,
): Promise<{ accessToken: string }> {
  const accessToken = await this.jwtService.signAsync(
    {
      sub: userId,
      login,
    },
    {
      secret: this.configService.get<string>(EnvEnum.JWT_ACCESS_SECRET),
      expiresIn: TokensTTLConst.accessTokenTTL,
    },
  );
  return { accessToken };
}

private async getTokens(userId: string, login: string) {
  const [accessToken, refreshToken] = await Promise.all([
    this.jwtService.signAsync(
      {
        sub: userId,
        login,
      },
      {
        secret: this.configService.get<string>(EnvEnum.JWT_ACCESS_SECRET),
        expiresIn: TokensTTLConst.accessTokenTTL,
      },
    ),
    this.jwtService.signAsync(
      {
        sub: userId,
        login,
      },
      {
        secret: this.configService.get<string>(EnvEnum.JWT_REFRESH_SECRET),
        expiresIn: TokensTTLConst.refreshTokenTTL,
      },
    ),
  ]);

  return {
    accessToken,
    refreshToken,
  };
}
```

```
  private async hash(data: string): Promise<string> {
    return argon2.hash(data);
  }

  private async verify(hash: string, hashToCompare: string): Promise<boolean> {
    return argon2.verify(hash, hashToCompare);
  }

  async refreshTokens(userId: string, refreshToken: string, ipAddress: string) {
    const { id, login, sessions } = await this.userService.findOneByUserId(
      userId,
    );
    const correctSession = sessions
      .filter(
        (session) =>
          session.isActive && session.sessionType === SessionTypeEnum.PERMANENT,
      )
      .find(async (session) => {
        return await argon2.verify(session.refreshToken, refreshToken);
      });
    if (!correctSession || !correctSession.refreshToken)
      throw new ForbiddenException('Access Denied');
    const tokens = await this.getTokens(id, login);
    await this.updateRefreshToken(
      correctSession.id,
      tokens.refreshToken,
      ipAddress,
    );
    return tokens;
  }
}
```

user.controller.ts:

```
import {
  Controller,
  Get,
  HttpCode,
  HttpStatus,
  UseGuards,
} from '@nestjs/common';
import { ApiBearerAuth, ApiOkResponse, ApiTags } from '@nestjs/swagger';
import { AccessTokenGuard } from '@common/guards';
import { JwtPayload } from '@common/types';
import { GetUser } from '@common/decorators';
import { GetUserResponse } from '@common/responses';

@ApiTags('User')
@Controller('users')
export class UserController {
  @ApiBearerAuth()
  @ApiOkResponse({ description: 'Return user', type: GetUserResponse })
  @UseGuards(AccessTokenGuard)
  @HttpCode(HttpStatus.OK)
  @Get('/userProfile')
  async getUser(@GetUser() user: JwtPayload) {
    return {
      id: user.sub,
      login: user.login,
    };
  }
}
```

user.module.ts:

```
import { Module } from '@nestjs/common';
import { UserService } from './user.service';
import { SharedModule } from '@shared/shared.module';
import { UserController } from './user.controller';

@Module({
  imports: [SharedModule],
  controllers: [UserController],
  providers: [UserService],
  exports: [UserService],
})
export class UserModule {}
```

user.service.ts:

```
import { Injectable } from '@nestjs/common';
import { UserRepository } from '@shared/repositories';
import { CreateUserDto, UpdateUserDto } from '@common/dto';

@Injectable()
export class UserService {
  constructor(private readonly userRepository: UserRepository) {}

  async findOneByLogin(login: string) {
    return this.userRepository.findOne({ login });
  }

  async findOneByUserId(userId: string) {
    return this.userRepository.findOne({ id: userId }, ['sessions']);
  }

  async create(payload: CreateUserDto) {
    return this.userRepository.create(payload);
  }

  async update(payload: UpdateUserDto) {
    const { userId, ...data } = payload;
    return this.userRepository.update(userId, data);
  }
}
```

userSession.module.ts:

```
import { Module } from '@nestjs/common';
import { UserSessionService } from './userSession.service';
import { SharedModule } from '@shared/shared.module';

@Module({
  imports: [SharedModule],
  providers: [UserSessionService],
  exports: [UserSessionService],
})
export class UserSessionModule {}
```

userSession.service.ts:

```
import { Injectable } from '@nestjs/common';
import { UserSessionRepository } from '@shared/repositories';
import { UserSession } from '@shared/entities';

@Injectable()
export class UserSessionService {
  constructor(private readonly userSessionRepository: UserSessionRepository) {}

  async create(data): Promise<UserSession> {
    return this.userSessionRepository.create(data);
  }

  async findOneByUserId(userId: string) {
    return this.userSessionRepository.findOne({ user: { id: userId } }, [
      'user',
    ]);
  }

  async update(id: string, data): Promise<UserSession> {
    return this.userSessionRepository.update(id, data);
  }
}
```

file.controller.ts:

```
import {
  Controller,
  Get,
  Post,
  Query,
  UploadedFile,
  UseGuards,
  UseInterceptors,
```

```
} from '@nestjs/common';
import { AccessTokenGuard } from '@common/guards';
import { FileInterceptor } from '@nestjs/platform-express';
import { diskStorage } from 'multer';
import { Request } from 'express';
import { extname } from 'path';
import * as crypto from 'crypto';
import { GetUser } from '@common/decorators';
import { JwtPayload } from '@common/types';
import { FileService } from './file.service';

@Controller('files')
export class FileController {
  constructor(private readonly fileService: FileService) {}

  @UseGuards(AccessTokenGuard)
  @Post('/uploadProject')
  @UseInterceptors(
    FileInterceptor('file', {
      storage: diskStorage({
        destination: './projects/',
        filename(
          req: Request,
          file: Express.Multer.File,
          callback: (error: Error | null, filename: string) => void,
        ) {
          const uuid = crypto.randomUUID();
          console.log(file);

          callback(null, `${uuid}${extname(file.originalname)}`);
        },
      }),
    }),
  )
  async uploadProject(
    @GetUser() user: JwtPayload,
    @UploadedFile() file: Express.Multer.File,
  ) {
    const fileId = file.filename.split('.')[0];
    console.log(file);

    return this.fileService.saveFile(fileId, user.sub);
  }

  @UseGuards(AccessTokenGuard)
  @Get('/project')
  async getProject(
    @GetUser() user: JwtPayload,
    @Query('fileId') fileId: string,
  ) {
    return this.fileService.getFile(fileId, user.sub);
  }

  @UseGuards(AccessTokenGuard)
  @Get()
  async getProjects(@GetUser() user: JwtPayload) {
    return this.fileService.getFiles(user.sub);
  }
}
```

file.module.ts:

```
import { Module } from '@nestjs/common';
import { FileController } from './file.controller';
import { SharedModule } from '@shared/shared.module';
import { FileService } from './file.service';

@Module({
  imports: [SharedModule],
  controllers: [FileController],
  providers: [FileService],
})
export class FileModule {}
```

file.service.ts:

```
import {
```

```
  BadRequestException,
  Injectable,
  StreamableFile,
} from '@nestjs/common';
import { FileRepository } from '@shared/repositories';
import { createReadStream } from 'fs';
import { join } from 'path';

@Injectable()
export class FileService {
  constructor(private readonly fileRepository: FileRepository) {}

  async saveFile(fileId: string, userId: string) {
    return this.fileRepository.create({
      user: { id: userId },
      id: fileId,
    });
  }

  async getFiles(userId: string) {
    const isUserOwner = await this.fileRepository.findOne({
      user: { id: userId },
    });

    if (!isUserOwner) {
      throw new BadRequestException('File is not related to user');
    }

    try {
      const file = createReadStream(
        join(process.cwd(), `./projects/${isUserOwner.id}.zip`),
      );

      return new StreamableFile(file);
    } catch (e) {
      console.log(e);
    }
  }

  async getFile(fileId: string, userId: string) {
    const isUserOwner = await this.fileRepository.findOne({
      id: fileId,
      user: { id: userId },
    });

    if (!isUserOwner) {
      throw new BadRequestException('File is not related to user');
    }

    try {
      const file = createReadStream(
        join(process.cwd(), `./projects/${fileId}.pdf`),
      );

      return new StreamableFile(file);
    } catch (e) {
      console.log(e);
    }
  }
}
```

package.json:

```
{
  "name": "graynode",
  "version": "0.0.1",
  "description": "",
  "author": "",
  "private": true,
  "license": "UNLICENSED",
  "scripts": {
    "prebuild": "rimraf dist",
    "build": "nest build",
    "format": "prettier --write \"src/**/*.ts\" \"test/**/*.ts\"",
    "start": "nest start",
    "start:dev": "nest start --watch",
    "start:debug": "nest start --debug --watch",
```

```
  "start:prod": "node dist/main",
  "start:docker": "docker-compose up",
  "lint": "eslint \"{src,apps,libs,test}/**/*.ts\" --fix",
  "test": "jest",
  "test:watch": "jest --watch",
  "test:cov": "jest --coverage",
  "test:debug": "node --inspect-brk -r tsconfig-paths/register -r ts-node/register node_modules/.bin/jest --runInBand",
  "test:e2e": "jest --config ./test/jest-e2e.json",
  "seed": "npm run build && node dist/modules/seeder/seeder.script.js"
},
"dependencies": {
  "@nestjs/common": "^9.0.0",
  "@nestjs/config": "^3.0.0",
  "@nestjs/core": "^9.0.0",
  "@nestjs/jwt": "^10.1.0",
  "@nestjs/passport": "^10.0.0",
  "@nestjs/platform-express": "^9.0.0",
  "@nestjs/swagger": "^7.1.2",
  "@nestjs/typeorm": "^10.0.0",
  "argon2": "^0.30.3",
  "class-transformer": "^0.5.1",
  "class-validator": "^0.14.0",
  "cookie-parser": "^1.4.6",
  "date-fns": "^2.30.0",
  "ncsrf": "^1.0.10",
  "passport": "^0.6.0",
  "passport-jwt": "^4.0.1",
  "pg": "^8.11.1",
  "reflect-metadata": "^0.1.13",
  "rimraf": "^3.0.2",
  "rxjs": "^7.2.0",
  "typeorm": "^0.3.17"
},
"devDependencies": {
  "@nestjs/cli": "^9.0.0",
  "@nestjs/schematics": "^9.0.0",
  "@nestjs/testing": "^9.0.0",
  "@types/cookie-parser": "^1.4.3",
  "@types/date-fns": "^2.6.0",
  "@types/express": "^4.17.13",
  "@types/jest": "28.1.8",
  "@types/multer": "^1.4.11",
  "@types/node": "^16.0.0",
  "@types/passport-jwt": "^3.0.9",
  "@types/supertest": "^2.0.11",
  "@typescript-eslint/eslint-plugin": "^5.0.0",
  "@typescript-eslint/parser": "^5.0.0",
  "eslint": "^8.0.1",
  "eslint-config-prettier": "^8.3.0",
  "eslint-plugin-prettier": "^4.0.0",
  "jest": "28.1.3",
  "prettier": "^2.3.2",
  "source-map-support": "^0.5.20",
  "supertest": "^6.1.3",
  "ts-jest": "28.0.8",
  "ts-loader": "^9.2.3",
  "ts-node": "^10.0.0",
  "tsconfig-paths": "4.1.0",
  "typescript": "^4.7.4"
},
"jest": {
  "moduleFileExtensions": [
    "js",
    "json",
    "ts"
  ],
  "rootDir": "src",
  "testRegex": ".*\\.spec\\.ts$",
  "transform": {
    "^.+\\.(t|j)s$": "ts-jest"
  },
  "collectCoverageFrom": [
    "**/*.(t|j)s"
  ],
  "coverageDirectory": "../coverage",
  "testEnvironment": "node"
}
```

ДОДАТОК В

Текст програми клієнтської частини

index.tsx:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { ChakraProvider } from '@chakra-ui/react';

const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement,
);

root.render(
  <ChakraProvider>
    <App />
  </ChakraProvider>,
);
```

consts.ts:

```
const LocalStorageConsts = Object.freeze({
  accessToken: 'accessToken',
});

const ApiUrlsConsts = Object.freeze({
  login: 'auth/signIn',
  register: 'auth/signUp',
  refreshTokens: 'auth/refresh',
  logout: 'auth/logout',
  updateUser: 'auth/userProfile',
  userProfile: 'users/userProfile',
  getPosts: 'files',
  addPost: 'files/uploadProject',
  getGenres: 'genres',
});

const DefaultConsts = Object.freeze({
  itemsOnPage: 15,
});

export { LocalStorageConsts, ApiUrlsConsts, DefaultConsts };
```

App.tsx:

```
import React, { useEffect } from 'react';
import { BrowserRouter } from 'react-router-dom';
import Nav from './components/Nav';
import AuthStore from './store/auth.store';
import GlobalStore from './store/global.store';
import LoadingSpinner from './components/LoadingSpinner';
import { observer } from 'mobx-react';
import Routes from './routes/Routes';

const App = observer(() => {
  const initializeApp = async () => {
    try {
      await AuthStore.getUser();
    } catch (error) {
      console.log(error);
    } finally {
      setTimeout(() => {
        GlobalStore.toggleIsLoaded();
      }, 1000);
    }
  };

  useEffect(() => {
    initializeApp();
  }, []);
```

```
  return GlobalStore.isLoaded ? (
    <div className="App">
      <BrowserRouter>
        <Nav />
        <Routes />
      </BrowserRouter>
    </div>
  ) : (
    <LoadingSpinner />
  );
});

export default App;
```

AutoResizeTextarea.tsx:

```
import ResizeTextarea from 'react-textarea-autosize';
import React, { FunctionComponent } from 'react';
import { Textarea, TextareaProps } from '@chakra-ui/react';

const AutoResizeTextarea: FunctionComponent<TextareaProps> = (
  props: TextareaProps,
) => {
  return (
    <Textarea
      minH="unset"
      overflow="hidden"
      w="100%"
      resize="none"
      minRows={1}
      as={ResizeTextarea}
      {...props}
    />
  );
};

export default AutoResizeTextarea;
```

FormInput.tsx:

```
import React, { Dispatch, FunctionComponent, useState } from 'react';
import {
  Button,
  FormControl,
  FormLabel,
  Input,
  InputGroup,
  InputRightElement,
} from '@chakra-ui/react';
import { ViewIcon, ViewOffIcon } from '@chakra-ui/icons';

interface Props {
  id: string;
  isRequired?: boolean;
  text: string;
  type: string;
  setText: Dispatch<string>;
  addViewIcon?: boolean;
}

const FormInput: FunctionComponent<Props> = ({
  text,
  id,
  isRequired = false,
  type,
  setText,
  addViewIcon = false,
}: Props) => {
  const [showPassword, setShowPassword] = useState<boolean>(false);

  return (
    <FormControl id={id} isRequired={isRequired}>
      <FormLabel>{text}</FormLabel>
      <InputGroup>
        <Input
          type={addViewIcon ? (showPassword ? 'text' : 'password') : type}
          onChange={(text) => setText(text.target.value)}
```

```
        />
        {addViewIcon ? (
          <InputRightElement h={'full'}>
            <Button
              variant={'ghost'}
              onClick={() => setShowPassword((showPassword) => !showPassword)}
            >
              {showPassword ? <ViewIcon /> : <ViewOffIcon />}
            </Button>
          </InputRightElement>
        ) : null}
      </InputGroup>
    </FormControl>
  );
};

export default FormInput;
```

LoadingSpinner.tsx:

```
import React, { FunctionComponent } from 'react';
import { Spinner } from '@chakra-ui/react';

const LoadingSpinner: FunctionComponent = () => {
  return (
    <Spinner
      thickness="4px"
      speed="0.65s"
      emptyColor="gray.200"
      color="blue.500"
      size="xl"
      position="fixed"
      top="50%"
      left="50%"
      transform="translate(-50%, -50%)"
    />
  );
};

export default LoadingSpinner;
```

Nav.tsx:

```
import React, { FunctionComponent } from 'react';
import {
  Box,
  Button,
  Flex,
  Stack,
  useColorMode,
  useColorModeValue,
} from '@chakra-ui/react';
import { MoonIcon, SunIcon } from '@chakra-ui/icons';
import SignInButton from './SignInButton';
import NavLink from './NavLink';
import ProfileCircle from './ProfileCircle';
import { observer } from 'mobx-react';
import AuthStore from '../store/auth.store';
import { isNotEmptyString } from '../common/helpers/data.helper';

const Nav: FunctionComponent = observer(() => {
  const { toggleColorMode } = useColorMode();

  return (
    <Box bg={useColorModeValue('gray.100', 'gray.900')} px={4}>
      <Flex h={16} alignItems={'center'} justifyContent={'space-between'}>
        <Box>
          <NavLink link={'/'}>Home</NavLink>
          <NavLink link={'/files'}>Files</NavLink>
        </Box>

        <Flex alignItems={'center'}>
          <Stack direction={'row'} spacing={7}>
            <Button onClick={toggleColorMode}>
              {useColorModeValue(<MoonIcon />, <SunIcon />)}
            </Button>
            {isNotEmptyString(AuthStore.login) ? (
```

```
            <ProfileCircle username={AuthStore.login} />
          ) : (
            <SignInButton />
          )}
        </Stack>
      </Flex>
    </Flex>
  </Box>
 );
});

export default Nav;
```

NavLink.tsx:

```
import { Link as ReactRouterLink } from 'react-router-dom';
import React, { FunctionComponent } from 'react';
import { Link, useColorModeValue } from '@chakra-ui/react';

interface Props {
 children: React.ReactNode;
 link: string;
}

const NavLink: FunctionComponent<Props> = (props: Props) => {
 const { children, link } = props;

 return (
  <Link
   as={ReactRouterLink}
   px={2}
   py={1}
   rounded={'md'}
   _hover={{
    textDecoration: 'none',
    bg: useColorModeValue('gray.200', 'gray.700'),
   }}
   to={link}
  >
   {children}
  </Link>
 );
};

export default NavLink;
```

Post.tsx:

```
import React, { FunctionComponent } from 'react';
import {
 Avatar,
 Button,
 Flex,
 Stack,
 Text,
 useColorModeValue,
} from '@chakra-ui/react';
import PostStore from '../store/post.store';

interface Props {
 id: string;
 username: string;
 topic: string;
}

const Post: FunctionComponent<Props> = ({ id, username, topic }: Props) => {
 return (
  <Stack
   p="4"
   boxShadow="lg"
   m="4"
   borderRadius="sm"
   bg={useColorModeValue('gray.50', 'gray.700')}
  >
   <Flex gap="10px" alignItems="center">
    <Avatar size={'sm'} name={username} />
    <Text lineHeight={1.1} fontSize={{ base: 'xl', sm: '2xl' }}>
```

```
          {username}
        </Text>
      </Flex>
      <Stack direction="row" alignItems="center">
        <Text fontWeight="semibold">{topic}</Text>
      </Stack>
      <Stack direction={'row'} gap={'20px'}>
        <Button
          onClick={() => {
            // @ts-ignore
            const blob = new Blob([PostStore.file], {
              type: 'application/pdf',
            });
            const url = window.URL.createObjectURL(blob);
            const a = document.createElement('a');
            a.href = url;
            a.download = 'Test.zip';
            document.body.appendChild(a);
            a.click();
            document.body.removeChild(a);
            window.URL.revokeObjectURL(url);
          }}
        >
          Download project{' '}
        </Button>
        <Button>Share project</Button>
        <Button>Delete project</Button>
      </Stack>
    </Stack>
  );
};

export default Post;
```

PostInput.tsx:

```
import React, {
  ChangeEvent,
  FunctionComponent,
  useEffect,
  useState,
} from 'react';
import {
  Button,
  Center,
  FormControl,
  FormLabel,
  Input,
  Stack,
  useColorModeValue,
  useToast,
} from '@chakra-ui/react';
import { observer } from 'mobx-react';
import AuthStore from '../store/auth.store';
import { isEmptyString, isNotEmptyString } from '../common/helpers/data.helper';
import PostStore from '../store/post.store';

const PostInput: FunctionComponent = observer(() => {
  const toast = useToast();
  const [topic, setTopic] = useState<string>('');
  const [file, setFile] = useState<Blob | null>(null);
  const [isDisabledSubmitButton, setIsDisabledSubmitButton] =
    useState<boolean>(true);

  useEffect(() => {
    if (!isDisabledSubmitButton) {
      setIsDisabledSubmitButton(true);
    }
    if (isNotEmptyString(topic)) {
      setIsDisabledSubmitButton(false);
    }
  }, [topic]);

  const onChangeTopic = (e: ChangeEvent<HTMLInputElement>) => {
    setTopic(e.target.value);
  };
```

```
  const onClickSubmit = async () => {
    await PostStore.addPost({
      file,
      toast,
    });
  };

  return (
    <Stack
      p="4"
      boxShadow="lg"
      m="4"
      borderRadius="sm"
      bg={useColorModeValue('gray.50', 'gray.700')}
    >
      <Center lineHeight={1.1} fontSize={{ base: '3xl' }} padding={'10px'}>
        Add new project
      </Center>
      <Stack direction={['column', 'row']} spacing={6}>
        <Center w={'full'}>
          <FormControl p={4} textAlign={'center'}>
            <FormLabel textAlign={'center'}>Project name</FormLabel>
            <Input
              onChange={onChangeTopic}
              isDisabled={isEmptyString(AuthStore.login)}
            />
          </FormControl>
        </Center>
      </Stack>
      <Stack direction={['column', 'row']} spacing={6}>
        <Center w={'full'}>
          <FormControl p={4} textAlign={'center'}>
            <FormLabel textAlign={'center'}>Project files</FormLabel>
            <Input
              type={'file'}
              onChange={(e) => {
                if (e && e.currentTarget && e.currentTarget.files) {
                  setFile(e.currentTarget.files[0] as Blob);
                }
              }}
              isDisabled={isEmptyString(AuthStore.login)}
            />
          </FormControl>
        </Center>
      </Stack>
      <Button m={4} onClick={onClickSubmit} isDisabled={isDisabledSubmitButton}>
        Submit
      </Button>
    </Stack>
  );
});

export default PostInput;
```

ProfileButton.tsx:

```
import React, { FunctionComponent } from 'react';
import { Button, useColorModeValue } from '@chakra-ui/react';
import { raiseColor } from '../helpers/colors.helper';

interface Props {
  text: string;
  lightColor: string;
  darkColor: string;
  isDisabled?: boolean;
  onClick: () => Promise<void> | void;
}

const ProfileButton: FunctionComponent<Props> = ({
  text,
  lightColor,
  darkColor,
  isDisabled = false,
  onClick,
}: Props) => {
  return (
    <Button
```

```
      bg={useColorModeValue(lightColor, darkColor)}
      color={'white'}
      w="full"
      isDisabled={isDisabled}
      _hover={{
        bg: useColorModeValue(raiseColor(lightColor), raiseColor(darkColor)),
      }}
      onClick={onClick}
    >
      {text}
    </Button>
  );
};

export default ProfileButton;
```

ProfileCircle.tsx:

```
import {
  Avatar,
  Button,
  Center,
  Menu,
  MenuButton,
  MenuDivider,
  MenuItem,
  MenuList,
  useToast,
} from '@chakra-ui/react';
import React, { FunctionComponent } from 'react';
import { Link as ReactRouterLink } from 'react-router-dom';
import AuthStore from '../store/auth.store';

interface Props {
  username: string;
}

const ProfileCircle: FunctionComponent<Props> = ({ username }: Props) => {
  const toast = useToast();

  return (
    <Menu>
      <MenuButton
        as={Button}
        rounded={'full'}
        variant={'link'}
        cursor={'pointer'}
        minW={0}
      >
        <Avatar size={'sm'} name={username} />
      </MenuButton>
      <MenuList alignItems={'center'}>
        <br />
        <Center>
          <Avatar size={'2xl'} name={username} />
        </Center>
        <br />
        <Center>
          <p>{username}</p>
        </Center>
        <br />
        <MenuDivider />
        <MenuItem as={ReactRouterLink} to={'/userProfile'}>
          Profile
        </MenuItem>
        <MenuItem
          as={ReactRouterLink}
          to={'/'}
          onClick={async () => await AuthStore.logout({ toast })}
        >
          Logout
        </MenuItem>
      </MenuList>
    </Menu>
  );
};
```

```
export default ProfileCircle;
```

ProtectedRoute.tsx:

```
import React, { FunctionComponent, ReactElement } from 'react';
import { Navigate } from 'react-router-dom';
import AuthStore from '../store/auth.store';
import { observer } from 'mobx-react';
import { isEmptyString, isNotEmptyString } from '../common/helpers/data.helper';

interface Props {
  children: ReactElement;
  isAuthorizedRoute?: boolean;
}

const ProtectedRoute: FunctionComponent<Props> = observer(
  ({ children, isAuthorizedRoute = true }: Props) => {
    if (
      (!isAuthorizedRoute && isNotEmptyString(AuthStore.login)) ||
      (isAuthorizedRoute && isEmptyString(AuthStore.login))
    ) {
      return <Navigate to="/" replace />;
    }
    return children;
  },
);

export default ProtectedRoute;
```

SignButton.tsx:

```
import React, { FunctionComponent } from 'react';
import { Button, Stack, useColorModeValue } from '@chakra-ui/react';

interface Props {
  onClick?: () => Promise<void>;
  isDisabled?: boolean;
  text: string;
}

const SignButton: FunctionComponent<Props> = ({
  onClick,
  isDisabled = false,
  text,
}: Props) => {
  return (
    <Stack spacing={10} pt={2}>
      <Button
        loadingText="Submitting"
        size="lg"
        bg={useColorModeValue('blue.400', 'gray.800')}
        color={'white'}
        onClick={onClick}
        isDisabled={isDisabled}
        _hover={{
          bg: useColorModeValue('blue.500', 'gray.900'),
        }}
      >
        {text}
      </Button>
    </Stack>
  );
};

export default SignButton;
```

SignInButton.tsx:

```
import { Button, Stack, useColorModeValue } from '@chakra-ui/react';
import { Link as ReactRouterLink } from 'react-router-dom';
import React, { FunctionComponent } from 'react';

const SignInButton: FunctionComponent = () => {
  return (
    <Stack
      flex={{ base: 1, md: 0 }}
      justify={'flex-end'}
```

```
      direction={'row'}
      spacing={6}
    >
      <Button
        as={ReactRouterLink}
        display={{ base: 'flex', md: 'inline-flex' }}
        fontSize={'sm'}
        fontWeight={600}
        color={'white'}
        bg={useColorModeValue('blue.400', 'gray.700')}
        to={'/signIn'}
        _hover={{
          bg: useColorModeValue('blue.500', 'gray.800'),
        }}
      >
        Sign In
      </Button>
    </Stack>
  );
};

export default SignInButton;
```

StatIcon.tsx:

```
import React, { FunctionComponent, ReactElement } from 'react';
import { Tag, TagLabel } from '@chakra-ui/react';

interface Props {
  icon: ReactElement;
  amount: number;
}

const StatIcon: FunctionComponent<Props> = ({ icon, amount }: Props) => {
  return (
    <Tag
      size="sm"
      key="likes"
      borderRadius="full"
      variant="solid"
      gap="4px"
      alignItems="center"
      colorScheme="gray.800"
    >
      {icon}
      <TagLabel>{amount}</TagLabel>
    </Tag>
  );
};

export default StatIcon;
```

Routes.tsx:

```
import React, { FunctionComponent } from 'react';
import { Route, Routes as ReactRoutes } from 'react-router-dom';
import Home from '../pages/Home';
import ProtectedRoute from '../components/ProtectedRoute';
import Login from '../pages/Login';
import Register from '../pages/Register';
import UserProfile from '../pages/UserProfile';
import Posts from '../pages/Posts';

const Routes: FunctionComponent = () => {
  return (
    <ReactRoutes>
      <Route path="/" element={<Home />} />
      <Route
        path="/signIn"
        element={
          <ProtectedRoute isAuthorizedRoute={false}>
            <Login />
          </ProtectedRoute>
        }
      />
      <Route
        path="/signUp"
```

```
        element={
          <ProtectedRoute isAuthorizedRoute={false}>
            <Register />
          </ProtectedRoute>
        }
      />
      <Route
        path="/userProfile"
        element={
          <ProtectedRoute>
            <UserProfile />
          </ProtectedRoute>
        }
      />
      <Route path="/files" element={<Posts />} />
      <Route path="/posts/:id" element={<Posts />} />
    </ReactRoutes>
  );
};

export default Routes;
```

auth.service.ts:

```
import client from './axios.instance';
import type { CreateToastFnReturn } from '@chakra-ui/react';
import { ApiUrlsConsts } from '../consts';

class AuthService {
  async register(login: string, mnemonicPhrase: string, symmetricKey: string): Promise<boolean> {
    try {
      await client.post(ApiUrlsConsts.register, {
        login,
        mnemonicPhrase,
        symmetricKey
      });
      return true;
    } catch (error) {
      return false;
    }
  }

  async login(
    login: string,
    symmetricKey: string,
    isRemember: boolean,
  ): Promise<string | null> {
    try {
      const {
        data: { accessToken },
      } = await client.post(ApiUrlsConsts.login, {
        login,
        symmetricKey,
        isRemember,
      });
      return accessToken;
    } catch (error) {
      return null;
    }
  }

  async updateUser(data: {
    login?: string;
    password?: string;
  }): Promise<boolean> {
    try {
      await client.put(ApiUrlsConsts.updateUser, data);
      return true;
    } catch (error) {
      return false;
    }
  }

  async refreshToken(): Promise<string | undefined> {
    const {
      data: { accessToken },
    } = await client.get(ApiUrlsConsts.refreshTokens);
```

```
    return accessToken;
  }

  async getUser(): Promise<null | string> {
    try {
      const { data } = await client.get(ApiUrlsConsts.userProfile);
      return data.login;
    } catch (error) {
      return null;
    }
  }

  async logout({ toast }: { toast: CreateToastFnReturn }): Promise<boolean> {
    try {
      await client.get(ApiUrlsConsts.logout);
      return true;
    } catch (error) {
      return false;
    }
  }
}

const authService = new AuthService();

export default authService;
```

axios.instance.ts:

```
import axios from 'axios';
import { ApiUrlsConsts, LocalStorageConsts } from '../consts';
import AuthStore from '../store/auth.store';

const ENV = process.env.NODE_ENV || 'development';

const client = axios.create({
  baseURL:
    ENV === 'development'
      ? process.env.REACT_APP_DEVELOPMENT_API_URL
      : process.env.REACT_APP_PRODUCTION_API_URL,
  timeout: 3000,
  withCredentials: true,
});

client.interceptors.request.use(
  (config) => {
    const accessToken = localStorage.getItem(LocalStorageConsts.accessToken);
    if (accessToken) {
      // @ts-ignore
      config.headers = {
        ...config.headers,
        Authorization: `Bearer ${accessToken}`,
      };
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  },
);

client.interceptors.response.use(
  (response) => {
    return response;
  },
  async function (error) {
    const originalRequest = error.config;
    if (
      error.response.status === 401 &&
      originalRequest.url !== ApiUrlsConsts.refreshTokens
    ) {
      try {
        await AuthStore.refreshToken();
      } catch {
        return Promise.reject(error);
      }
      return client(originalRequest);
    }
```

```
    return Promise.reject(error);
  },
);

export default client;
```

post.service.ts:

```
import client from './axios.instance';
import { ApiUrlsConsts } from '../consts';
import { Post } from '../common/types';

class PostService {
  async getPosts(): Promise<Blob | null> {
    const { data } = await client.get(ApiUrlsConsts.getPosts);
    return data;
  }

  async addPost({ file }: any): Promise<Post | undefined> {
    try {
      const formData = new FormData();
      formData.append('file', file);

      const { data } = await client.post(ApiUrlsConsts.addPost, formData);
      return data;
    } catch (error) {
      console.log(error);
    }
  }
}

const postService = new PostService();

export default postService;
```

auth.store.ts:

```
import { makeAutoObservable, runInAction } from 'mobx';
import AuthService from '../services/auth.service';
import { CreateToastFnReturn } from '@chakra-ui/react';
import { NavigateFunction } from 'react-router';
import { LocalStorageConsts } from '../consts';
import { isNotEmptyString } from '../common/helpers/data.helper';

interface LoginDto {
  login: string;
  symmetricKey: string;
  isRemember: boolean;
  toast: CreateToastFnReturn;
  navigate: NavigateFunction;
}

interface RegisterDto extends Omit<LoginDto, 'isRemember'> {
  mnemonicPhrase: string;
}

interface UpdateUserDto extends Omit<LoginDto, 'isRemember'> {}

class AuthStore {
  login: string = '';

  constructor() {
    makeAutoObservable(this);
  }

  async getUser(): Promise<void> {
    const login = await AuthService.getUser();
    if (login) {
      runInAction(() => {
        this.login = login;
      });
    }
  }

  async registerUser({
    login,
    mnemonicPhrase,
```

```
  symmetricKey,
  toast,
  navigate,
}: RegisterDto): Promise<void> {
  if (mnemonicPhrase.length < 8) {
    toast({
      title: 'Bad Mnemonic Phrase',
      description: 'Mnemonic Phrase must contain at least 12 words',
      status: 'error',
      duration: 9000,
      isClosable: true,
    });
    return;
  }


  const isSuccess = await AuthService.register(login, mnemonicPhrase, symmetricKey);
  if (isSuccess) {
    toast({
      title: 'Successful',
      description: 'Successfully registered',
      status: 'success',
      duration: 9000,
      isClosable: true,
    });
    navigate('/signIn');
    return;
  }
  toast({
    title: 'Error',
    description: 'Something wrong',
    status: 'error',
    duration: 9000,
    isClosable: true,
  });
  return;
}

async loginUser({
  login,
  symmetricKey,
  isRemember,
  toast,
  navigate,
}: LoginDto): Promise<void> {

  const accessToken = await AuthService.login(login, symmetricKey, isRemember);
  if (accessToken) {
    localStorage.setItem(LocalStorageConsts.accessToken, accessToken);
    toast({
      title: 'Successful',
      description: 'Successfully login',
      status: 'success',
      duration: 9000,
      isClosable: true,
    });
    this.setUsername(login);
    navigate('/');
    return;
  }

  toast({
    title: 'Error',
    description: 'Username or password are incorrect',
    status: 'error',
    duration: 9000,
    isClosable: true,
  });
  return;
}

async updateUser({
  login,
  symmetricKey,
  toast,
  navigate,
}: UpdateUserDto): Promise<void> {
```

```
    let data = {};
    if (isNotEmptyString(login)) {
      data = { ...data, login };
    }

    const isSuccess = await AuthService.updateUser(data);
    if (isSuccess) {
      toast({
        title: 'Successful',
        description: 'Successfully updated',
        status: 'success',
        duration: 9000,
        isClosable: true,
      });
      this.removeUsername();
      localStorage.removeItem(LocalStorageConsts.accessToken);
      navigate('/signIn', { replace: true });
      return;
    }
    toast({
      title: 'Error',
      description: 'Username is already exists',
      status: 'error',
      duration: 9000,
      isClosable: true,
    });
    return;
  }

  async refreshToken(): Promise<void> {
    const accessToken = await AuthService.refreshToken();
    if (accessToken) {
      localStorage.removeItem(LocalStorageConsts.accessToken);
      localStorage.setItem(LocalStorageConsts.accessToken, accessToken);
    }
  }

  async logout({ toast }: { toast: CreateToastFnReturn }) {
    const isSuccess = await AuthService.logout({ toast });
    if (isSuccess) {
      toast({
        title: 'Successful',
        description: 'Successfully logout',
        status: 'success',
        duration: 9000,
        isClosable: true,
      });
      this.removeUsername();
      localStorage.removeItem(LocalStorageConsts.accessToken);
      return;
    }
    toast({
      title: 'Error',
      description: 'Something wrong',
      status: 'error',
      duration: 9000,
      isClosable: true,
    });
    return;
  }

  setUsername(login: string): void {
    runInAction(() => {
      this.login = login;
    });
  }

  removeUsername(): void {
    runInAction(() => {
      this.login = '';
    });
  }
}

const authStore = new AuthStore();

export default authStore;
```

globa.store.ts:

```
import { makeAutoObservable, runInAction } from 'mobx';

class GlobalStore {
  isLoaded: boolean = false;

  constructor() {
    makeAutoObservable(this);
  }

  toggleIsLoaded() {
    runInAction(() => {
      this.isLoaded = !this.isLoaded;
    });
  }
}

const globalStore = new GlobalStore();

export default globalStore;
```

post.store.ts:

```
import { autorun, makeAutoObservable, runInAction } from 'mobx';
import PostService from '../services/post.service';
import { GenresForSelect, Page, Post } from '../common/types';
import { CreateToastFnReturn } from '@chakra-ui/react';
import { MultiValue } from 'chakra-react-select';

interface NewPost {
  topic: string;
  text: string;
  genres: MultiValue<GenresForSelect>;
}

class PostStore {
  postPage: Page<Post> | undefined = undefined;
  text: string = '';
  file: Blob | null = null;

  constructor() {
    makeAutoObservable(this);
  }

  async getPosts(): Promise<void> {
    const postPage = await PostService.getPosts();
    if (!postPage) return;
    runInAction(() => {
      this.text = 'Test';
      this.file = postPage;
    });
  }

  async addPost({
    file,
    toast,
  }: {
    file: Blob | null;
    toast: CreateToastFnReturn;
  }): Promise<void> {
    await PostService.addPost({
      file,
    });
    toast({
      title: 'Successful',
      description: 'Successfully added',
      status: 'success',
      duration: 9000,
      isClosable: true,
    });
    await this.getPosts();
  }
}

const postStore = new PostStore();
```

```
autorun(async () => {
  try {
    await postStore.getPosts();
  } catch (error) {
    console.log(error);
  }
});

export default postStore;
```

Home.tsx:

```
import React, { Fragment, FunctionComponent } from 'react';

const Home: FunctionComponent = () => {
  return <Fragment></Fragment>;
};

export default Home;
```

Login.tsx:

```
import React, { FunctionComponent, useEffect, useState } from 'react';
import {
  Box,
  Checkbox,
  Flex,
  Heading,
  Link,
  Stack,
  Text,
  useColorModeValue,
  useToast,
  Input, FormLabel, InputGroup, InputRightElement, Button, FormControl
} from '@chakra-ui/react';
import { useNavigate } from 'react-router';
import { Link as ReactRouterLink } from 'react-router-dom';
import FormInput from '../components/FormInput';
import SignButton from '../components/SignButton';
import AuthStore from '../store/auth.store';
import { isNotEmptyString } from '../common/helpers/data.helper';
import {ViewIcon, ViewOffIcon} from "@chakra-ui/icons";

const Login: FunctionComponent = () => {
  const toast = useToast();
  const navigate = useNavigate();
  const [login, setLogin] = useState<string>('');
  const [symmetricKey, setSymmetricKey] = useState<string>('');
  const [isRememberMe, setIsRememberMe] = useState<boolean>(false);
  const [isDisabledRegisterButton, setIsDisabledRegisterButton] =
    useState<boolean>(true);

  useEffect(() => {
    if (!isDisabledRegisterButton) {
      setIsDisabledRegisterButton(true);
    }
    if (isNotEmptyString(login)) {
      setIsDisabledRegisterButton(false);
    }
  }, [login]);

  return (
    <Flex
      minH={'92vh'}
      align={'center'}
      justify={'center'}
      bg={useColorModeValue('gray.50', 'gray.800')}
    >
      <Stack spacing={8} mx={'auto'} minW={'400'} maxW={'lg'} py={12} px={6}>
        <Stack align={'center'}>
          <Heading fontSize={'4xl'} textAlign={'center'}>
            Sign in ✌️
          </Heading>
        </Stack>
        <Box
          rounded={'lg'}
```

```
        bg={useColorModeValue('white', 'gray.700')}
        boxShadow={'lg'}
        p={8}
      >
        <Stack spacing={4}>
          <FormInput
            text="Login"
            id="login"
            type="text"
            isRequired={true}
            setText={setLogin}
          />
          <FormControl id={'3'} isRequired={true}>
            <FormLabel>Symmetric Key</FormLabel>
            <InputGroup>
              <Input type='file' onChange={(e) => {
                if (e && e.currentTarget && e.currentTarget.files) {
                  const key = e.currentTarget.files[0] as Blob;
                  const reader = new FileReader()
                  reader.onload = function(e) {
                    const { result } = reader;
                    if(result && typeof result === "string") setSymmetricKey(result);
                  }
                  reader.readAsText(key)
                }
              }}/>
            </InputGroup>
          </FormControl>
          <Checkbox
            onChange={() => setIsRememberMe(!isRememberMe)}
            checked={isRememberMe}
          >
            Remember me
          </Checkbox>
          <SignButton
            text="Sign In"
            isDisabled={isDisabledRegisterButton}
            onClick={async () => {
              await AuthStore.loginUser({
                login,
                symmetricKey,
                isRemember: isRememberMe,
                toast,
                navigate,
              });
            }}
          />
          <Stack pt={6}>
            <Text align={'center'}>
              New to Threads?{' '}
              <Link as={ReactRouterLink} color={'blue.400'} to={'/signUp'}>
                Sign Up
              </Link>
            </Text>
          </Stack>
        </Stack>
      </Box>
    </Stack>
  </Flex>
 );
};

export default Login;
```

Posts.tsx:

```
import React, { Fragment, FunctionComponent } from 'react';
import { observer } from 'mobx-react';
import PostStore from '../store/post.store';
import Post from '../components/Post';
import PostInput from '../components/PostInput';

const Posts: FunctionComponent = observer(() => {
 return (
   <Fragment>
     <PostInput />
     {!!PostStore.text && (
```

```
        <Post id={'0'} key={0} username={'Test'} topic={PostStore.text} />
      )}
    </Fragment>
  );
});

export default Posts;
```

Register.tsx:

```
import React, { FunctionComponent, useEffect, useState } from 'react';
import {
  Box,
  Flex,
  Heading,
  Link,
  Stack,
  Text,
  useColorModeValue,
  useToast,
  Button
} from '@chakra-ui/react';
import * as crypto from 'crypto';
import { useNavigate } from 'react-router';
import { Link as ReactRouterLink } from 'react-router-dom';
import FormInput from '../components/FormInput';
import SignButton from '../components/SignButton';
import AuthStore from '../store/auth.store';
import { isNotEmptyString } from '../common/helpers/data.helper';

const generateRandomString = (length: number) => {
  let result = '';
  const characters =
    'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
  const charactersLength = characters.length;
  for (let i = 0; i < length; i++) {
    result += characters.charAt(Math.floor(Math.random() * charactersLength));
  }
  return result;
};

const Register: FunctionComponent = () => {
  const toast = useToast();
  const navigate = useNavigate();
  const [login, setLogin] = useState<string>('');
  const [mnemonicPhrase, setMnemonicPhrase] = useState<string>('');
  const [symmetricKey, setSymmetricKey] = useState<string>('');
  const [isDisabledRegisterButton, setIsDisabledRegisterButton] =
    useState<boolean>(true);

  useEffect(() => {
    if (!isDisabledRegisterButton) {
      setIsDisabledRegisterButton(true);
    }
    if (
      isNotEmptyString(login) &&
      isNotEmptyString(mnemonicPhrase) &&
      isNotEmptyString(symmetricKey)
    ) {
      setIsDisabledRegisterButton(false);
    }
  }, [login, mnemonicPhrase, symmetricKey]);

  return (
    <Flex
      minH={'92vh'}
      align={'center'}
      justify={'center'}
      bg={useColorModeValue('gray.50', 'gray.800')}
    >
      <Stack spacing={8} mx={'auto'} minW={'400'} maxW={'lg'} py={12} px={6}>
        <Stack align={'center'}>
          <Heading fontSize={'4xl'} textAlign={'center'}>
            Sign up ✌️
          </Heading>
        </Stack>
        <Box
```

```
          rounded={'lg'}
          bg={useColorModeValue('white', 'gray.700')}
          boxShadow={'lg'}
          p={8}
        >
          <Stack spacing={4}>
            <FormInput
              text="Login"
              id="login"
              type="text"
              isRequired={true}
              setText={setLogin}
            />
            <FormInput
              text="MnemonicPhrase"
              id="MnemonicPhrase"
              type="text"
              isRequired={true}
              setText={setMnemonicPhrase}
            />
            <Button onClick={() => {
              const key = generateRandomString(64);
              setSymmetricKey(key);
              const blob = new Blob([key], { type: "text/plain" });
              const url = window.URL.createObjectURL(blob);
              const a = document.createElement('a');
              a.href = url;
              a.download = 'key.txt';
              document.body.appendChild(a);
              a.click();
              document.body.removeChild(a);
              window.URL.revokeObjectURL(url);
            }}>Generate Symmetric Key </Button>
            <SignButton
              text="Sign Up"
              isDisabled={isDisabledRegisterButton}
              onClick={async () => {
                await AuthStore.registerUser({
                  login,
                  mnemonicPhrase,
                  symmetricKey,
                  toast,
                  navigate,
                });
              }}
            />
            <Stack pt={6}>
              <Text align={'center'}>
                Already a user?{' '}
                <Link as={ReactRouterLink} color={'blue.400'} to={'/signIn'}>
                  Sign In
                </Link>
              </Text>
            </Stack>
          </Stack>
        </Box>
      </Stack>
    </Flex>
  );
};

export default Register;
```

UserProfile.tsx:

```
'use client';

import {
  Avatar,
  Center,
  Flex,
  FormControl,
  Heading,
  Stack,
  useColorModeValue,
  useToast,
} from '@chakra-ui/react';
```

```
import ProfileButton from '../components/ProfileButton';
import AuthStore from '../store/auth.store';
import FormInput from '../components/FormInput';
import React, { FunctionComponent, useEffect, useState } from 'react';
import { useNavigate } from 'react-router';
import { isNotEmptyString } from '../common/helpers/data.helper';

const UserProfile: FunctionComponent = () => {
  const toast = useToast();
  const navigate = useNavigate();
  const [login, setUsername] = useState<string>('');
  const [symmetricKey, setSymmetricKey] = useState<string>('');
  const [isDisabledSubmitButton, setIsDisabledSubmitButton] =
    useState<boolean>(true);

  useEffect(() => {
    if (!isDisabledSubmitButton) {
      setIsDisabledSubmitButton(true);
    }
    if (isNotEmptyString(login)) {
      setIsDisabledSubmitButton(false);
    }
  }, [login]);

  return (
    <Flex
      minH={'100vh'}
      align={'center'}
      justify={'center'}
      bg={useColorModeValue('gray.50', 'gray.800')}
    >
      <Stack
        spacing={4}
        w={'full'}
        maxW={'md'}
        bg={useColorModeValue('white', 'gray.700')}
        rounded={'xl'}
        boxShadow={'lg'}
        p={6}
        my={12}
      >
        <Heading lineHeight={1.1} fontSize={{ base: '2xl', sm: '3xl' }}>
          Profile
        </Heading>
        <FormControl id="userName">
          <Stack direction={['column', 'row']} spacing={6}>
            <Center>
              <Avatar size="2xl" name={AuthStore.login} />
            </Center>
            <Center w="full">
              <Heading lineHeight={1.1} fontSize={{ base: '2xl', sm: '3xl' }}>
                {AuthStore.login}
              </Heading>
            </Center>
          </Stack>
        </FormControl>
        <FormInput
          text="Username"
          id="login"
          type="text"
          setText={setUsername}
        />
        <FormInput
          text="Password"
          id="password"
          type="password"
          setText={setSymmetricKey}
          addViewIcon={true}
        />
        <Stack spacing={6} direction={['column', 'row']}>
          <ProfileButton
            text="Cancel"
            lightColor="red.400"
            darkColor="blackAlpha.400"
            onClick={() => navigate('/')}
          />
          <ProfileButton
```

```
            text="Submit"
            lightColor="blue.400"
            darkColor="gray.800"
            isDisabled={isDisabledSubmitButton}
            onClick={async () =>
              await AuthStore.updateUser({ login, symmetricKey, toast, navigate })
            }
          />
        </Stack>
      </Stack>
    </Flex>
  );
};

export default UserProfile;
```

package.json:

```
{
  "name": "threads.frontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@chakra-ui/icons": "^2.1.0",
    "@chakra-ui/react": "^2.8.0",
    "@emotion/react": "^11.11.1",
    "@emotion/styled": "^11.11.0",
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "@types/jest": "^27.5.2",
    "@types/node": "^16.18.39",
    "@types/react": "^18.2.18",
    "@types/react-dom": "^18.2.7",
    "axios": "^1.4.0",
    "chakra-react-select": "^4.7.0",
    "framer-motion": "^10.16.1",
    "mobx": "^6.10.0",
    "mobx-react": "^9.0.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router": "^6.15.0",
    "react-router-dom": "^6.15.0",
    "react-scripts": "^5.0.1",
    "react-textarea-autosize": "^8.5.3",
    "typescript": "^4.9.5",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start:dev": "NODE_ENV=development && react-scripts start",
    "start:dev:windows": "set NODE_ENV=development && react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "devDependencies": {
    "@types/axios": "^0.14.0",
    "prettier": "^3.0.0"
  }
```

ЗАТВЕРДЖУЮ

Директор ТОВ «СМІСС»

_____ Марина ВЕРТИКОВА

«_12_» _____грудня_____ 2023 р.

АКТ

про впровадження результатів магістерської кваліфікаційної роботи

Палія Олексія Миколайовича

Комісія у складі: голова комісії – директор Марина Вертикова, розглянувши матеріали магістерської роботи Олексія Палія на тему «Метод та засіб захищеного обміну корпоративною інформацією розробників програмного забезпечення», склала цей акт про те, що у ТОВ «СМІСС» впроваджено результати цієї роботи, а саме модель розмежування прав доступу.

Впроваджені результати дозволяють здійснювати захищений обмін корпоративною інформацією між розробниками, що дало змогу підвищити ефективність управління підприємством.

Голова комісії:

Директор _____ Марина ВЕРТИКОВА

Додаток Д

ПРОТОКОЛ ПЕРЕВІРКИ
МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Метод та засіб захищеного обміну корпоративною інформацією розробників програмного забезпечення

Автор роботи: Палій Олексій Миколайович

Тип роботи: магістерська кваліфікаційна робота

Підрозділ кафедра захисту інформації ФІТКІ
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність – 95,86%. Схожість – 4,14%.

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Валентина КАПЛУН
(підпис)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Олексій ПАЛІЙ
(підпис)

Керівник роботи _____ Юрій БАРИШЕВ
(підпис)

Додаток Е

ІЛЮСТРАТИВНА ЧАСТИНА

МЕТОД ТА ЗАСІБ ЗАХИЩЕНОГО ОБМІНУ КОРПОРАТИВНОЮ ІНОРФМАЦІЄЮ РОЗРОБНИКІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Актуальність, мета та задачі МКР

У СУЧАСНОМУ СВІТІ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ, ДЕ ОБСЯГ КОРПОРАТИВНОЇ ІНФОРМАЦІЇ ЗРОСТАЄ НА ЗАЧІПКУ, А ТЕХНОЛОГІЇ НАДАЮТЬ МОЖЛИВОСТІ ДЛЯ ЇЇ МИТТЄВОГО ОБМІНУ ТА ЗБЕРЕЖЕННЯ, ПИТАННЯ БЕЗПЕКИ ТА КОНФІДЕНЦІЙНОСТІ ІНФОРМАЦІЇ ВИЯВЛЯЮТЬСЯ НА ПЕРШОМУ ПЛАНІ. ОСОБЛИВО АКТУАЛЬНИМ СТАЄ ЗАВДАННЯ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ОБМІНУ КОРПОРАТИВНОЮ ІНФОРМАЦІЄЮ, ЩО СТОСУЄТЬСЯ НЕ ЛИШЕ ВЕЛИКИХ КОРПОРАЦІЙ ТА УРЯДОВИХ СТРУКТУР, АЛЕ Й НЕВЕЛИКИХ ПІДПРИЄМСТВ, А ТАКОЖ ІНДИВІДУАЛЬНИХ РОЗРОБНИКІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

СВІТОВІ ТЕНДЕНЦІЇ СВІДЧАТЬ ПРО ПОСТІЙНЕ ЗРОСТАННЯ КІЛЬКОСТІ ІНФОРМАЦІЇ, ЯКА ОБМІНЮЄТЬСЯ МІЖ ОРГАНІЗАЦІЯМИ ТА КОРИСТУВАЧАМИ. З ЦИМ ЗРОСТАННЯМ ОБСЯГІВ ІНФОРМАЦІЇ ЗБІЛЬШУЄТЬСЯ РИЗИК ЇЇ НЕЗАКОННОГО ДОСТУПУ, ПЕРЕХОПЛЕННЯ, ВИТОКУ АБО ПОШКОДЖЕННЯ. ЦЯ СИТУАЦІЯ СТАВИТЬ ПІДПРИЄМСТВА ТА РОЗРОБНИКІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПЕРЕД НАГАЛЬНОЮ НЕОБХІДНІСТЮ ЗАБЕЗПЕЧИТИ СТІЙКИЙ ЗАХИСТ ОБМІНУ КОРПОРАТИВНОЮ ІНФОРМАЦІЄЮ.

РОЗРОБНИКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СТАЮТЬ ЦЕНТРАЛЬНОЮ ПОСТАТТЮ В РЕАЛІЗАЦІЇ БЕЗПЕКИ ОБМІНУ ІНФОРМАЦІЄЮ. ВОНИ ВІДПОВІДАЮТЬ ЗА СТВОРЕННЯ ПРОГРАМ, ЯКІ ОБРОБЛЯЮТЬ, ПЕРЕДАЮТЬ ТА ЗБЕРІГАЮТЬ ВАЖЛИВІ ДАНІ. У СВОЇЙ РОБОТІ ВОНИ НЕ ТІЛЬКИ ПОВИННІ РОЗУМІТИ ТЕХНІЧНІ АСПЕКТИ БЕЗПЕКИ, АЛЕ І РОЗРОБЛЯТИ СТРАТЕГІЇ ТА ЗАХОДИ ДЛЯ ЕФЕКТИВНОГО ЗАХИСТУ КОРПОРАТИВНОЇ ІНФОРМАЦІЇ ВІД ЗОВНІШНІХ ЗАГРОЗ І ВНУТРІШНІХ ПОРУШЕНЬ БЕЗПЕКИ.

МЕТОЮ ДАНОЇ РОБОТИ Є ПОКРАЩЕННЯ ЗАХИСТУ ІНФОРМАЦІЇ У ХМАРНИХ СЕРВІСАХ, ШЛЯХОМ ВПРОВАДЖЕННЯ МЕХАНІЗМІВ ЗАХИСТУ ВІД НЕСАНКЦІОНОВАНОГО ДОСТУПУ ДО ЦИХ ДАНИХ.

ДЛЯ ДОСЯГНЕННЯ МЕТИ ПОТРІБНО РОЗВ'ЯЗАТИ ТАКІ ЗАДАЧІ:

-ПРОАНАЛІЗУВАТИ ПРОБЛЕМИ ХМАРНИХ СЕРЕДОВИЩ;

-СТВОРИТИ МОДЕЛЬ РОЗМЕЖУВАННЯ ПРАВ ДОСТУПУ;

-РОЗРОБИТИ АРХІТЕКТУРУ КЛІЄНТСЬКИХ ТА СЕРВЕРНИХ ЧАСТИН;

-РОЗРОБИТИ АЛГОРИТМИ МОДУЛЯ ШИФРУВАННЯ;

РОЗРОБИТИ АЛГОРИТМИ РОБОТИ ЗАСОБУ.

Наукова новизна та практична цінність

НАУКОВА НОВИЗНА РОБОТИ ПОЛЯГАЄ В ТАКОМУ:
-УДОСКОНАЛЕНО МЕТОД ЗАХИЩЕНОГО ОБМІНУ КОРПОРАТИВНОЮ ІНФОРМАЦІЄЮ РОЗРОБНИКІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ЯКИЙ НА ВІДМІНУ ВІД ВІДОМИХ ВИКОРИСТОВУЄ ПЕРЕДАВАННЯ ТА ЗБЕРІГАННЯ ВСІЄЇ ІНФОРМАЦІЇ НЕЗАЛЕЖНО ВІД ЇЇ ВИДУ В ЗАХИЩЕНОМУ ВИГЛЯДІ НА СТОРОНІ СЕРВЕРА, ЩО ДОЗВОЛИЛО ВРАХУВАТИ ВИРОБНИЧІ ПРОЦЕСИ РОЗРОБКИ ПРОГРАМИ, ЗБЕРІГАЮЧИ ЗАХИЩЕНІСТЬ КОРПОРАТИВНОЇ ІНФОРМАЦІЇ;
-ОТРИМАЛА ПОДАЛЬШИЙ РОЗВИТОК МОДЕЛЬ РОЗМЕЖУВАННЯ ПРАВ ДОСТУПУ, ЯКА НА ВІДМІННУ ВІД ВІДОМИХ, СТВОРЮЄ ПОПЕРЕДНІ НАЛАШТУВАННЯ ПРАВИЛ РОЗМЕЖУВАННЯ ПРАВ ДОСТУПУ У ВИГЛЯДІ РОЛЕЙ З ПОДАЛЬШИМ ВИКОРИСТАННЯМ МОДЕЛІ ГАРРІСОНА-РУЗЗО-УЛЬМАНА, ЩО ДОЗВОЛИЛО РОЗМЕЖОВУВАТИ ПРАВА ДОСТУПУ ДО ІНФОРМАЦІЙНИХ РЕСУРСІВ В МОМЕНТ ЇХ СТВОРЕННЯ.

ПРАКТИЧНА ЦІННІСТЬ ПРОГРАМИ ПОЛЯГАЄ В ТАКОМУ:
-РОЗРОБЛЕНО ПРОГРАМНИЙ ЗАСІБ ДЛЯ ЗАХИЩЕНОГО ОБМІНУ КОРПОРАТИВНОЮ ІНФОРМАЦІЄЮ;
-НАБОРИ СЦЕНАРІЇВ ДЛЯ БЛОКОВОГО ТА ІНТЕГРАЦІЙНОГО ТЕСТУВАННЯ ЗАСОБІВ ОБМІНУ КОРПОРАТИВНОЮ ІНФОРМАЦІЄЮ;
-МОДУЛЬ СТАТИЧНОГО ТЕСТУВАННЯ БЕЗПЕКИ ЗАСТОСУНКІВ, РОЗРОБЛЕНИХ МОВОЮ JAVASCRIPT.

Метод шифрування даних

КРОК 1. ГЕНЕРУВАННЯ КЛЮЧІВ СЕРВЕРОМ: ПЕРШИЙ КРОК ПОЛЯГАЄ В ТОМУ, ЩОБ СЕРВЕР ЗГЕНЕРУВАВ СИМЕТРИЧНИЙ КЛЮЧ, ЯКИЙ БУДЕ ВИКОРИСТОВУВАТИСЯ ДЛЯ ШИФРУВАННЯ ТА РОЗШИФРУВАННЯ ФАКТИЧНИХ ДАНИХ. ВАЖЛИВОЮ Є НАДІЙНІСТЬ ЦЬОГО КРОКУ, ОСКІЛЬКИ СИМЕТРИЧНИЙ КЛЮЧ БУДЕ ВИКОРИСТОВУВАТИСЯ ДЛЯ НАДІЙНОГО ШИФРУВАННЯ ІНФОРМАЦІЇ.
КРОК 2. ЗАШИФРУВАННЯ СИМЕТРИЧНОГО КЛЮЧА ПУБЛІЧНИМ КЛЮЧЕМ КОРИСТУВАЧА: СИМЕТРИЧНИЙ КЛЮЧ, ЗГЕНЕРОВАНИЙ СЕРВЕРОМ, ДАЛІ ШИФРУЄТЬСЯ ПУБЛІЧНИМ КЛЮЧЕМ КОРИСТУВАЧА. ЦЕ ЗАБЕЗПЕЧУЄ БЕЗПЕКУ ПЕРЕДАЧІ СИМЕТРИЧНОГО КЛЮЧА ЧЕРЕЗ НЕБЕЗПЕЧНІ КАНАЛИ, ОСКІЛЬКИ ЛИШЕ ПРИВАТНИЙ КЛЮЧ КОРИСТУВАЧА МОЖЕ РОЗШИФРУВАТИ ЦЕЙ КЛЮЧ. ЗА ДОПОМОГОЮ АСИМЕТРИЧНОГО ШИФРУВАННЯ, НАПРИКЛАД, RSA, СИМЕТРИЧНИЙ КЛЮЧ ЗАШИФРОВУЄТЬСЯ ТАК, ЩОБ ЛИШЕ ПРИВАТНИЙ КЛЮЧ КОРИСТУВАЧА МІГ РОЗШИФРУВАТИ ЙОГО. ЦЕЙ ПРОЦЕС МОЖНА ВІДОБРАЗИТИ У ФОРМУЛІ:

$$ENCRYPTEDKEY = ASYMMETRICENCRYPT(SYMMETRICKEY, PUBLICKEY), \quad (2.6)$$

ДЕ ENCRYPTEDKEY – ЗАШИФРОВАНИЙ СИМЕТРИЧНИЙ КЛЮЧ, ASYMMETRICENCRYPT – ФУНКЦІЯ ШИФРУВАННЯ, SYMMETRICKEY – СИМЕТРИЧНИЙ КЛЮЧ, ЩО БУВ ВИКОРИСТАНИЙ ДЛЯ ШИФРУВАННЯ ФАКТИЧНИХ ДАНИХ, PUBLICKEY – ПУБЛІЧНИЙ КЛЮЧ ОДЕРЖУВАЧА.
КРОК 3. ВІДПРАВЛЕННЯ ЗАШИФРОВАНОГО СИМЕТРИЧНОГО КЛЮЧА ТА ДАНИХ: ЗАШИФРОВАНИЙ СИМЕТРИЧНИЙ КЛЮЧ ТА ФАКТИЧНІ ДАНІ ВІДПРАВЛЯЮТЬСЯ НА СЕРВЕР. ЦЕ МОЖЕ ВІДБУВАТИСЯ ЧЕРЕЗ ЗАХИЩЕНІ КАНАЛИ ПЕРЕДАЧІ ДАНИХ, ТАКІ ЯК HTTPS, ДЛЯ ЗАБЕЗПЕЧЕННЯ ДОДАТКОВОЇ БЕЗПЕКИ.
КРОК 4. ЗБЕРЕЖЕННЯ СИМЕТРИЧНОГО КЛЮЧА НА СЕРВЕРІ: СЕРВЕР ЗБЕРІГАЄ ЗАШИФРОВАНИЙ СИМЕТРИЧНИЙ КЛЮЧ У БЕЗПЕЧНОМУ МІСЦІ, ДЕ ВІН ЗАЛИШАЄТЬСЯ ЗАХИЩЕНИМ ВІД НЕСАНКЦІОНОВАНОГО ДОСТУПУ.
КРОК 5. РОЗШИФРУВАННЯ СИМЕТРИЧНОГО КЛЮЧА НА КОМП'ЮТЕРІ КОРИСТУВАЧА: КОРИСТУВАЧ ВИКОРИСТОВУЄ СВІЙ ПРИВАТНИЙ КЛЮЧ ДЛЯ РОЗШИФРУВАННЯ СИМЕТРИЧНОГО КЛЮЧА НА СВОЄМУ КОМП'ЮТЕРІ. ЦЕЙ КРОК ГАРАНТУЄ, ЩО ЛИШЕ ВЛАСНИК ПРИВАТНОГО КЛЮЧА МОЖЕ ОТРИМАТИ ДОСТУП ДО СИМЕТРИЧНОГО КЛЮЧА. ЦЕЙ ПРОЦЕС МОЖНА ВІДОБРАЗИТИ У ФОРМУЛІ:

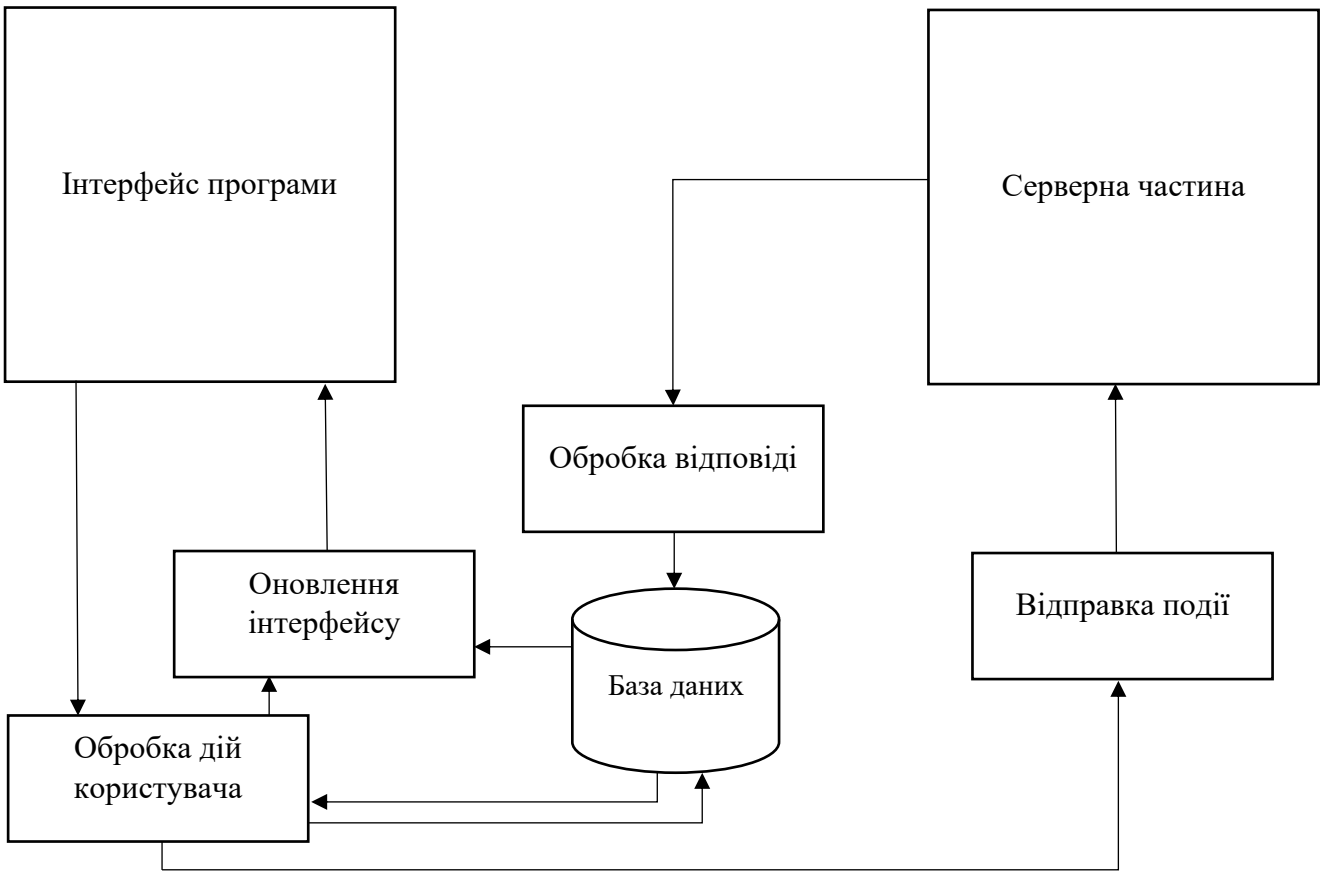
$$SYMMETRICKEY = ASYMMETRICDECRYPT(ENCRYPTEDKEY, PRIVATEKEY), \quad (2.7)$$

ДЕ ENCRYPTEDKEY – ЗАШИФРОВАНИЙ СИМЕТРИЧНИЙ КЛЮЧ, ASYMMETRICDECRYPT – ФУНКЦІЯ РОЗШИФРУВАННЯ, SYMMETRICKEY – СИМЕТРИЧНИЙ КЛЮЧ, PRIVATEKEY – ПРИВАТНИЙ КЛЮЧ ОДЕРЖУВАЧА.
КРОК 6. РОЗШИФРУВАННЯ ТА РОБОТА З ДАНИМИ: ОДЕРЖАВШИ СИМЕТРИЧНИЙ КЛЮЧ, КОРИСТУВАЧ МОЖЕ РОЗШИФРУВАТИ ФАКТИЧНІ ДАНІ І ПРАЦЮВАТИ З НИМИ НА СВОЄМУ КОМП'ЮТЕРІ. ЦЕ ЗАБЕЗПЕЧУЄ НАДІЙНИЙ ОБМІН І КОНФІДЕНЦІЙНІСТЬ ДАНИХ.
КРОК 7. ЗБЕРЕЖЕННЯ КЛЮЧІВ І ЗАХИСТ ВІД НЕСАНКЦІОНОВАНОГО ДОСТУПУ. ВАЖЛИВИМ Є ЗБЕРЕЖЕННЯ І ЗАХИСТ ПУБЛІЧНИХ ТА ПРИВАТНИХ КЛЮЧІВ КОРИСТУВАЧІВ І СЕРВЕРА ДЛЯ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ОБМІНУ ДАНИМИ.

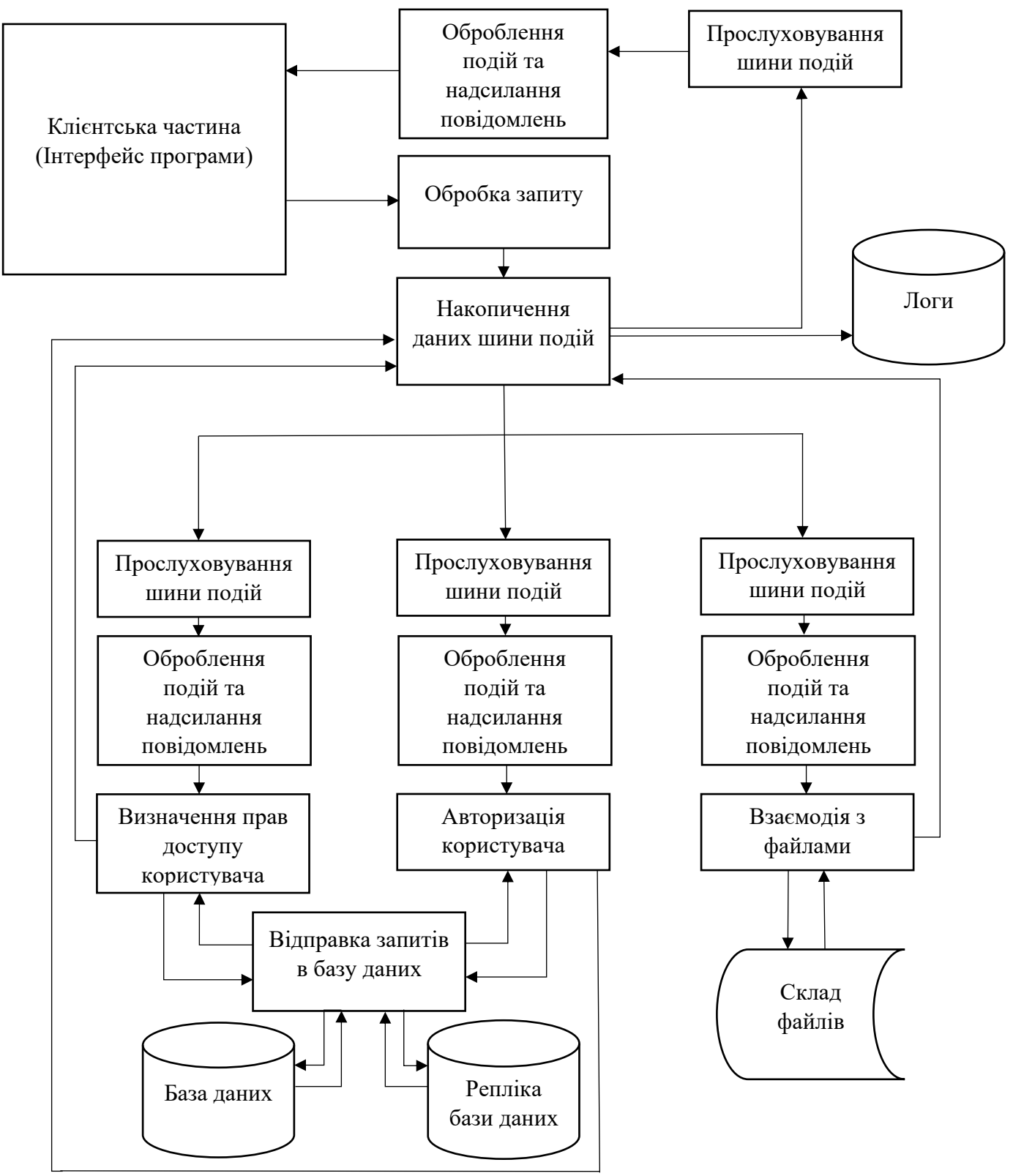
Модель розмежування прав доступу

Об'єкти / Суб'єкти	Сирий код	Проекти	Баг-репорти	Документи	Звіти з код-рев'ю
Розробник	R	R	-	-	-
HR	-	-	-	R	-
Tech Lead	R	R	R	-	R
QA	-	-	RW	-	-
Не Команда	ηRWX	ηRWX	-	-	-

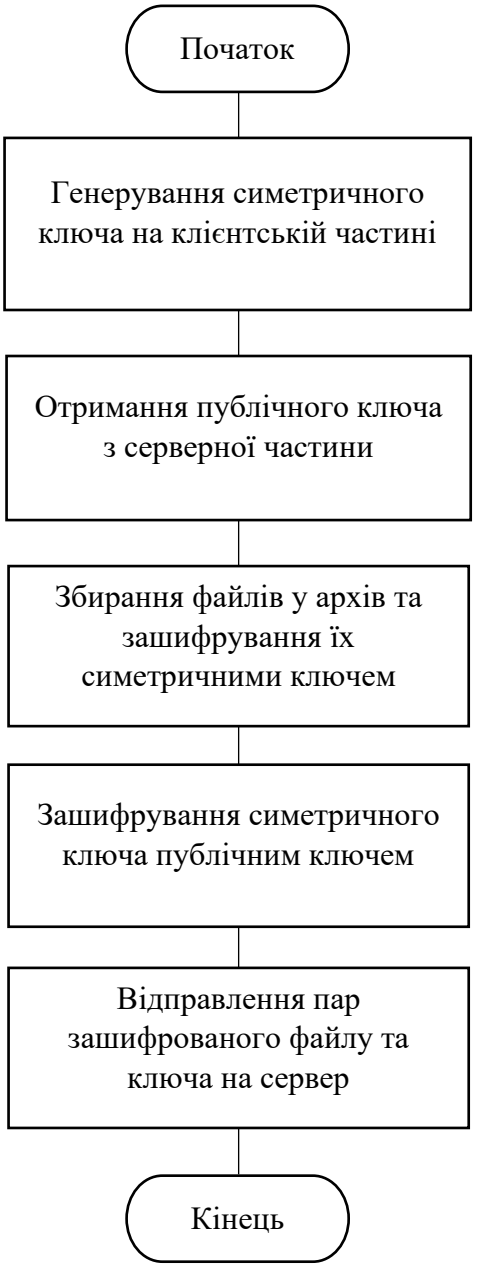
Архітектура клієнтської частини

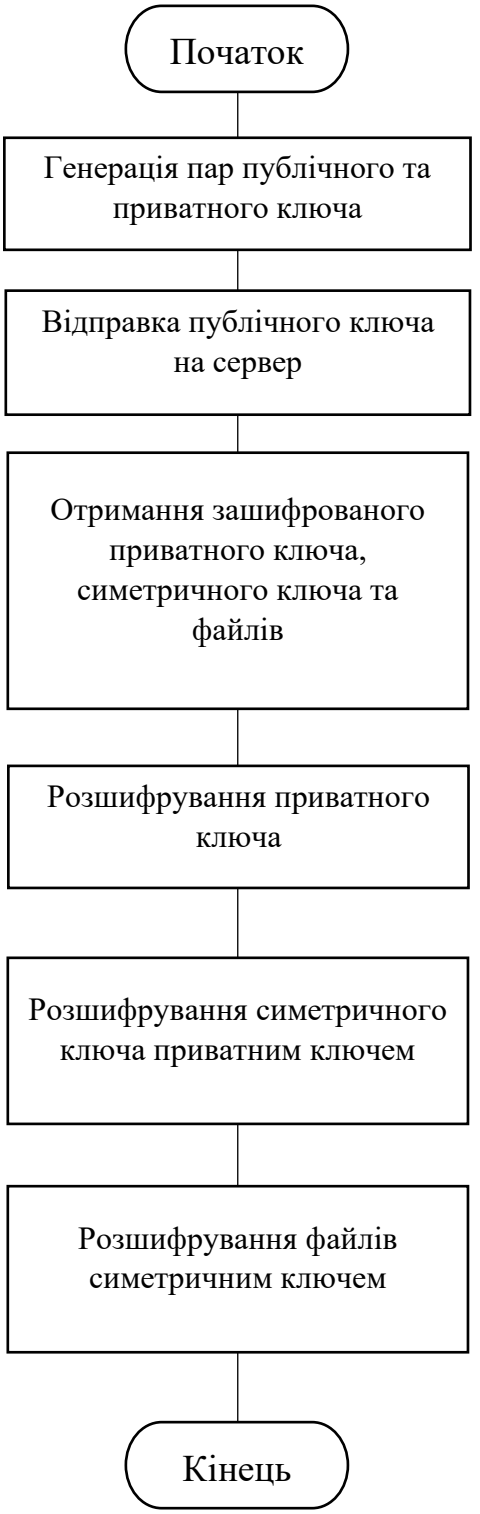


Архітектура серверної частини

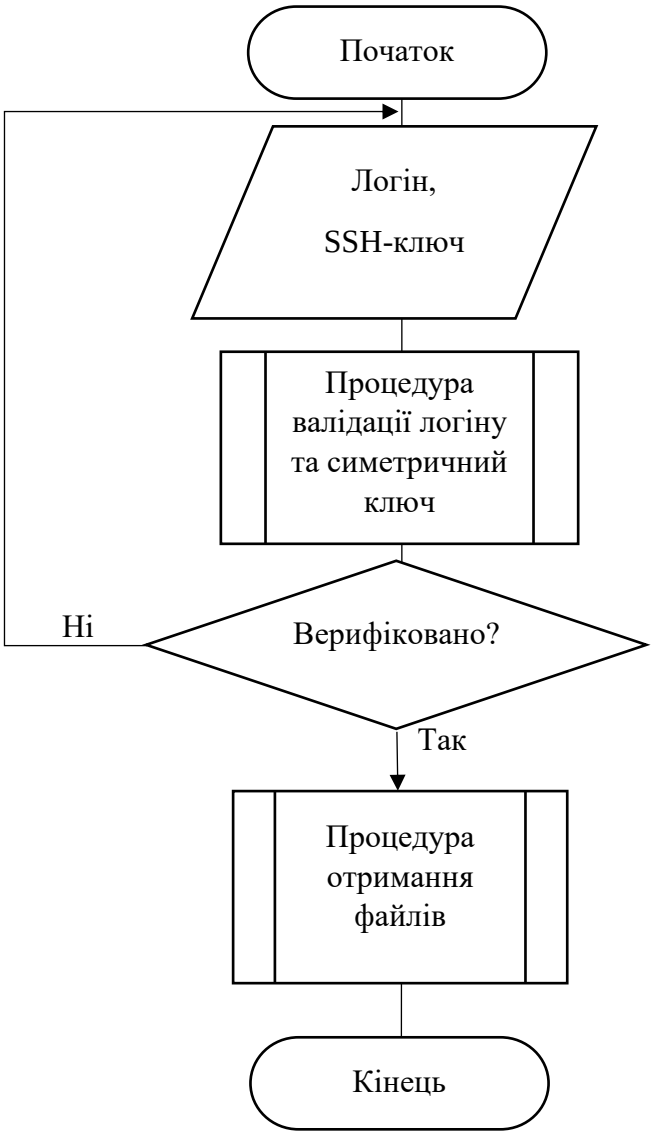


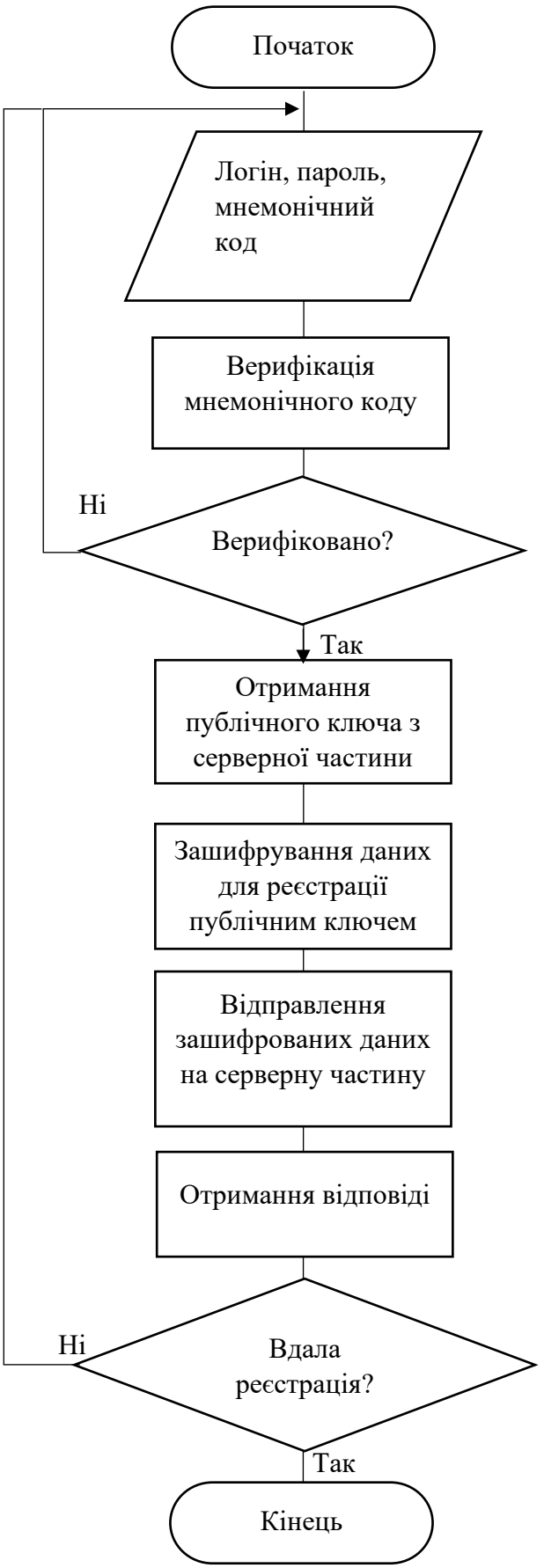
Алгоритми шифрування





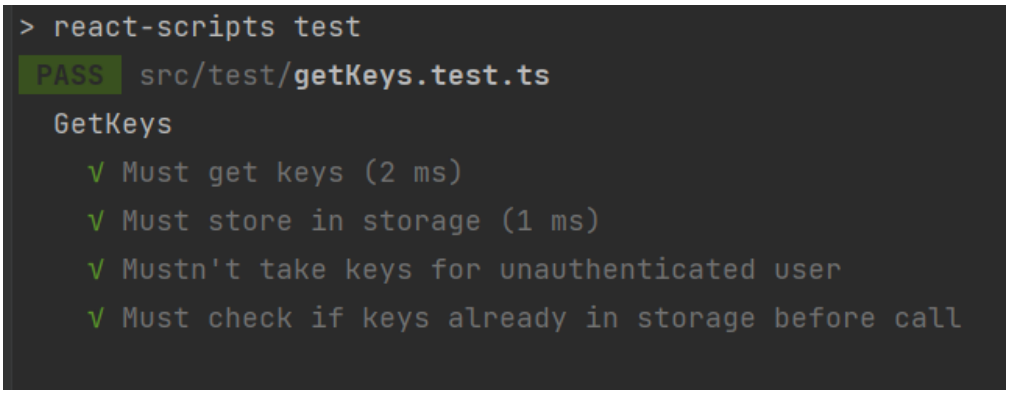
Алгоритми автентифікації/реєстрації



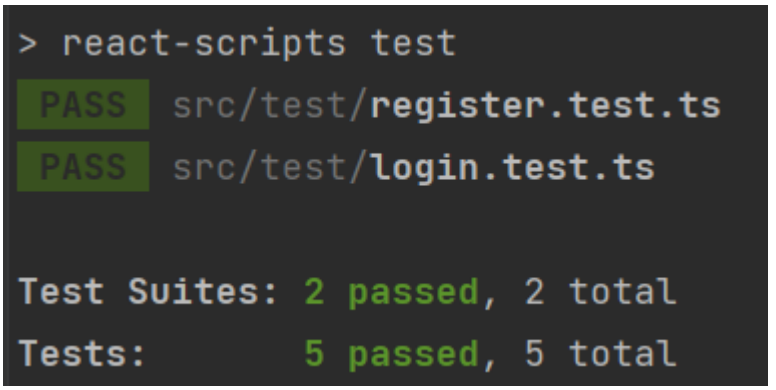


Результати блокового тестування

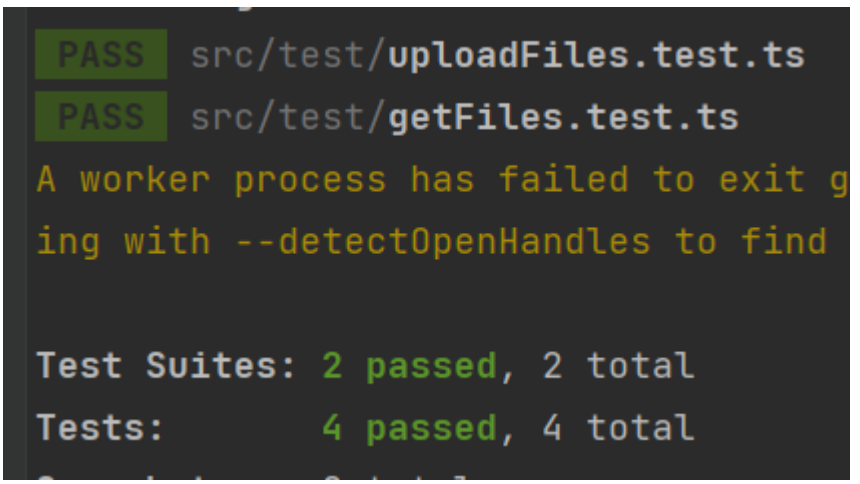
Результат тестування методу GetKeys



Результат тестування методів Login/Register

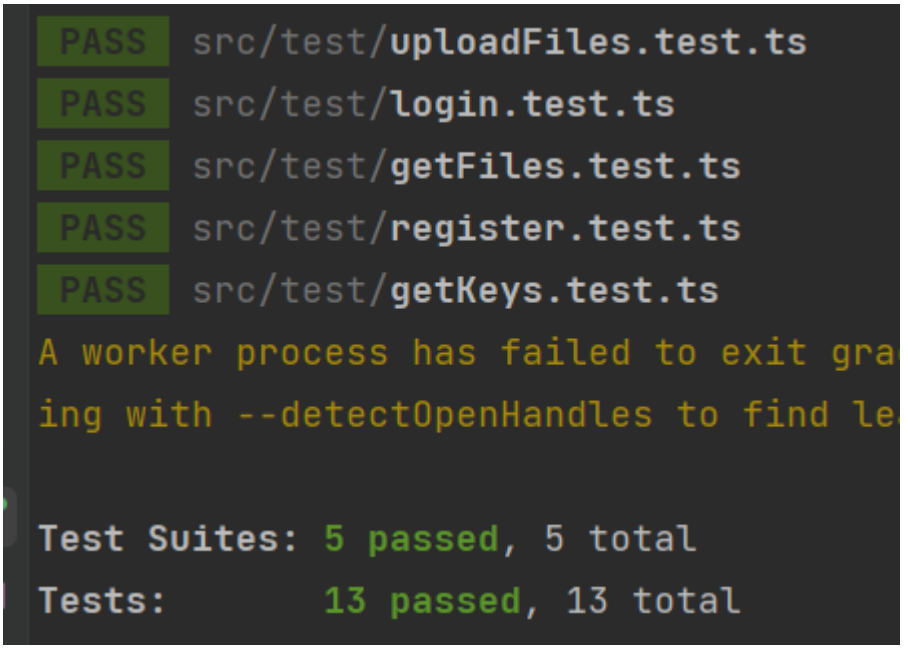


Результат тестування методу GetKeys

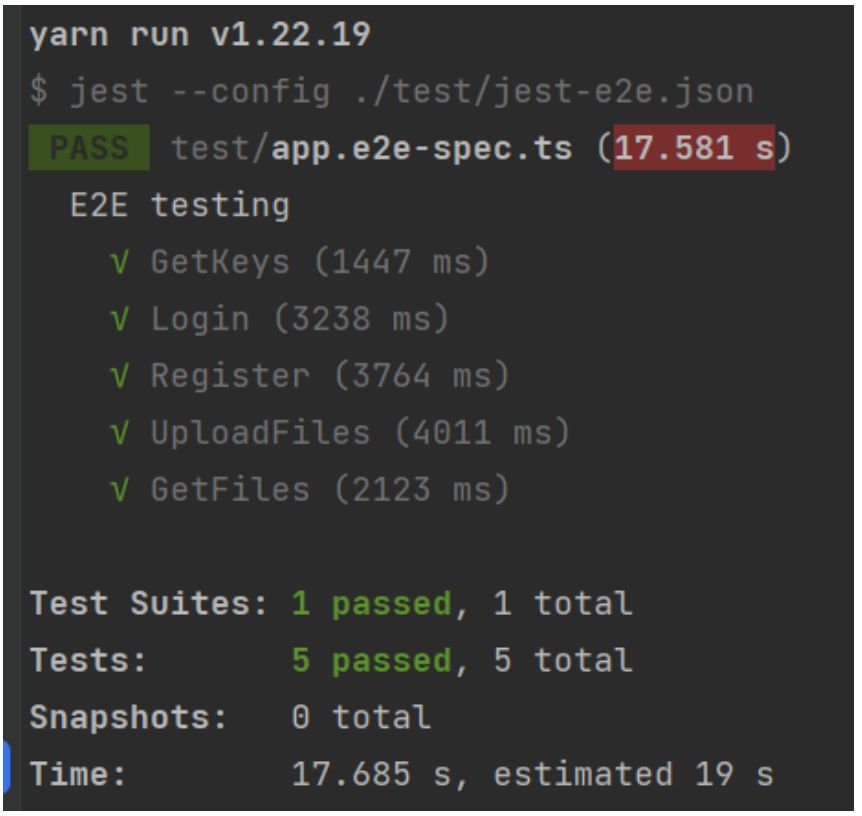


Результати блокового тестування

Результат тестування методів GetKeys, Login, Register, UploadFiles, GetFiles на серверній частині



Результати інтеграційного тестування

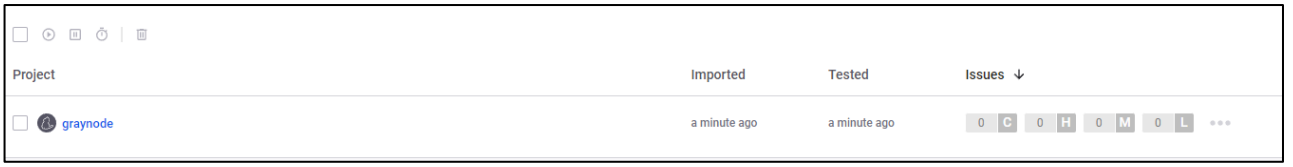


Результати статичного тестування безпеки

Результат статичного тестування безпеки за допомогою «yarn audit»

```
PS C:\Users\alexe\Desktop\graynode\graynode-backend> yarn audit --groups dependencies
yarn audit v1.22.19
0 vulnerabilities found - Packages audited: 248
Done in 0.79s.
```

Результат статичного тестування безпеки за допомогою Snyk



Результат статичного тестування безпеки за допомогою SonarQube