Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА

на тему:

Розробка захищеного корпоративного файлообмінника з двофакторною автентифікацією

Виконав: студент 4 курсу, гр.1КІТС-20б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Волос В. С.
(прізвище та ініціали)

Керівник: к.т.н., доц., каф. МБІС
Карпінець В. В.
(прізвище та ініціали)
«9» червня 2024 р.

Рецензент: к.т.н., доц., каф. ОТ
Кожем'яко А. В.
(прізвище та ініціали)
«9» червня 2024 р.

Допущено до захисту
Голова секції УБ кафедри МБІС
д.т.н., проф. Яремчук Ю.Є.
"9" червня 2024р.

Вінниця ВНТУ – 2024

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти І-й (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС

д.т.н., проф. Яремчук Ю.Є.
“ 22 ” березня 2024 р.

З А В Д А Н Н Я
на бакалаврську дипломну роботу студенту
Волосу Віталію Сергійовичу
(прізвище, ім`я, по-батькові)

1. Тема роботи Розробка захищеного корпоративного файлообмінника з двофакторною автентифікацією
Керівник роботи Карпінець Василь Васильович
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “11” березня 2024року № 80
2. Термін подання студентом роботи “5” червня 2024 року
3. Вихідні дані до роботи Електронні джерела, наукові статті, книги та документи, повязані з темою. Існуюче програмне забезпечення, технічна документація, вимоги та обмеження до програмного забезпечення
4. Зміст текстової частини:
Для досягнення мети, було поставлено наступні задачі: здійснити аналіз існуючих корпоративних файлообмінників, здійснити аналіз існуючих методів двофакторної автентифікації, побудувати запропоновані алгоритми роботи системи, здійснити проектування бази даних, здійснити проектування інтерфейсу, обрати мову програмування, та середовище розробки, програмно реалізувати захищений корпоративний файлообмінник з двофакторною автентифікацією, провести тестування розробленого корпоративного файлообмінника з двофакторною автентифікацією.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень)
Блок-схема алгоритму реєстрації, блок-схема алгоритму авторизації, блок-схема алгоритму перевірки сесії користувача, блок-схема алгоритму виходу користувача з облікового запису, блок-схема алгоритму завантаження файлів, блок-схема алгоритму видалення файлу.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Карпінець В. В., к.т.н., доц. каф. МБІС		
Розділ II	Карпінець В. В., к.т.н., доц. каф. МБІС		
Розділ III	Карпінець В. В., к.т.н., доц. каф. МБІС		

7.Дата видачі завдання 22 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1.	Визначення напрямку бакалаврської дипломної роботи, формулювання теми	23.03.2024	01.04.2024	Виконано
2.	Аналіз предметної області обраної теми	02.04.2024	08.04.2024	Виконано
3.	Розробка запропонованих алгоритмів роботи системи	09.04.2024	15.04.2024	Виконано
4.	Проектування бази даних	17.04.2024	21.04.2024	Виконано
5.	Проектування користувацького інтерфейсу	26.04.2024	29.04.2024	Виконано
6.	Програмна реалізація захищеного корпоративного файлообмінника з двофакторною автентифікацією	01.05.2024	14.05.2024	Виконано
7.	Тестування розробленого захищеного корпоративного файлообмінника з двофакторною автентифікацією	15.05.2024	19.05.2024	Виконано
8.	Попередній захист бакалаврської дипломної роботи	23.05.2024	23.05.2024	Виконано
9.	Виправлення, уточнення, корегування бакалаврської дипломної роботи	24.05.2024	29.06.2024	Виконано
10.	Захист бакалаврської дипломної роботи	12.06.2024	17.06.2024	Виконано

Студент _____ Вовк В.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Карпінець В.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дана дипломна робота присвячена розробці захищеного корпоративного файлообмінника з двофакторною автентифікацією. Метою дослідження і розробки є забезпечення безпечного обміну і зберігання файлів 3D моделей, між інженерами підприємства яке займається 3D друком.

Під час розробки захищеного корпоративного файлообмінника з двофакторною автентифікацією, було розроблено запропоновані алгоритми його роботи, та побудовано відповідні блок-схеми.

Захищений корпоративний файлообмінник був реалізований в середовищі розробки Visual Studio Code, за допомогою мови програмування JavaScript та Node js, для розробки серверу, було використано програмний каркас Express.js. Під час програмної реалізації було також використано додаткові модулі, фреймворки, та бібліотеки, такі як Dotenv, Mongoose, Speakeasy, Express-Session, Bcryptjs, та Multer.

В ході роботи, було проведено аналіз існуючих аналогів корпоративних файлообмінників, проаналізовано їх переваги та недоліки, також було здійснено аналіз методів двофакторної автентифікації.

В результаті виконаної роботи, було проведено різноманітні тестування працездатності, і захищеності розробленого захищеного корпоративного файлообмінника з двофакторною автентифікацією.

Ключові слова: корпоративний файлообмінник, двофакторна автентифікація, безпечний обмін, захищений корпоративний файлообмінник.

ABSTRACT

This thesis is devoted to the development of a secure corporate file sharing service with two-factor authentication. The purpose of research and development is to ensure secure exchange and storage of 3D model files between engineers of a 3D printing company.

During the development of a secure corporate file sharing service with two-factor authentication, the proposed algorithms for its operation were developed and the corresponding flowcharts were built.

The secure corporate file sharing service was implemented in the Visual Studio Code development environment, using the JavaScript and Node js programming languages, and the Express.js software framework was used to develop the server. Additional modules, frameworks, and libraries such as Dotenv, Mongoose, Speakeasy, Express-Session, Bcryptjs, and Multer were also used during the software implementation.

In the course of the work, we analyzed existing analogues of corporate file sharing services, analyzed their advantages and disadvantages, and analyzed two-factor authentication methods.

As a result of the work performed, various tests of the performance and security of the developed secure corporate file sharing service with two-factor authentication were carried out.

Keywords: corporate file sharing, two-factor authentication, secure exchange, secure corporate file sharing.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі безпека даних стала пріоритетом для більшості компаній. Запобігання витокам інформації, крадіжкам даних та несанкціонованому доступу стало життєво важливим завданням для будь-якої організації. Особливо це стосується корпоративних файлообмінників, які використовуються для обміну конфіденційною інформацією між співробітниками та зовнішніми партнерами.

Одним із ефективних методів захисту даних є використання двофакторної автентифікації. Цей підхід вимагає використання користувачем не лише основного пароля який був вказаний ним під час реєстрації, але й додаткового фактору автентифікації, такого як пароль на основі часу, SMS-код підтвердження, або відбиток пальця, чи лиця. Завдяки цьому навіть у випадку коли зловмисник має доступ до основного пароля, йому не вдасться отримати доступ до захищеного ресурсу, так як існує додатковий фактор автентифікації який йому не відомий, чи він не має до нього доступ.

Тому, аналізуючи наведені вище аргументи і факти, формується висновок того що розробка та використання захищеного корпоративного файлообмінника з двофакторною автентифікацією стає критично важливою для сучасних компаній. Використання такого рішення, дозволить забезпечити безпеку даних які зберігаються, на рівні корпоративного обміну між працівниками підприємства яке займається 3D друком, зменшуючи ризик їх витоку, чи небажаного несанкціонованого доступу.

Мета роботи. Метою роботи є розробка захищеного корпоративного файлообмінника з двофакторною автентифікацією, який буде сприяти підвищенню конфіденційності, цілісності і доступності даних, які циркулюють на підприємстві яке займається 3D друком.

Для досягнення даної мети, необхідно виконати наступні завдання:

- здійснити аналіз існуючих аналогів корпоративних файлообмінників;
- здійснити аналіз поняття двофакторної автентифікації;

- здійснити аналіз існуючих методів двофакторної автентифікації;

- розробити запропоновані алгоритми роботи захищеного корпоративного файлообмінника з двофакторною автентифікацією;

- обрати, та спроектувати базу даних;

- здійснити проектування користувацького інтерфейсу;

- здійснити вибір мови програмування, фреймворків, та бібліотек, здійснити вибір середовища розробки;

- здійснити програмну реалізацію спроектованого захищеного корпоративного файлообмінника з двофакторною автентифікацією;

- здійснити тестування роботи розробленого захищеного корпоративного файлообмінника з двофакторною автентифікацією;

Об’єктом дослідження є захищений корпоративний файлообмінник з двофакторною автентифікацією, для підприємства яке займається 3D друком.

Предметом дослідження є розробка та ефективність застосування захищеного корпоративного файлообмінника з двофакторною автентифікацією.

Новизна роботи полягає в розробці та дослідженні комплексного підходу до захисту конфіденційної інформації, в контексті підприємства яке займається 3D друком, через розробку, та впровадження захищеного корпоративного файлообмінника з двофакторною автентифікацією. Для забезпечення безпечного обміну файлами 3D моделей.

Практична цінність роботи, полягає в підвищені захищеності обміну даними, через розробку захищеного корпоративного файлообмінника з двофакторною автентифікацією. Розроблене рішення, дозволить підприємству яке займається 3D друком, уникнути фінансових, репутаційних втрат, та крадіжки конфіденційних даних, через можливі витоки, завдяки несанкціонованому доступу.

1. АНАЛІЗ ІСНУЮЧИХ КОРПОРАТИВНИХ ФАЙЛООБМІННИКІВ ТА МЕТОДІВ ДВОФАКТОРНОЇ АВТЕНТИФІКАЦІЇ

В даному розділі буде проаналізовано найпопулярніші корпоративні файлообмінники, та визначено їх переваги та недоліки. Також буде сформульовано поняття автентифікації, та визначено завдання які вона виконує. Ще буде розглянуто основні, найпопулярніші методи двофакторної автентифікації, проаналізовано їх переваги та недоліки. Та поставлено задачу для виконання в наступних розділах.

1.1 Аналіз існуючих корпоративних файлообмінників

Перш за все необхідно розібратися з поняттям корпоративного файлообмінника, розглянути найпопулярніші які є на ринку, та провести їх аналіз, визначити переваги та недоліки.

Корпоративний файлообмінник, файлхостинг або файловий хостинг - сервіс, що надає користувачеві місце під його файли та цілодобовий доступ до них через web, як правило, за протоколом http. Такий сервіс дозволяє зручно "обмінюватися" файлами. На спеціальній сторінці файлообмінника (найчастіше на головній) користувач завантажує файл на сервер файлообмінника, а файлообмінник віддає користувачеві постійне посилання, яке може розсилати по e-mail, публікувати в блогах, на форумах або пересилати через системи IM. Перейшовши за таким посиланням, будь-який інший користувач може завантажити початковий файл [1].

Так як технології постійно розвиваються, і потреби користувачів в обміні даними лише зростають, тому на ринку існує дуже багато, різноманітних корпоративних сервісів, які дозволяють зберігати і обмінюватися файлами.

Dropbox Business– це найпопулярніше хмарне сховище у світі представлене ц 2007 році компанією Dropbox Inc, призначене для зберігання файлів, з можливістю їх синхронізації на різних пристроях. Dropbox Business дозволяє зберігати і ділитися файлами будь яких форматів з іншими користувачами [2] (рис. 1.1).

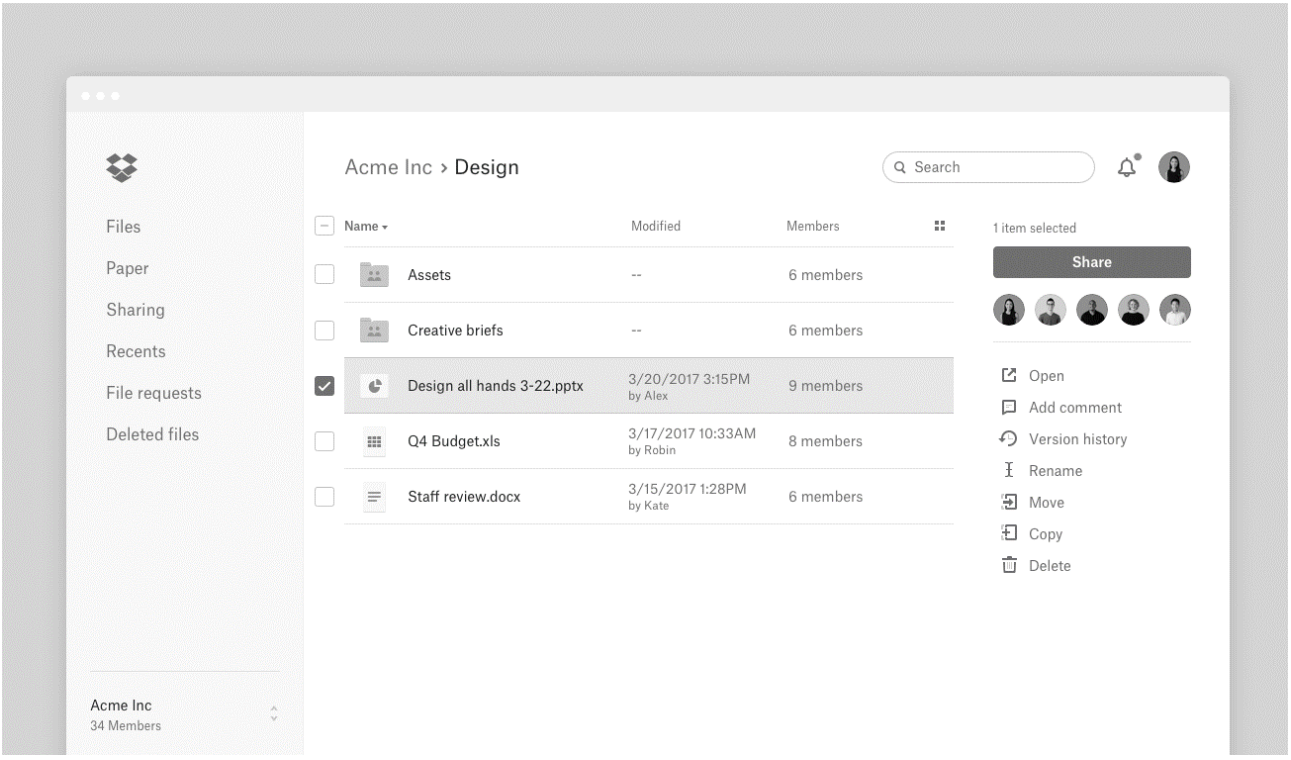


Рисунок 1.1 – зовнішній вигляд корпоративного файлообмінника Dropbox Business

Dropbox дає можливість переглядати, завантажувати файли, та створювати різноманітні папки для зручного сортування вмісту. За допомогою вбудованого текстового редактора Paper у Dropbox можна створювати та спільно працювати з документами. Також документи можна створювати за допомогою Google Документів та Microsoft Word . Ділитись файлами можна за посиланням або запрошенням, переглядати файли можуть навіть незареєстровані користувачі Dropbox.

Dropbox Business дозволяє користувачам створити спеціальну папку на своїх комп'ютерах, яку Dropbox синхронізує таким чином, що вона має однаковий вміст незалежно від того, який пристрій використовується для перегляду. Файли, розміщені в цій папці, також доступні через веб-сайт Dropbox і мобільні додатки. Dropbox працює за моделлю Freemium, в якій користувачі мають можливість створити безкоштовний обліковий запис із заданою кількістю вільного простору, тоді як для збільшення обсягу облікового запису необхідна платна підписка [3].

Переваги Dropbox Business [4]:

- суттєвою перевагою Dropbox є те що ним досить зручно користуватися так як розробники розробили повноцінну програму, як для мобільних пристроїв, так і для ПК.

- Dropbox працює майже на всіх операційних системах, таких як: IOS, Android, Mac OS, Windows, Linux.

- в Dropbox є можливість спробувати будь який тариф безкоштовно, протягом 30-ти денного пробного періоду. Пробний період можна скасувати в будь який момент [5].

- в Dropbox є можливість завантажувати великі файли розміром до 100 Гб.

- Dropbox дає можливість створювати резервні копії вивантажених файлів.

- є можливість переглядати історію версії файлів.

- Dropbox надає можливість відновлювати видалені файли протягом 180-165 днів, залежно від обраного тарифу.

Недоліки Dropbox [6]:

- ключовим недоліком є те що Dropbox використовує один фактор автентифікації, на основі звичайного статичного паролю.

- значним недоліком Dropbox є його вартість.

- недоліком Dropbox є обмежений функціонал безкоштовного тарифу.

Висновок: Dropbox є досить хорошим хмарним сервісом для обміну і зберігання файлів, так як він має більше переваг ніж недоліків.

Google Drive – це хмарне сховище розроблене компанією Google Inc, і призначене для зберігання, та обмінну файлів користувачів у хмарі. Google Drive включає в собі: Google Документи, Google Таблиці, та Google Презентації, це дозволяє декільком користувачам, спільно редагувати документи, електронні таблиці, презентації, малюнки, форми та багато інших файлів [7] (рис. 1.2).

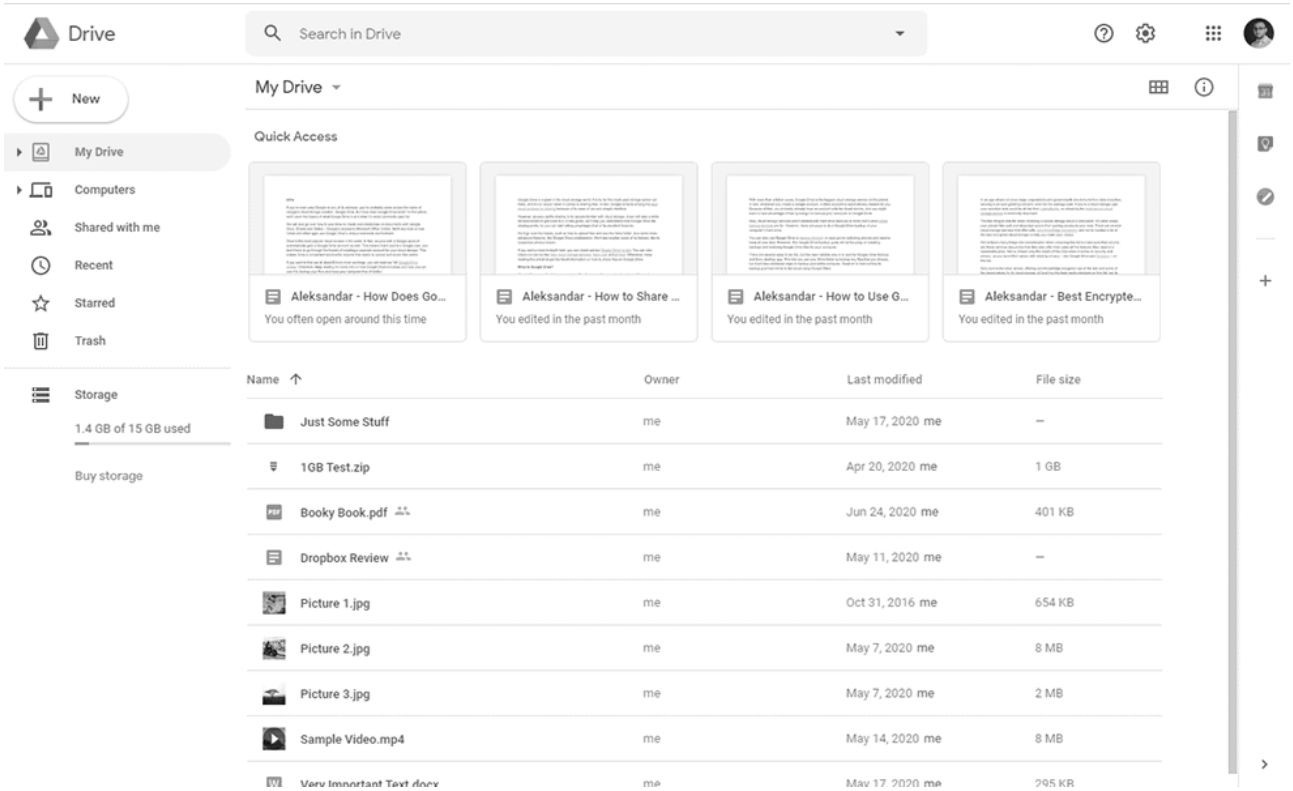


Рисунок 1.2 - зовнішній вигляд корпоративного файлообмінника Google Drive

Хмарне сховище Google Drive, є досить популярним так як щомісячно налічує більше 240 мільйонів активних користувачів.

Переваги Google Drive:

- Google Drive надає безкоштовно 15 Гб сховища для власного використання і завантаження будь яких файлів.

- є можливість підключити сімейну підписку на платний тариф і заощадити 17% вартості.

- наявність повноцінного застосунку як для мобільного пристрою, так і для ПК.

- Google Drive досить зручний у використанні, також встановлення і реєстрація, займає мало часу.

- досить зручний і інтуїтивно зрозумілий інткрфейс [8].

Недоліки Google Drive:

- ключовим недоліком є те що Google Drive використовує один фактор автентифікації, на основі звичайного статичного паролю.

- Google Drive не надає можливості автоматичного вивантаження фото з пристрою в хмару. Замість цього у Google є функція Auto Backup в мобільному додатку Google+, яке відправляє фотографії в профіль Google+.

- Google Drive немає можливості безкоштовно протестувати додаткові можливості платних тарифів.

Висновок: Google Drive є досить хорошим і зручним хмарним сховищем, так як його інтерфейс є доволі зручним і зрозумілим, крім цього сховище має можливість спільного редагування різноманітних файлів.

OneDrive Business – це хмарне сховище, продукт компанії Microsoft, який був представлений в 2007 році. Сервіс дозволяє користувачам зберігати файли різних типів, такі як документи, фото, відео, аудіо в хмарному сховищі, це забезпечує постійний, віддалений доступ до них з різних пристроїв [9] (рис. 1.3).

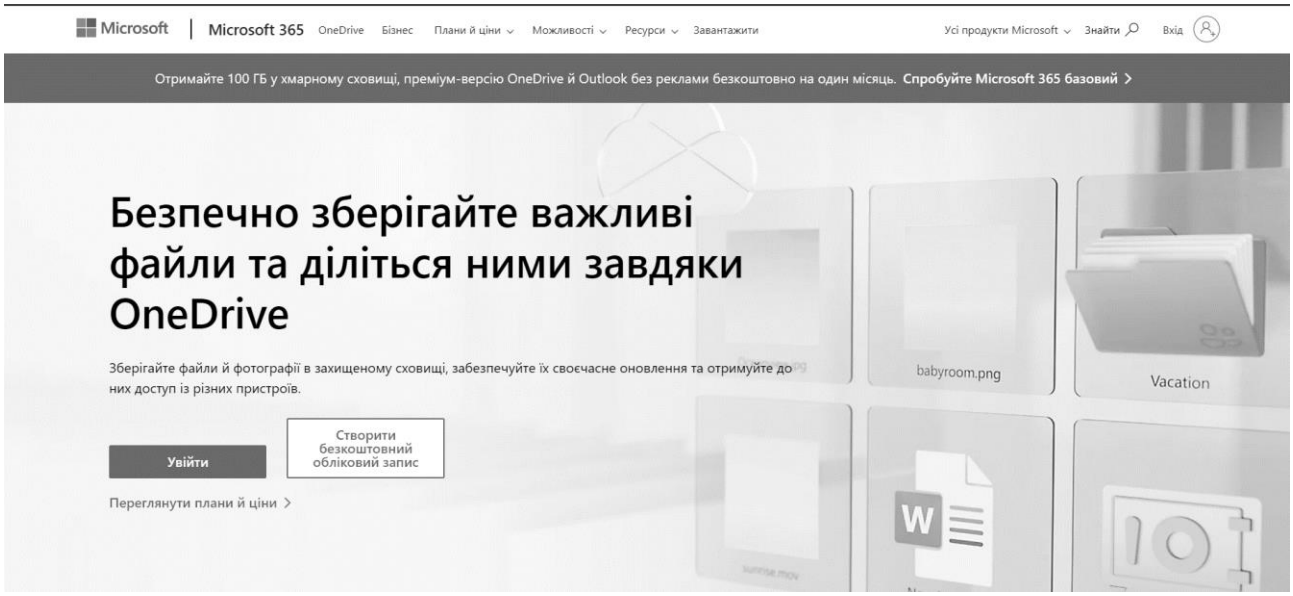


Рисунок 1.3 - зовнішній вигляд корпоративного файлообмінника OneDrive Business

Переваги OneDrive Business:

- безкоштовне сховище розміром 5 Гб.

- доступність до вивантажених файлів з будь якого під’єднаного пристрою, який має доступ до інтернету.

- OneDrive має інтеграцію з Microsoft, що дозволяє спільно використовувати і редагувати файли.

- OneDrive має можливість автоматично синхронізувати файли між під’єднаними пристроями.

- OneDrive має інтеграцію з Windows, що дає можливість автоматично вивантажувати важливі файли в хмару.

Недоліки OneDrive:

- ключовим недоліком є те що OneDrive використовує один фактор автентифікації, на основі звичайного статичного паролю.

- безкоштовне сховище розміром 5 Гб, може бути замалим для більшості користувачів.

- OneDrive може повноцінно працювати лише на пристроях з операційною системою Windows, а от для користуачів з операційними системами Mac OS чи Linux, функціонал обмежений.

Наведені вище корпоративні файлообмінники, не є достатньо захищеними, так як доступ до захищених даних використовується лише один фактор автентифікації, а саме звичайний статичний пароль. Тому було прийнято рішення розробити корпоративний файлообмінник з додатковим фактором автентифікації, для більш захищеного обміну секретними файлами на підприємстві.

1.2 Загальні поняття автентифікації

Перш за все необхідно визначити таке поняття як автентифікація, за що відповідає, та чим вона відрізняється від авторизації.

Автентифікація — це процес визначення того, чи хтось або щось є тим, ким або тим, за кого себе видає. Технологія автентифікації забезпечує контроль доступу до систем, перевіряючи, чи збігаються облікові дані користувача з обліковими даними в базі даних авторизованих користувачів або на сервері автентифікації даних. При цьому автентифікація гарантує безпеку систем, процесів і корпоративної інформації [10].

Існує кілька типів аутентифікації. Для ідентифікації, користувачі зазвичай ідентифікуються за ідентифікатором користувача; автентифікація відбувається,

коли користувач надає облікові дані, такі як пароль, які відповідають його ідентифікатору користувача.

Автентифікація дозволяє організаціям підтримувати безпеку своїх мереж, дозволяючи лише автентифікованим користувачам або процесам доступ до захищених ресурсів. Це може включати персональні комп'ютери, бездротові мережі, бездротові точки доступу, бази даних, веб-сайти та інші мережеві програми та служби.

Після автентифікації користувач або процес зазвичай піддається процесу авторизації, щоб визначити, чи слід автентифікованому об'єкту надати доступ до певного захищеного ресурсу чи системи. Користувача можна автентифікувати, але не отримати доступ до ресурсу, якщо цьому користувачеві не було надано дозволу на доступ до нього [11].

Поняття автентифікації та авторизації часто використовуються як однакові, але це не так, це різні поняття, які виконують різні функції. Автентифікація передбачає підтвердження ідентифікації раніше зареєстрованого користувача або процесу перед наданням доступу до захищених мереж чи систем. Авторизація — це більш детальний процес, який гарантує, що автентифікованому користувачу або процесу надано дозвіл на доступ до конкретного ресурсу [12].

Процес, за допомогою якого доступ до деяких захищених ресурсів обмежується певними користувачами, називається контролем доступу. У моделях контролю доступу автентифікація завжди передує авторизації. Різні типи контролю доступу вимагають різних рівнів автентифікації [13].

Під час автентифікації облікові дані, надані користувачем, порівнюються на ідентичність з обліковими даними, збереженими в базі даних авторизованих користувачів. Ця база даних може бути розташована або на локальному сервері операційної системи, або на сервері автентифікації. Якщо надані дані збігаються з збереженими даними, тоді автентифікований користувач має право використовувати ресурс, тому отримує доступ.

Традиційно автентифікацію здійснюють системи або ресурси, до яких здійснюється доступ. Наприклад, сервер автентифікує користувачів, за допомогою власних алгоритмів автентифікації, на основі раніше наданих даних, користувачем, таких як паролі, або інші ідентифікатори для входу.

Організації використовують автентифікацію, щоб контролювати, хто може отримати доступ до корпоративних мереж і ресурсів, а також ідентифікувати та контролювати, які машини та сервери мають доступ. Компанії також використовують автентифікацію, щоб надати віддаленим співробітникам безпечний доступ до програм і мереж [14].

Серед конкретних випадків використання автентифікації, є вхід в корпоративні системи, де методи автентифікації використовуються для перевірки особистості співробітників і надання їм доступу до корпоративних систем, таких як електронна пошта, бази даних і сховища документів, інтернет-банкінг і фінансові операції, в яких методи автентифікації перевіряють особу клієнтів і гарантують, що лише авторизовані користувачі можуть отримати доступ до банківських рахунків, затверджувати фінансові операції та виконувати інші онлайн-банкінгові операції, та транзакції електронної комерції, де методи автентифікації також використовуються для перевірки особи клієнтів, захисту конфіденційної інформації та забезпечення безпечних онлайн-транзакцій.

Автентифікація забезпечує конфіденційність, цілісність і доступність корпоративних даних, допомагає запобігти шахрайству, підвищити довіру клієнтів, та забезпечує підтримку безпеки мережевої інфраструктури організації [15].

1.3 Аналіз існуючих методів двофакторної автентифікації

Як було визначено в попередньому підрозділі, автентифікація — це процес визначення того, чи хтось або щось є тим, ким або тим, за кого себе видає, за допомогою ідентифікатора користувача, та пароля. Але такий метод автентифікації вважається не достатньо надійним, оскільки він покладається

лише на один фактор автентифікації. Саме такий фактор автентифікації використовують всі існуючі на ринку корпоративні файлообмінники.

Так як метою є розробка корпоративного файлообмінника з двофакторною автентифікацією, що є більш захищеним рішенням, тому слід розглянути її детальніше. Так як двофакторна автентифікація, покладена на два різні типи факторів автентифікації. Її суть полягає в тому, що під час перевірки особи враховується не лише пароль і ідентифікатор користувача, а і ще якийсь фактор автентифікації [16].

Існує декілька факторів автентифікації:

- Фактор знань. Фактор знань або щось, що може знати користувач , може бути будь яким знанням, яким володіє користувач, наприклад персональний ідентифікаційний номер (PIN), ім’я користувача, пароль або відповіді на секретні питання [17].

- фактор володіння. Фактор володіння або щось , чим володіє користувач, може бути будь-чим, чим може володіти та носити їх із собою користувач, включно з апаратними пристроями, такими як смарт-карта чи мобільний телефон, які використовуються для набору текстового повідомлення або запуску програми автентифікації який може генерувати одноразовий пароль (OTP) або PIN-код [18].

- фактор приналежності. Фактор приналежності або те, ким є користувач, зазвичай базується на біометричній ідентифікації, наприклад відбитках пальців або великого пальця, розпізнаванні обличчя або скануванні сітківки ока [19].

- фактор розташування. Те де знаходиться користувач, може бути менш конкретним, але іноді використовується як доповнення до інших факторів. Розташування можна визначити з достатньою точністю за допомогою пристроїв, обладнаних, або з меншою точністю шляхом перевірки мережевих адрес і маршрутів [20].

- фактор часу. Подібно до фактору розташування, фактор часу є ще одним додатковим механізмом для відсіювання зловмисників, які намагаються отримати доступ до ресурсу, коли цей ресурс недоступний для авторизованого

користувача. Це часто поєднується з місцем розташування. Наприклад, якщо користувач востаннє автентифікувався в 12:00 в США, спроба автентифікації з Німеччини через годину, буде відхилена [21].

Серед різноманітних методів і варіантів реалізації двофакторної автентифікації, варто виділити найпопулярніші, ті які широко використовуються в сучасних системах, для додаткового рівня захисту:

1. Одноразові коди які надходять SMS, електронною поштою, чи дзвінком.

Один із найпопулярніших методів реалізації двофакторної автентифікації, є одноразовий код підтвердження. Як правило вони надсилаються текстовим повідомленням, у вигляді SMS, чи повідомлення на електронну пошту, але також може бути передане під час голосового дзвінка, на номер мобільного телефону, який був наданий під час реєстрації (рис. 1.4) [22].

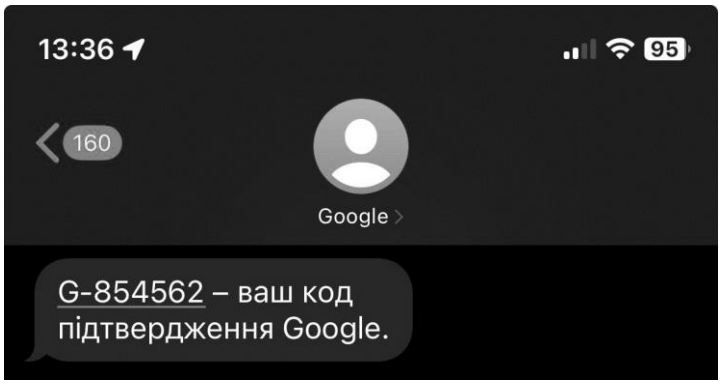


Рисунок 1.4 – приклад SMS коду підтвердження для двофакторної автентифікації

Незалежно від того яким методом було надано одноразовий код підтвердження, ідея залишається однаковою, перевірити здатність користувача, який хоче отримати доступ до захищеного ресурсу, отримати доступ до якогось іншого облікового запису чи номеру мобільного телефону. Таким чином, якщо зловмисник знаючи пароль, здійснить спробу несанкціонованого доступу до захищеного ресурсу, йому це не вдасця, так як доступу до телефону чи електронної пошти він не має.

Але цей метод має також свої недоліки, якщо телефон буде втрачено чи заблоковано, то відновити доступ до облікового запису буде неможливо. Або якщо користувач використовує однаковий пароль для доступу до електронної пошти, так і для облікового запису який намагається захистити, то зловмисник зможе легко отримати доступ до електронної пошти, і в подальшому отримати доступ до захищеного ресурсу.

2. Одноразові коди з програми автентифікації.

Цей метод двофакторної автентифікації, є досить популярним, так як є дуже зручним, і легким ц використанні. Суть даного методу полягає в тому що, для його роботи користувачу необхідно завантажити собі на пристрій спеціальну програму, таку як Google Authenticator або Authy, після чого помістити туди випадково згенерований програмою яку потрібно захистити токен, який відповідає за двофакторну автентифікацію. Після цього програма генерує тимчасовий (в більшості випадків на 30 секунд) ключ (зазвичай 6 цифр), який необхідно використовувати при авторизації, разом із основним паролем (рис. 1.5) [23].

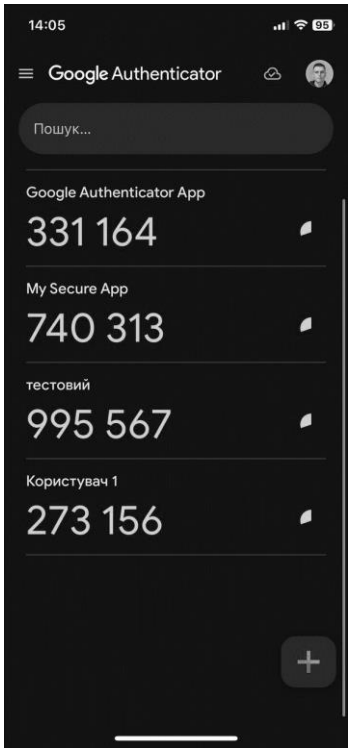


Рисунок 1.5 – приклад генерації тимчасового пароля в застосунку Google Authenticator для двофакторної автентифікації

Незважаючи на зручність даного методу, він також має один недолік, він полягає в тому що при втраті пристрою з програмою для генерації тимчасових ключів, зловмисник отримає доступ до всіх тимчасових ключів які були підключені в програмі, це може загрожувати втраті доступу над ними.

3. Біометрія: відбиток пальця, обличчя, або голосу [24].

Цей метод, є також досить популярним, і з кожним роком популярність його використання лише зростає, так як все більше виробників мобільних пристроїв в своїх виробах використовують, спеціальні датчики для сканування і аналізу структури обличчя, або відбитку пальця. Також даний метод є досить ефективним так як похибка роботи Face ID на пристроях компанії Apple становить 1 до 1 000 000, а Touch ID 1 до 50 000, це значить що вірогідність того що зловмисник буде сприйнятий як власник, дуже низька (рис. 1.6) [25-26].

Рисунок 1.6 – логотипи Face ID, і Touch ID від компанії Apple

Недоліком такого методу є те що біометричні дані є досить чутливими, і в разі якоїсь травми, даний метод перестає працювати. Ще одним недоліком є те що далеко не у всіх пристроях є необхідне обладнання для його роботи, що значно знижує його популярність.

В загальному, сьогодні двофакторна автентифікація є досить надійним методом захисту даних від несанкціонованого доступу. Залежно від проблеми, потреби і можливостей організації, можна обрати той фактор додаткової автентифікації, який буде підходити ідеально, щоб система була максимально захищеною, але в той же час не надто дорогою і складною в реалізації.

Проаналізувавши всі можливі методи двофакторної автентифікації, врахувавши їх переваги та недоліки, було прийнято рішення використовувати

другий фактор автентифікації для захищеного корпоративного файлообмінника, який базується на основі одноразового тимчасового коду, так як в використанні, він є досить зручним і надійним.

1.4 Висновки та постановка задач

В даному розділі було проаналізовано найпопулярніші сучасні корпоративні файлообмінники, такі як, Dropbox Business, Google Drive, OneDrive Business, визначено їх переваги та недоліки, зроблено відповідні висновки.

Також було визначено і сформовано поняття автентифікації і двофакторної автентифікації, та визначено відмінність від авторизації.

Було здійснено аналіз сучасних методів двофакторної автентифікації, таких як одноразові коди які надходять SMS, електронною поштою, чи дзвінком, одноразові коди з програми автентифікації, та біометричні методи двофакторної автентифікації, такі як відбиток пальця, обличчя, або голосу. Було проаналізовано переваги та недоліки кожного з методів.

В результаті даного розділу було визначено проблему в відсутності на ринку достатньо захищених корпоративних файлообмінників з двофакторною автентифікацією. Та визначено і обрано метод двофакторної автентифікації, який буде впроваджено під час розробки в захищений корпоративний файлообмінник.

Виходячи з проведеного аналізу, можна сформувати тему роботи: «Розробка захищеного корпоративного файлообмінника з двофакторною автентифікацією».

Для досягнення визначеної мети необхідно виконати наступні завдання:

- побудувати запропоновані алгоритми роботи системи;
- обрати мову програмування, та середовище розробки;
- програмно реалізувати захищений корпоративний файлогобмінник з двофакторною автентифікацією;
- провести тестування розробленого корпоративного файлообмінника з двофакторною автентифікацією.

2. ПРОЕКТУВАННЯ ЗАХИЩЕНОГО КОРПОРАТИВНОГО ФАЙЛООБМІННИКА З ДВОФАКТОРНОЮ АВТЕНТИФІКАЦІЄЮ

В даному розділі буде розроблено запропоновані алгоритми роботи захищеного корпоративного файлообмінника з двофакторною автентифікацією, та побудовано відповідні блок-схеми даних алгоритмів, буде обрано і спроектовано базу даних, буде розроблено її ER-модель. Також буде спроектовано користувацький інтерфейс, для захищеного корпоративного файлообмінника з двофакторною автентифікацією.

2.1 Розробка запропонованих алгоритмів роботи системи захищеного корпоративного файлообмінника з двофакторною автентифікацією

Пропонується розробити веб додаток який буде мати 3 основні сторінки. А саме, сторінка реєстрації нових користувачів, сторінка авторизації раніше зареєстрованих користувачів, і захищена сторінка самого корпоративного файлообмінника. Файлообмінник буде розроблено для використання в корпоративних цілях, а саме розробляється для підприємства яке займається 3D друком. Тому його користувачами є інженери даного підприємства, вони матимуть змогу обмінюватися файлами 3D моделей, збільшуючи їх захист за рахунок двофакторної автентифікації.

Виходячи з цього в даному підрозділі буде розроблено запропоновані алгоритми роботи системи захищеного корпоративного файлообмінника з двофакторною автентифікацією.

Першим алгоритмом який було розроблено, став алгоритм реєстрації користувачів, в якого є всі необхідні перевірки користувачів [27].

Детальний, покроковий огляд алгоритму реєстрації:

Крок 1 – Початок.

Користувач розпочинає процес реєстрації відкривши відповідну сторінку.

Крок 2 – Введення даних для реєстрації.

Користувач вводить свої дані на відповідній сторінці для реєстрації.

Крок 3 – Надсилання запиту на сервер.

Після заповнення всіх обов’язкових полів для реєстрації користувач натискає відповідну кнопку «Зареєструватись», таким чином його дані відправляються на сервер.

Крок 4 – Отримання запиту.

Після чого сервер отримує запит на реєстрацію.

Крок 5 – Перевірка чи email дозволений для реєстрації.

Сервер перевіряє чи введений клієнтом email, дозволений для реєстрації, для того щоб уникнути несанкціонованої реєстрації користувачів, яким реєстрація заборонена.

Крок 6 – Email дозволений?

Крок 6.1 – Ні. Надсилання помилки про заборону реєстрації даного користувача (email).

Якщо даного email немає в списку дозволених, то користувачу надсилається помилка, з забороною реєстрації.

Крок 6.1.1 – Отримання відповіді.

Клієнт отримує відповідь.

Крок 6.1.2 – Виведення помилки користувачу.

Виводиться помилка користувачу про заборону реєстрації даного email.

Крок 6.2 – Так.

Якщо email є в списку дозволених для реєстрації, виконується крок 7.

Крок 7 – Перевірка чи користувач з таким email ще не зареєстрований.

Сервер перевіряє чи ще не зареєстрований користувач з таким email.

Крок 8 – Email вже зареєстрований?

Крок 8.1 – Так. Надсилання помилки про те що користувач з таким email уже зареєстрований.

Якщо користувач з таким email уже зареєстрований, то клієнту надсилається помилка, про те що користувач з даним email уже зареєстрований.

Крок 8.1.1 – Отримання відповіді.

Клієнт отримує відповідь.

Крок 8.1.2 – Виведення помилки користувачу.

Виводиться помилка користувачу про те що користувач з таким email уже зареєстрований.

Крок 8.2 – Ні.

Якщо email ще не зареєстрований, виконується крок 9.

Крок 9 – Хешування паролю.

Сервер хешує введений пароль.

Крок 10 – Генерація секретного ключа.

Сервер генерує секретний ключ для двофакторної автентифікації.

Крок 11 – Створення нового користувача в базі даних.

Сервер створює нового користувача в базі даних.

Крок 12 – Надсилання відповіді клієнту.

Сервер надсилає відповідь клієнту.

Крок 13 – Отримання відповіді.

Клієнт отримує відповідь.

Крок 14 – Вивід повідомлення користувачу.

Клієнту виводиться секретний ключ для двофакторної автентифікації, і повідомлення про успішну реєстрацію.

Крок 15 – Кінець.

Таким чином відбувається процес реєстрації, з забороною несанкціонованої реєстрації користувачів.

Також було розроблено блок-схему запропонованого розробленого алгоритму реєстрації (рис. 2.1).

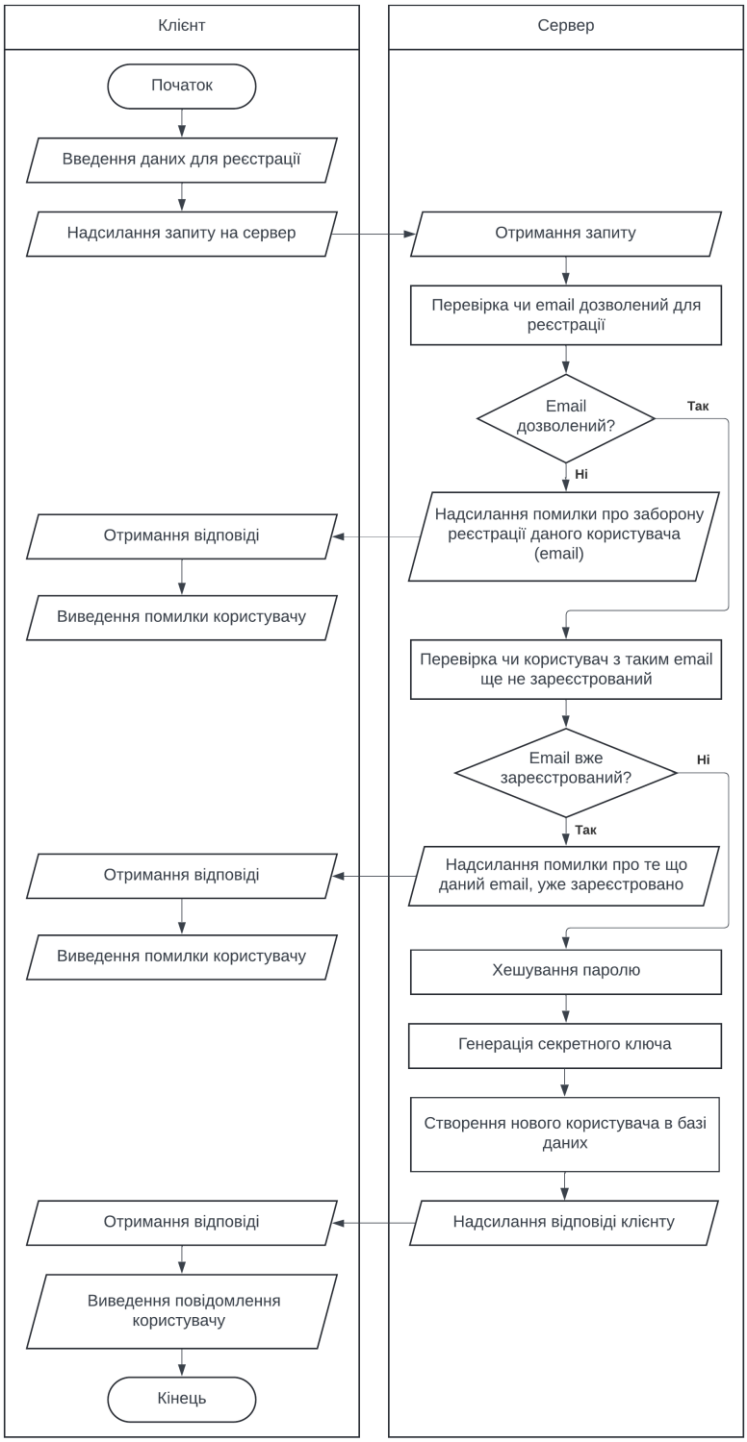


Рисунок 2.1 – розроблена блок-схема запропонованого алгоритму реєстрації користувачів

Блок схема демонструє алгоритм реєстрації, з усіма необхідними перевірками, для того щоб уникнути несанкціонованої реєстрації користувачів, яким не дозволено доступ, до захищеного корпоративного файлообмінника.

Наступним було розроблено запропонований алгоритм автентифікації, зареєстрованих користувачів.

Детальний, покроковий огляд алгоритму автентифікації:

Крок 1 – Початок

Користувач розпочинає процес автентифікації відкривши відповідну сторінку.

Крок 2 – Введення даних для автентифікації.

Користувач вводить свої дані на відповідній сторінці для автентифікації.

Крок 3 – Надсилання запиту на сервер.

Після заповнення всіх полів для автентифікації, користувач натискає відповідну кнопку «Увійти», таким чином його дані відправляються на сервер, для подальшої обробки.

Крок 4 – Отримання запиту.

Сервер отримує запит на автентифікації.

Крок 5 – Перевірка чи користувач з введеним email уже зареєстрований.

Сервер перевіряє чи є зареєстрований користувач з введеним email клієнтом.

Крок 6 – Email зареєстрований?

6.1 – Ні. Надсилання помилки про те що даний email (користувач), не зареєстрований.

Якщо користувач з введеним email ще не зареєстрований то клієнту надсилається відповідна помилка.

6.2 – Так.

Якщо користувач з введеним email зареєстрований, виконується крок 7.

Крок 7 – Перевірка введеного паролю, з паролем з бази даних.

Сервер перевіряє захешований введений пароль клієнтом, з захешованим паролем з бази даних.

Крок 8 – Пароль співпадає?

Крок 8.1 – Ні. Надсилання помилки про неправильно введений пароль.

Якщо пароль не співпадає, то клієнту надсилається помилка, про неправильно введений пароль.

Крок 8.1.1 – Отримання відповіді.

Клієнт отримує відповідь сервера.

Крок 8.1.2. Виведення помилки користувачу.

Виводиться помилка користувачу, про неправильно введений пароль.

Крок 8.2 – Так.

Якщо пароль співпадає, виконується крок 9.

Крок 9 – Перевірка валідності введеного токену.

Сервер перевіряє чи введений користувачем токен для двофакторної автентифікації, вірний.

Крок 10 – Токен валідний?

Крок 10.1 – Ні. Надсилання помилки про неправильно введений токен.

Якщо токен введений неправильно, то клієнту надсилається помилка про неправильний токен.

Крок 10.1.1 – Отримання відповіді.

Клієнт отримує відповідь сервера.

Крок 10.1.2 – Виведення помилки користувачу.

Виводиться помилка користувачу про неправильно введений токен.

Крок 10.2 – Так.

Якщо токен введений вірно, виконується крок 11.

Крок 11 – Створення сесії користувача.

Створюється сесія активності з збереженням даних користувача.

Крок 12 – Надсилання відповіді клієнту.

Сервер надсилає відповідь клієнту про успішну автентифікацію.

Крок 12.1 – Отримання відповіді.

Клієнт отримує відповідь сервера.

Крок 12.2 – Виведення повідомлення користувачу.

Користувачу виводиться повідомлення про успішну автентифікацію.

Крок 12.3 – Перенаправлення користувача на захищену сторінку.

Після успішної автентифікації користувача, автоматично відкривається захищена сторінка з самим корпоративним файлообмінником.

Крок 13 – Кінець.

Таким чином побудовано алгоритм автентифікації користувача, з різними перевірками і автоматичним перенаправленням на захищену сторінку корпоративного файлообмінника.

Для даного розробленого алгоритму також було побудовано блок-схему алгоритму автентифікації (рис. 2.2).

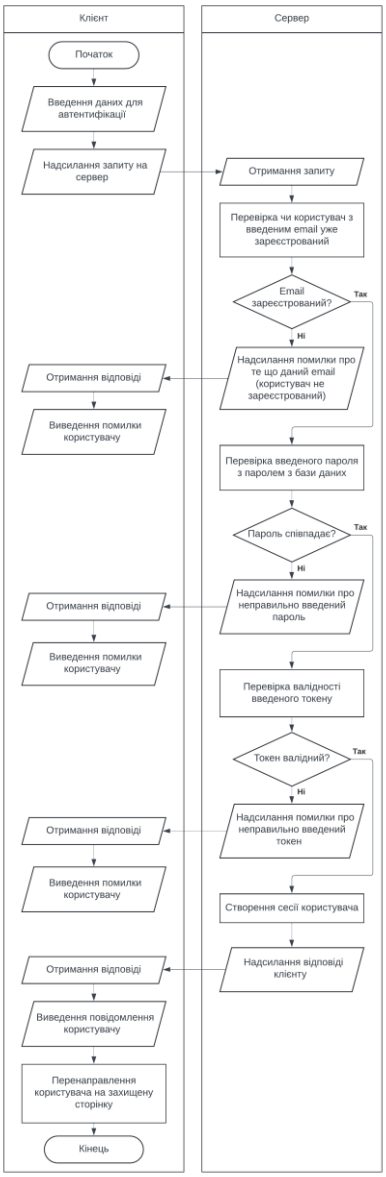


Рисунок 2.2 – розроблена блок-схема запропонованого алгоритму автентифікації користувачів

Дана блок-схема демонструє алгоритм автентифікації користувачів, і створення їм сесії активності з збереженням їхніх даних.

Для побудови захищеного корпоративного файлообмінника було розроблено також алгоритм, для перевірки активних сесій автентифікованих користувачів, і надання їм доступу до захищеної сторінки.

Детальний, покроковий огляд алгоритму перевірки сесій користувачів:

Крок 1 – Початок.

Крок 2 – Спроба отримати доступ до захищеної сторінки.

Клієнт розпочинає процес перевірки доступу, відкривши відповідну, захищену сторінку.

Крок 3 – Надсилання запиту на сервер.

Відкривши захищену сторінку користувач надсилає запит на сервер.

Крок 4 – Отримання запиту.

Сервер отримує запит, на перевірку доступу.

Крок 5 – Перевірка наявності сесії поточного користувача.

Сервер перевіряє чи є активна сесія для конкретного користувача який хоче отримати доступ до захищеної сторінки.

Крок 6 – Сесія існує?

Крок 6.1 – Ні. Надсилання помилки про відсутність сесії.

Сервер надсилає клієнту помилку про те що сесія відсутня.

Крок 6.1.1 – Отримання відповіді.

Клієнт отримує відповідь про відсутність активної сесії.

Крок 6.1.2 – Відмова в доступі, до захищеної сторінки.

Користувачу відмовлено в доступі до захищеної сторінки, через наявність активної сесії.

Крок 6.1.3 – Виведення помилки користувачу.

Користувачу виводиться повідомлення про відмову в доступі до захищеної сторінки.

Крок 6.2 – Так.

Якщо сесія існує, виконується крок 7.

Крок 7 – Надсилання відповіді про активну сесію.

Клієнту надсилається відповідь, про те що його сесія активна.

Крок 7.1 – Отримання відповіді.

Клієнт отримує відповідь сервера, про те що його сесія активна.

Крок 7.2 – Надання доступу до захищеної сторінки.

Якщо сесія користувача активна, йому дозволено доступ до захищеної сторінки.

Крок 8 – Кінець.

Таким чином було реалізовано алгоритм перевірки активних сесій користувачів, для подальшого надання їм доступу до захищеної сторінки корпоративного файлообмінника.

Для алгоритму перевірки активних сесій користувачів, також було побудовано блок-схему (рис. 2.3).

Рисунок 2.3 – розроблена блок-схема запропонованого алгоритму перевірки сесій користувачів

Розроблені вище алгоритми є ключовими алгоритмами захищеного корпоративного файлообмінника з двофакторною автентифікацією, так як вони відіграють ключову роль в захищеності системи, надають можливість реєстрації лише дозволеним користувачам, під час входу користувача в систему, перевіряє чи користувач зареєстрований, та надають доступ до захищеної сторінки лише тим користувачам, в яких є сесія активності.

Наступним розроблено запропонований алгоритм виходу користувачів із свого облікового запису.

Детальний, покроковий огляд алгоритму виходу користувачів, з облікового запису:

Крок 1 – Початок.

Крок 2 – Натиснення кнопки вихід.

Клієнт розпочинає процес виходу із облікового запису натиснувши відповідну кнопку.

Крок 3 – Надсилання запиту на сервер.

Натиснувши кнопку виходу, користувач надсилає запит на сервер.

Крок 4 – Отримання запиту.

Сервер отримує запит, на вихід користувача.

Крок 5 – Видалення сесії користувача.

Крок 6 – Сесію видалено?

Крок 6.1 – Ні. Надсилання помилки про збій під час видалення.

Якщо сесію не було видалено, клієнту надсилається відповідна помилка.

Крок 6.1.1 – Отримання відповіді

Клієнт отримує відповідь про те що його сесія не була видалена.

Крок 6.1.2 – Виведення помилки користувачу.

Користувачу виводиться відповідне повідомлення, про помилку під час видалення сесії.

Крок 6.2 – Так.

Якщо сесію було успішно видалено. Виконується крок 7.

Крок 7 – Надсилання відповіді клієнту.

Клієнту надсилається відповідь, якщо сесію було видалено.

Крок 8 – Отримання відповіді

Клієнт отримує відповідь про успішне видалення сесії.

Крок 9 – Виведення повідомлення користувачу.

Користувачу виводиться відповідне повідомлення.

Крок 10 – Кінець.

Так виглядає алгоритм виходу користувачів з своїх облікових записів.

Для цього алгоритму також було розроблено блок-схему, для зручного, візуального представлення (рис. 2.4).

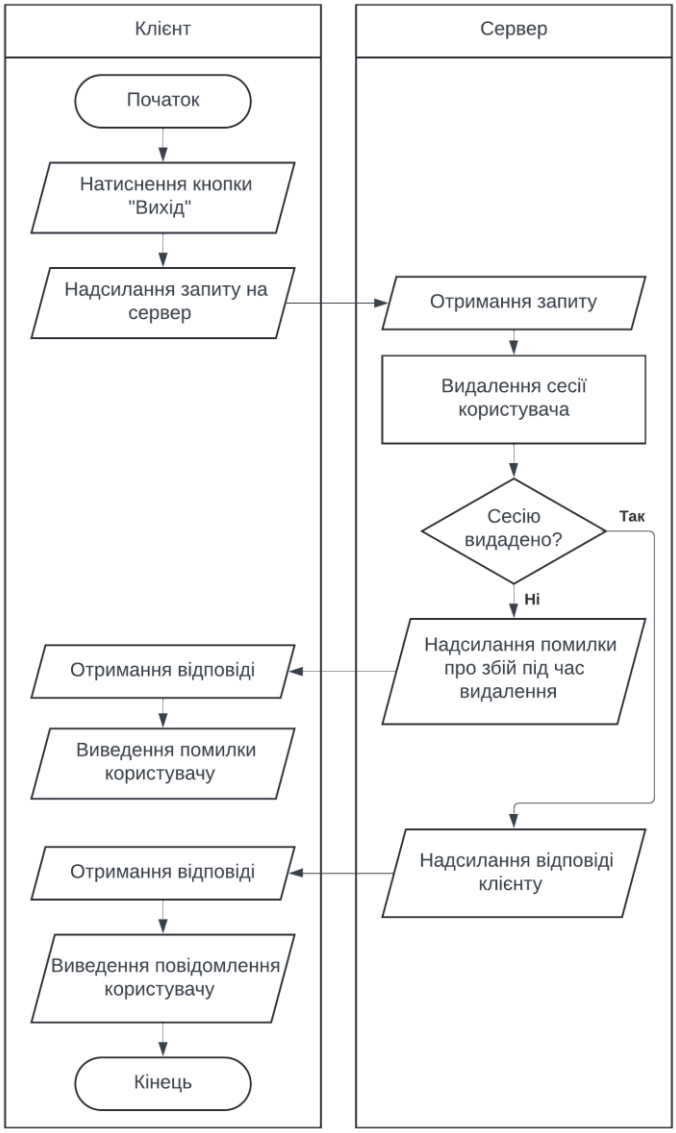


Рисунок 2.4 – розроблена блок-схема запропонованого алгоритму виходу користувачів, з облікових записів

Наступним було розроблено запропонований алгоритм завантаження файлів користувачами, з клієнтської сторони в колекцію бази даних.

Детальний, покроковий огляд алгоритму завантаження файлів:

Крок 1 – Початок

Користувач розпочинає процес завантаження файлів, обравши файл для завантаження на відповідній сторінці корпоративного захищеного файлообмінника.

Крок 2 – Вибір файлу для завантаження.

Користувач обирає файл який хоче завантажити в файлообмінник.

Крок 3 – Ввід опису файлу.

Користувач додає опис до обраного файлу.

Крок 4 – Надсилання запиту на сервер.

Після вибору файлу і вводу його опису, користувач натискає відповідну кнопку «Завантажити», таким чином файл і його опис відправляються на сервер, для подальшої обробки.

Крок 5 – Отримання запиту.

Сервер отримує запит на завантаження файлу.

Крок 6 – Отримання імені користувача.

З сесії активності, сервер отримує ім’я користувача, для визначення хто саме з користувачів завантажив файл.

Крок 7 – Збереження файлу.

Сервер зберігає інформацію про файл в базі даних, а саме його оригінальну назву, доданий користувацький опис, зашифрований вміст самого файлу, і ім’я користувача який завантажив файл.

Крок 8 – Файл збережено?

Крок 8.1 – Ні. Надсилання помилки про збій під час завантаження.

Якщо під час збереження файлу виникла помилка, клієнту надсилається відповідна помилка.

Крок 8.1.1 – Отримання відповіді.

Клієнт отримує відповідь.

Крок 8.1.2 – Виведення помилки користувачу.

Крок 8.2 – Надсилання відповіді клієнту.

Якщо файл було успішно збережено, клієнту надсилається відповідь.

Крок 8.2.1 – Отримання відповіді.

Клієнт отримує відповідь сервера.

Крок 8.2.2 – Виведення повідомлення користувачу.

Користувачу виводиться повідомлення про успішне завантаження файлу.

Крок 9 – Кінець.

Також було розроблено блок-схему для запропонованого алгоритму завантаження файлів (рис. 2.5).

Рисунок 2.5 – розроблена блок-схема запропонованого алгоритму завантаження файлів

Наступним було розроблено алгоритм видалення завантажених файлів, з перевіркою власності файлу.

Детальний, покроковий огляд алгоритму видалення файлів, з перевіркою власності:

Крок 1 – Початок

Користувач розпочинає процес видалення файлу, натиснувши відповідну кнопку біля файлу.

Крок 2 – Натиснення кнопки «Видалити».

Користувач здійснює спробу видалення файлу.

Крок 3 – Надсилання запиту на сервер.

Після натиснення користувачем на кнопку «Видалити», запит на видалення файлу надсилається на сервер.

Крок 4 – Отримання запиту.

Сервер отримує запит на видалення файлу.

Крок 5 – Отримання ідентифікатора файлу.

Сервер отримає унікальний ідентифікатор файлу, який користувач бажає видалити.

Крок 6 – Пошук файлу за ідентифікатором.

Сервер розпочинає пошук файлу в базі даних, за його унікальним ідентифікатором.

Крок 7 – Файл знайдено?

Крок 7.1 – Ні. Надсилання помилки про відсутність файлу.

Якщо бажаний файл не було знайдено в базі даних, то клієнту надсилається відповідна помилка.

Крок 7.1.1 – Отримання відповіді.

Клієнт отримує відповідь.

Крок 7.1.2 – Виведення повідомлення користувачу.

Користувачу виводиться відповідне повідомлення про те що файл який він бажає видалити, не знайдено.

Крок 7.2 – Так.

Якщо файл було знайдено, то виконується крок 8.

Крок 8 – Перевірка власності файлу.

Відбувається перевірка того, чи є користувач який намагається видалити файл, тим самим користувачем який його завантажив.

Крок 9 – Користувач є власником?

Крок 9.1 – Ні. Надсилання помилки про відсутність доступу.

Якщо користувач який намагається видалити файл, і користувач який його завантажив, є різними користувачами, то клієнту надсилається відповідна помилка.

Крок 9.1.1 – Отримання відповіді.

Клієнт отримує відповідь сервера.

Крок 9.1.2 – Вивід повідомлення про відсутність доступу до файлу.

Користувачу виводиться повідомлення про те що він не є власником файлу, може не може його видалити.

Крок 9.2 – Видалення файлу.

Якщо користувач який намагається видалити файл, і користувач який його завантажив, є одним і тим самим користувачем, то відбувається видалення файлу з бази даних.

Крок 10 – Надсилання відповіді клієнту.

Якщо файл успішно видалено, то клієнту надсилається відповідь.

Крок 11 – Отримання відповіді.

Клієнт отримує відповідь сервера.

Крок 12 – Вивід повідомлення про успішне видалення файлу.

Користувачу виводиться повідомлення про успішне видалення файлу.

Крок 13 – Кінець.

Таким чином, було побудований алгоритм видалення файлів з бази даних, з перевіркою власності.

Також для розробленого алгоритму видалення завантажених файлів, з перевіркою власності файлу0, було побудовано його блок-схему (рис. 2.6).

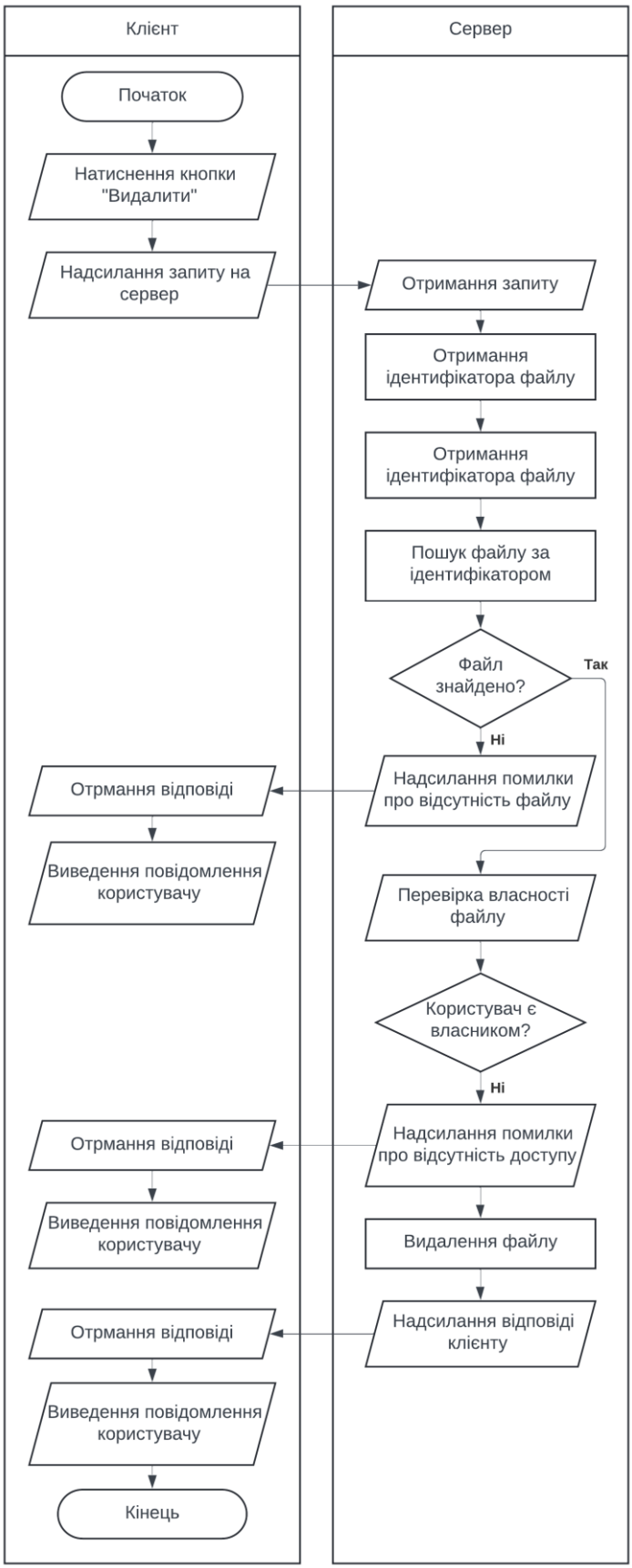


Рисунок 2.6 – розроблена блок-схема запропонованого алгоритму видалення файлів

Попередні 2 розроблені алгоритми відповідають за завантаження файлів в базу даних і видалення, з перевіркою власності, що забороняє видаляти файли тим користувачам, які не є їх власниками.

2.2 Проектування бази даних для захищеного корпоративного файлообмінника з двофакторною автентифікацією

Для побудови захищеного корпоративного файлообмінника, необхідно використовувати не менш надійну і захищену бузу даних. Так як в ній буде зберігатись як персональні дані зареєстрованих користувачів, так і самі файли які користувачі будуть завантажувати в файлообмінник. Тому для цього було обрано базу даних MongoDB.

MongoDB — це база даних, яка використовується для створення високодоступних і масштабованих інтернет-додатків. Завдяки гнучким підходом до схеми, вона популярна серед команд розробників, які використовують гнучкі методології. Пропонуючи драйвери для всіх основних мов програмування, MongoDB дозволяє негайно розпочати створення програми, не витрачаючи час на налаштування бази даних. Кожен запис у базі даних MongoDB є документом, описаним у BSON, двійковому представленні даних. Потім програми можуть отримати цю інформацію у форматі JSON [28].

Суттєвою перевагою при виборі бази даних MongoDB стало те що існує зручний застосунок MongoDB, який адаптований для різних операційних систем, і чудово працює як на MacOS, так і на Windows, це дозволяє зручно керувати вмістом бази даних, і самою базою даних.

Ще однією перевагою яка вплинула на вибір саме MongoDB, є те що вміст таблиць бази даних можна переглядати, і експортувати в будь якому зручному для себе форматі, а саме в форматі JSON, і вигляді списків і таблиць [29].

Для реалізації захищеного корпоративного файлообмінника, було прийнято рішення створити 2 таблиці в базі даних MongoDB, для збереження даних, це таблиця «Файли», і таблиця «Користувачі».

В таблиці «Користувачі», було створено наступні поля:

- id;

- ім’я;

- email;

- пароль;

- секретний ключ.

Перше поле ID, тут автоматично базою даних MongoDB буде генеруватись унікальний id ідентифікатор для кожного користувача який успішно зареєструвався.

Наступне поле ім’я, тут буде зберігатись ім’я користувача яке він вказав під час реєстрації.

В полі email, буде зберігатись email користувача який він вказав під час реєстрації.

Наступне поле пароль, тут буде зберігатись пароль користувача в захешованому вигляді, який він вказав під час реєстрації.

Останнє поле, секретний ключ, в якому буде зберігатися секретний ключ, згенерований кожному користувачу окремо під час реєстрації. Секретний ключ буде використовуватись для двофакторної автентифікації, а саме для генерації 6-ти значного токену.

В таблиці «Файли», було створено наступні поля:

- id;

- назва файлу;

- файл;

- опис;

- автор;

Перше поле ID, тут автоматично базою даних MongoDB буде генеруватись унікальний id ідентифікатор для кожного файлу який успішно було завантажено в базу даних.

Наступне поле назва файлу, сюди буде автоматично додаватись назва файлу (назва файлу на комп’ютері), який буде успішно завантажено.

В полі файл буде зберігатись, сам файл у закодованому вигляді.

Наступне поле опис, тут буде зберігатись опис до файлу, який користувач надасть під час завантаження файлу.

Останнє поле автор, тут буде автоматично додаватись ім’я користувача який завантажив даний файл.

Для наочного графічного представлення бази даних було побудовано ER-модель, так як вона візуалізує процеси сутності і зв’язків в середині спроектованої бази даних (рис. 2.7).

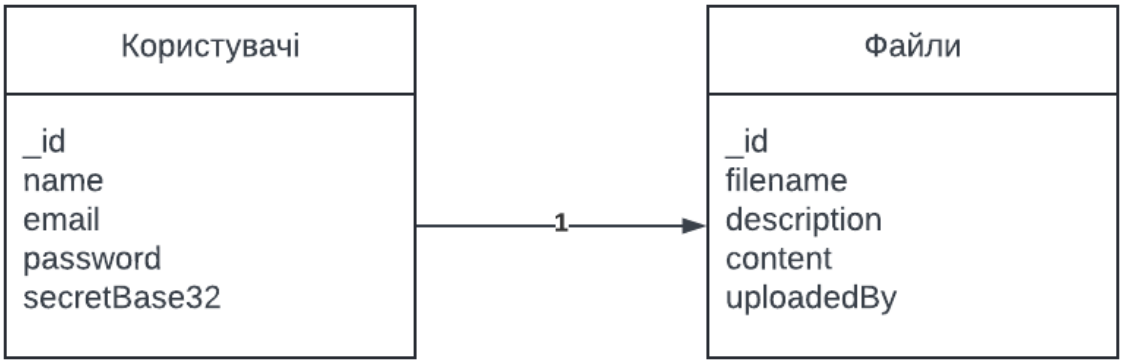


Рисунок 2.7 – розроблена ER-модель спроектованої бази даних

Було визначено 2 сутності розробленої бази даних, такі як: «Користувачі», і «Файли».

Для сутності «Користувачі», було визначено атрибути такі як: _id; name; email; password; secretBase32. Так як користувачі це основна сутність, яка зберігає інформацію про користувачів, які можуть завантажувати файли.

Для сутності «Файли», також було визначено атрибути такі як: _id; filename; description; content; uploadedBy. Так як сутність «Файли»,, яка зберігає інформацію про файли, завантажені користувачами. Кожен файл пов'язаний з одним користувачем через поле "Автор".

Зв’язок між сутностями «Користувачі», та «Файли» = 1, так як один користувач може завантажувати багато файлів, але кожен файл має одного автора.

Таким чином можна візуально спостерігати зв’язки між окремими таблицями розробленої бази даних, яка буде зберігати інформацію про користувачів, та файли.

2.3 Проектування інтерфейсу для захищеного корпоративного файлообмінника з двофакторною автентифікацією

В попередніх підрозділах були розроблені запропоновані алгоритми роботи системи, виходячи з того, захищений корпоративний файлообмінник з двофакторною автентифікацією, буде складатися з 3-х основних сторінок, таких як:

- сторінка реєстрації;
- сторінка авторизації;
- захищена сторінка завантаження файлів.

На сторінці реєстрації, користувач повинен мати змогу ввести особисті дані для реєстрації, а саме:

- ім’я;
- email;
- пароль.

Після введених даних, користувач повинен натиснути на кнопку «Зареєструватись», щоб розпочати процес реєстрації, тому дана кнопка також повинна бути в інтерфейсі користувача. Але якщо в користувача вже є обліковий запис, тому він повинен мати змогу перейти на сторінку авторизації, натиснувши кнопку «Увійти».

Виходячи з попереднього аналізу було спроектовано макет інтерфейсу сторінки реєстрації (рис. 2.8).

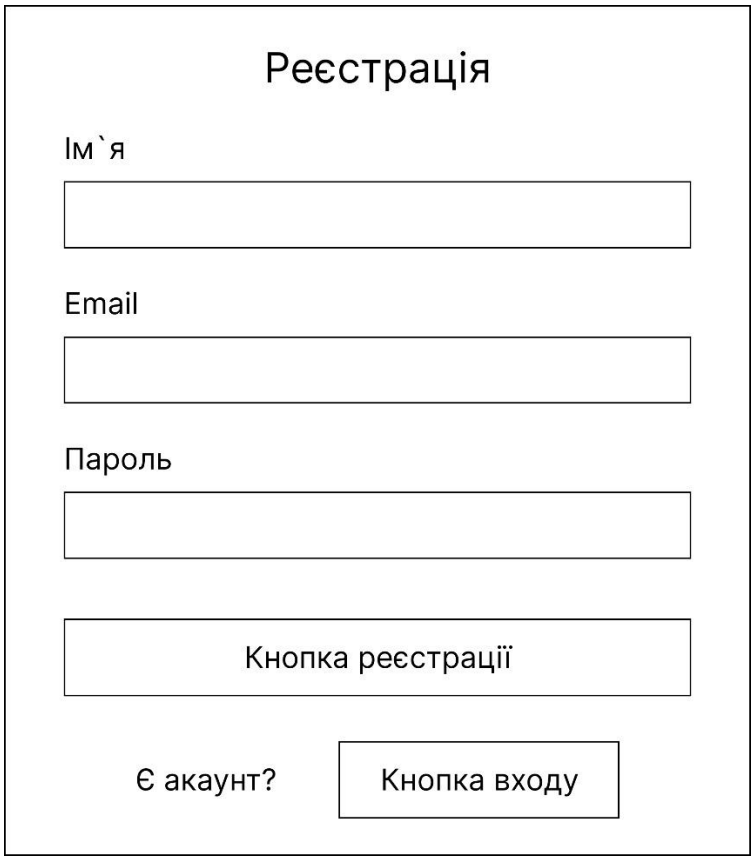


Рисунок 2.8 – спроектований макет інтерфейсу сторінки реєстрації

На сторінці авторизації користувач повинен мати змогу ввести особисті дані для авторизації, такі як:

- email;

- пароль;

- пароль на основі часу (для двофакторної автентифікації).

Після введених даних, користувач повинен натиснути на кнопку «Увійти», щоб розпочати процес авторизації, тому дана кнопка також повинна бути в інтерфейсі користувача. Але якщо в користувача ще немає облікового запису, тому він повинен мати змогу перейти на сторінку реєстрації, натиснувши кнопку «Реєстрація».

Виходячи з попереднього аналізу було спроектовано макет інтерфейсу сторінки авторизації (рис. 2.9).

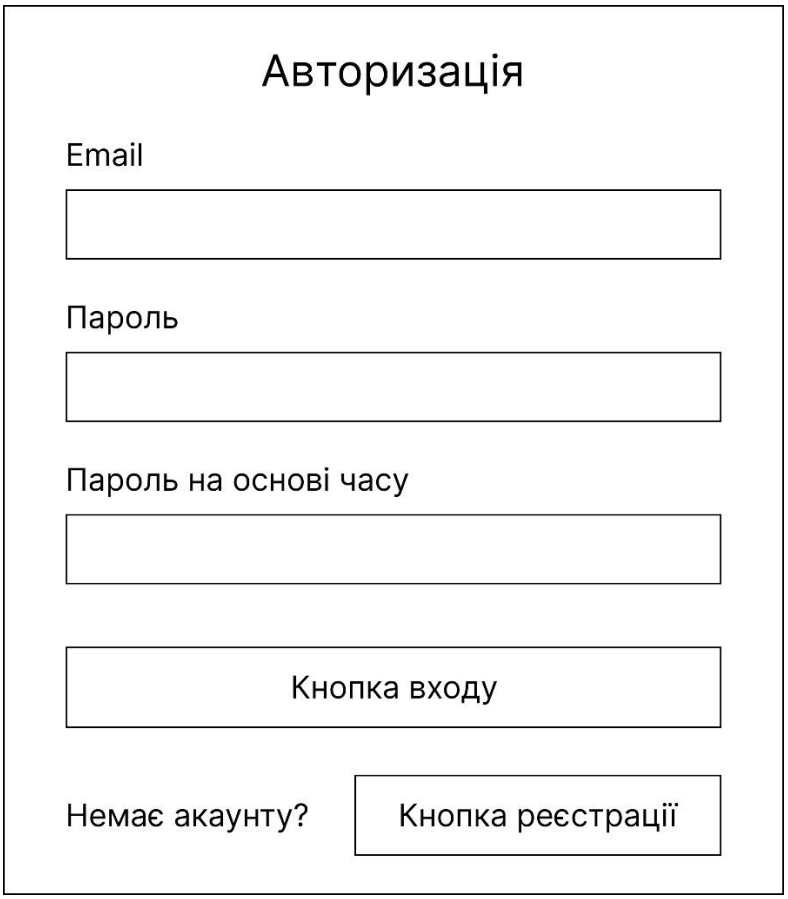


Рисунок 2.9 – спроектований макет інтерфейсу сторінки авторизації

Для захищеної сторінки завантаження файлів, необхідно надати можливість користувачам:

- обрати файл для завантаження;
- додати опис файлу.

Після заповнення попередніх полів користувач повинен натиснути на кнопку «Завантажити», яка також має бути в інтерфейсі захищеної сторінки завантаження файлів, для захищеного корпоративного файлообмінника з двофакторною автентифікацією.

Також необхідно додати поле куди будуть виводитися всі завантаженні файли, з можливість видалення натиснувши на кнопку «Видалити». Та має бути кнопка «Вихід», яка дає можливість користувачу вийти з власного облікового запису.

На основі попередньо проведеного аналізу, було спроектовано макет інтерфейсу захищеної сторінки завантаження файлів, для захищеного корпоративного файлообмінника з двофакторною автентифікацією (рис. 2.10).

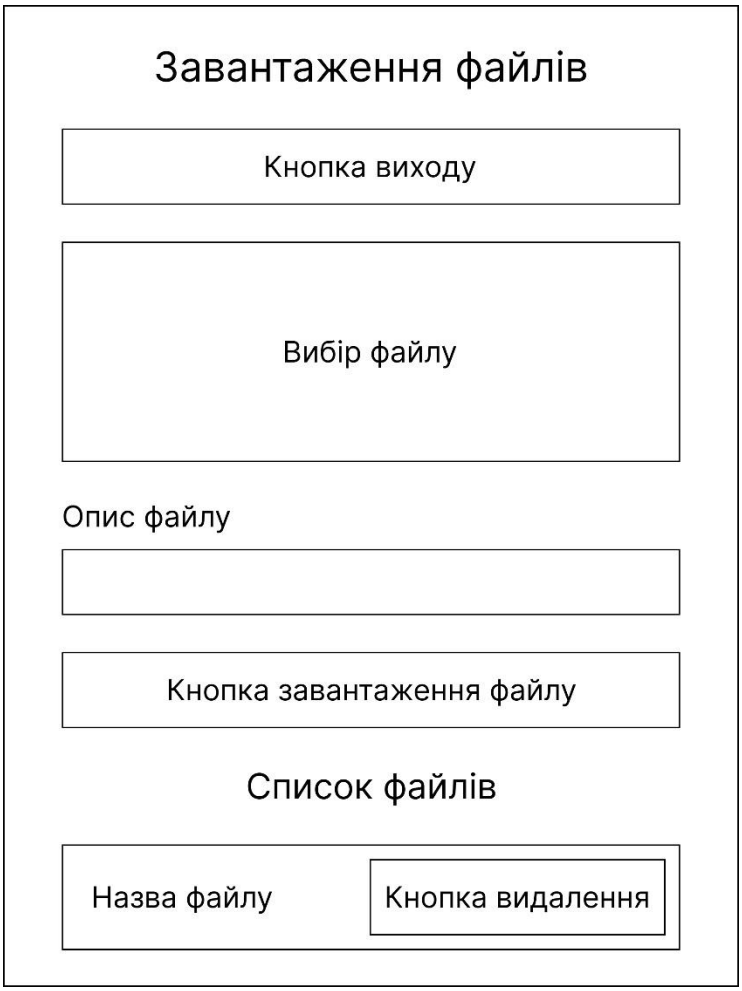


Рисунок 2.10 – спроектований макет інтерфейсу захищеної сторінки завантаження файлів

Також було спроектовано макет вікна повідомлень, з кнопкою згоди, натиснувши на яку, вікно закривається (рис. 2.11).

Рисунок 2.11 – спроектований макет вікна повідомлень

Таким чином, виходячи з раніше розроблених запропонованих алгоритмів роботи системи, було повністю спроектовано інтерфейс всіх сторінок захищеного корпоративного файлообмінника з двофакторною автентифікацією.

2.4 Висновки до розділу 2

В даному розділі було побудовано блок-схеми запропонованих алгоритмів роботи системи, а саме тих, які впливають на безпеку і захищеності корпоративного файлообмінника, алгоритми реєстрації, авторизації з додаванням сесій користувачів, алгоритм перевірки сесії користувачів, з наданням їм доступу до захищеної сторінки, та алгоритм виходу користувачів з облікових записів. Крім того було розроблено та побудовано блок-схеми запропонованих алгоритмів які відповідають за завантаження файлів в базу даних з клієнтського боку, та алгоритм видалення файлів з перевіркою власності файлу. Також було, спроектовано базу даних, та розроблено ER-модель зв’язків між окремими таблицями бази даних, такими як «Користувачі», та «Файли». На основі розроблених алгоритмів роботи системи, і проведеного детального аналізу, було спроектовано користувацький інтерфейс для кожної сторінки захищеного корпоративного файлообмінника з двофакторною автентифікацією.

Результатом даного розділу є розроблені запропоновані алгоритми функціонування захищеного корпоративного файлообмінника з двофакторною автентифікацією. Також розроблена база даних де буде зберігатись інформація про зареєстрованих користувачів, та завантажених файлів. Та спроектований користувацький інтерфейс.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО ЗАХИЩЕНОГО КОРПОРАТИВНОГО ФАЙЛООБМІННИКА З ДВОФАКТОРНОЮ АВТЕНТИФІКАЦІЄЮ

В даному розділі буде обрано і обґрунтовано мову програмування, фреймворки, бібліотеки та середовище розробки. Також буде практично реалізовано захищений корпоративний файлообмінник з двофакторною автенифікацією. В результаті розробки, буде виконано тестування розробленого захищеного корпоративного файлообмінника з двофакторною автентифікацією, особливу увагу буде зосереджено на тестуванні його захищеності.

3.1 Обґрунтування вибору мови програмування, фреймворків, бібліотек та середовища розробки

Після розробки алгоритмів роботи захищеного корпоративного файлообмінника з двофакторною автентифікацією, та вибору і проектування бази даних, необхідно визначити за допомогою якої мови програмування буде реалізований весь функціонал програми який був спроектований раніше. Саме це буде зроблено в даному підрозділі.

Для програмної реалізації захищеного корпоративного файлообмінника з двофакторною автентифікацією, було обрано мову програмування JavaScript, для реалізації клієнтської частини, і Node.js, для реалізації серверної частини.

JavaScript – це одна з найпопулярніших, об’єктно-орієнтованих мов програмування в світі, яка підтримується абсолютно всіма існуючими браузерами, так як побудована на рушії V8. JavaScript є досить гнучкою і універсальною мовою програмування [30].

Node.js - це потужне середовище виконання JavaScript коду з відкритим вихідним кодом, на сервері. Потужною перевагою Node.js є асинхронне програмування, що дозволяє швидко обробляти багато запитів одночасно, без блокування операцій які не були виконані. Це стало ключовою перевагою при виборі, так як в файлообміннику передбачається виконання багатьох команд, різними користувачами одночасно [31].

Ще однією перевагою Node.js, є його розвинена екосистема, яка включає в себе тисячі додаткових пакетів і модулів які можна досить швидко і просто встановити за допомогою вбудованого менеджера пакетів npm (Node Package Manager). Це дозволяє розробникам легко використовувати вже готові рішення, для вирішення важких завдань, тим самим суттєво прискорити процес розробки.

Node.js використовує єдину мову програмування JavaScript, це дозволяє розробникам використовувати єдину кодову базу як для клієнтської, так і для серверної частини додатку.

Для створення сервера було прийнято рішення використовувати фреймворк Express, так як віє є досить простий у застосуванні, та не пов’язує велику кількість правил і структур. Також суттєвою перевагою є те що у Express є дуже корисне розширення Middleware, яке дозволяє створювати сценарії обробки запитів перед тим, як вони досягнуть обробника маршруту. За допомогою Middleware, можна розробляти сценарії перевірки автентифікації користувачів, валідації даних логування запитів і так далі. Це робить Express відмінним вибором для розробки різноманітних веб-додатків, від простих API до складних веб-сервісів [32].

Для приховування деяких конфіденційних даних, таких як ключі API, паролі до бази даних чи інші налаштування, які не повинні бути включені в репозиторій було використано бібліотеку Dotenv, вона дозволяє завантажувати конфіденційні дані проекту з файлу .env в корені проекту та доступатися до них у додатку [33].

Однією з ключових переваг використання Dotenv є його простота в використанні, так як для приховання даних потрібно просто встановити дану бібліотеку, записати потрібні дані в файл .env, і визначити змінну в проекті.

Для підключення бази даних MongoDB, було використано бібліотеку Mongoose, так як вона є однією з найпопулярніших бібліотек для роботи з цією базою даних у середовищі Node.js [34].

Однією з ключових переваг використання бібліотеки Mongoose є його об'єктно-документне відображення (Object-Document Mapping, ORM), це

дозволяє використовувати об'єктно-орієнтований підхід до роботи з даними в базі даних MongoDB, замість прямого маніпулювання JSON-подібними об'єктами. Mongoose надає можливість легко описати схему даних, що спрощує валідацію, та зручний доступ до даних. Також Mongoose забезпечує структурованість даних у базі. Завдяки своїй гнучкості та потужним можливостям, Mongoose є важливим і потужним інструментом для розробників Node.js, які працюють з NoSQL базою даних MongoDB.

Для реалізації двофакторної автентифікації, було обрано бібліотеку Speakeasy, так як вона є досить потужним інструментом для генерації OTP (One-Time Password) в застосунках, які використовують двофакторну аутентифікацію.

При виборі бібліотеки, ключовим аргументом стало те що Speakeasy, бібліотека написана на мові програмування JavaScript, що відмінно поєднується з екосистемою Node.js, це робить її відмінним вибором для веб-додатків, побудованих на цій технології [35].

Ще однією перевагою Speakeasy є його простота у використанні та інтеграції в проект, так як бібліотека надає зручний API для генерації та перевірки OTP, що дозволяє впроваджувати двофакторну автентифікацію.

Бібліотеку Speakeasy, можна досить гнучко використовувати, так як вона підтримує різноманітні методи генерації OTP, включаючи базові методи, такі як (TOTP - Time-Based One-Time Password, HOTP - HMAC-Based One-Time Password) та розширені методи, такі як HMAC-SHA256 та HMAC-SHA512. Це дозволяє обрати саме те рішення яке буде найбільш оптимальним у вирішенні конкретного завдання.

Для створення сесій користувачів, було обрано бібліотеку Express-Session, так як ця бібліотека є основним компонентом фреймворка Express, який було використано, для створення серверу.

Express-Session дозволяє досить зручно зберігати стан сесій користувачів та дозволяє розробникам легко керувати інформацією про сесії між різними запитами. Використовуючи бібліотеку Express-Session, робота з сесіями в Express стає простішою та ефективнішою [36].

Також Express-Session, досить легко поєднується з іншими компонентами Express, такими як Middleware. Це суттєво спрощує роботу з сесіями у веб-додатках.

Для забезпечення конфіденційного збереження користувацьких паролів в базі даних, було прийнято рішення використовувати бібліотеку Bcryptjs, для хешування паролів.

Суттєвою перевагою бібліотеки Bcryptjs є те що вона написана на мові програмування JavaScript, це значить що вона дуже добре працює в забезпеченні конфіденційності паролів у поєднанні з середовищем Node.js.

Bcryptjs використовує адаптивний алгоритм хешування bcrypt, який станом на зараз є одним з найбільш надійних методів хешування паролів. Тому це забезпечує високий рівень безпеки для захисту паролів, від несанкціонованого доступу [37].

Крім того, бібліотека Bcryptjs, надає простий та зручний API для роботи з хешуванням паролів. Вона дозволяє легко створювати хеші паролів, перевіряти їх на відповідність та працювати зі сіллю для забезпечення додаткового рівня безпеки. Загалом, бібліотека bcryptjs є невід'ємною частиною екосистеми Node.js та забезпечує високий рівень безпеки для збереження та обробки паролів у веб-додатках.

Для обробки файлів які завантажуються на сервер, використовується бібліотека Multer. Це бібліотека використовується для веб-додатків на основі Node.js і фреймворка Express. Бібліотека Multer, також написана на мові програмування JavaScript, тому чудово з нею працює [38].

Multer дає можливість створювати маршрути для обробки файлів у веб-додатках, та додавати додаткові налаштування для завантаження, такі як максимальний розмір файлу, розширення і так далі. Також Multer дозволяє зберігати файли відразу в хмарних сховищах, таких як Amazon S3 або Google Cloud Storage.

Для програмної реалізації захищеного корпоративного файлообмінника з двофакторною автентифікацією, було обрано Visual Studio Code, як середовище розробки.

Visual Studio Code був представлений компанією Microsoft в 2015 році, і став одним із найпопулярніших редакторів коду, через свою зручність, водночас простоту і багатофункціональність [39].

Перевагами Visual Studio Code є те що він безкоштовний для будь яких користувачів які працюють на операційних системахWindows, Linux та MacOS. Ще однією перевагою є те що він підтримує безліч мов програмування, таких як JavaScript і всі його варіації, Python, Java, C#, PHP. Крім цих основних мов, VS Code також підтримує множину інших мов програмування, включаючи Ruby, Go, Swift, Kotlin та багато інших. Багатофункціональність та гнучкість VS Code роблять його відмінним інструментом для розробки.

Також перевагою Visual Studio Code є те що в ньому є безліч розширень під різноманітні завдання, які дозволяють розробникам налаштувати середовище розробки під себе. Можна встановити розширення для роботи з конкретними мовами програмування, фреймворками, системами контролю версій та багато іншого [40].

Крім того, VS Code має вбудовану підтримку Git, що робить спільну роботу над проектами більш зручною. Завдяки інтеграції з Git можна легко відстежувати зміни проекту, та співпрацювати з іншими розробниками безпосередньо з середовища розробки.

Таким чином було повністю підготовлено все до програмної реалізації захищеного корпоративного файлообмінника, який буде розроблено в наступному розділі даної дипломної роботи.

3.2 Програмна реалізація захищеного корпоративного файлообмінника з двофакторною автентифікацією

Для програмної реалізація захищеного корпоративного файлообмінника з двофакторною автентифікацією, перш за все було створено файл «app.js», в якому буде прописана вся серверна логіка.

Перш за все було встановлено фреймворк Express.js, після чого в створеному файлі «app.js», було створено сервер за допомогою даного фреймворка.

```
const express = require('express');
const app = express();
const PORT = 2220;
app.get('/', (req, res) => {
res.sendFile(path.join(__dirname, 'public', 'index.html'));});
app.listen(PORT, () => {
console.log(`Сервер працює і слухає порт ${PORT}`);});
```

Після цього було встановлено всі необхідні, додаткові модулі, бібліотеки і фреймворки, та оголошено їх залежності, які будуть використовуватися в подальшому.

```
const dotenv = require('dotenv');
const cors = require('cors');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const path = require('path');
const speakeasy = require('speakeasy');
const qrcode = require('qrcode');
const bcrypt = require('bcryptjs');
const multer = require('multer');
const session = require('express-session');
```

Наступним кроком було підключено базу даних MongoDB до проекту за допомогою бібліотеки Mongoose, але для приховування основного посилання для її підключення, його було приховано за допомогою додатково встановленої бібліотеки Dotenv, і файлу «.env».

```
dotenv.config();
mongoose.connect(process.env.MONGO_URI, {useNewUrlParser: true, useUnifiedTopology: true});
const db = mongoose.connection;
db.on('error', console.error.bind(console, 'Помилка підключення до бази даних MongoDB:'));
db.once('open', () => {
console.log('Підключено до бази даних MongoDB');});
```

Після чого було створено файл «index.html», де прописано клієнтський інтерфейс реєстрації користувачів корпоративного файлообмінника, з усіма необхідними полями вводу.

```
<div class="container-block">
<h1>Реєстрація користувачів</h1>
<form id="registrationForm">
<label for="name">Name:</label>
<input type="text" id="name" name="name" required><br>
<label for="email">Email:</label>
<input type="email" id="email" name="email" required><br>
<label for="password">Password:</label>
<input type="password" id="password" name="password" required><br>
<button type="submit">Register</button>
</form>
```

```
<p>Є акаунт? Увійдіть<a href="login.html">Signup</a></p>
</div>
```

Після чого в серверному файлі «app.js», було прописано масив allowedEmails, який містить електронні адреси, які мають право зареєструватися в системі.

```
const allowedEmails = [
'user01@gmail.com',
'user02@gmail.com',
'user03@ gmail.com'];
```

Наступним було прописано код який збирає введені дані з полів вводу на клієнтській стороні і відправляє запит реєстрації на сервер.

```
document.addEventListener('DOMContentLoaded', () => {
const registrationForm = document.getElementById('registrationForm');
registrationForm.addEventListener('submit', async (event) => {
event.preventDefault();
const name = document.getElementById('name').value;
const email = document.getElementById('email').value;
const password = document.getElementById('password').value;
try {
const response = await fetch('http://localhost:2220/register', {
method: 'POST',
headers: {
'Content-Type': 'application/json',},
body: JSON.stringify({ name, email, password }),});
const result = await response.json();
console.log(result);
alert(result.message);
} catch (error) {
console.error('Помилка при реєстрації:', error);}});
```

Після чого на стороні сервера, було прописано алгоритм обробки запитів реєстрації користувачів, які надходять з клієнтського боку, та схему збереження користувачів в базі даних.

```
app.post('/register', async (req, res) => {
const {name, email, password} = req.body;
try {
if (!allowedEmails.includes(email)) {
return res.status(400).json({success: false, message: 'Даний email не дозволено для реєстрації'});}
const existingUser = await User.findOne({email});
if (existingUser) {
return res.status(400).json({success: false, message: 'Користувач з таким email вже зареєстрований'});}
const hashedPassword = await bcrypt.hash(password, 10);
const secret = speakeasy.generateSecret({length: 20});
const secretBase32 = secret.base32;
const newUser = new User({name, email, password: hashedPassword, secretBase32});
await newUser.save();
res.json({success: true, message: `Користувач ${name} зареєстрований успішно ${secretBase32}`});
} catch (error) {
console.error('Error during registration:', error);
res.status(500).json({success: false, message: 'Помилка реєстрації користувача'});}});
const userSchema = new mongoose.Schema({
name: String,
email: String,
```

```
password: String,
secretBase32: String,});
const User = mongoose.model('User', userSchema);
```

Даний алгоритм містить всі необхідні перевірки, хешування пароля, генерація токену яка реалізована за допомогою бібліотеки Speakeasy, збереження користувачів в базі даних, та надсилання відповіді клієнту.

Після чого було створено файл «login.html», де прописано клієнтський інтерфейс авторизації, з усіма необхідними полями вводу для авторизації.

```
<div class="container-block">
<h1>Авторизація</h1>
<form id="loginForm">
<label for="email">Email:</label>
<input type="email" id="email" name="email" required><br>
<label for="password">Password:</label>
<input type="password" id="password" name="password" required><br>
<label for="secterkey">Secter key:</label>
<input type="text" id="secterkey" name="secterkey" required><br>
<button type="submit" id="login-button">Login</button>
</form>
</div>
```

Та в цьому ж файлі було прописано код який збирає введені дані з полів вводу на клієнтській стороні і відправляє запит авторизації на сервер.

```
document.addEventListener('DOMContentLoaded', () => {
const loginForm = document.getElementById('loginForm');
loginForm.addEventListener('submit', async (event) => {
event.preventDefault();
const email = document.getElementById('email').value;
const password = document.getElementById('password').value;
const secterkey = document.getElementById('secterkey').value;
try {
const response = await fetch('http://localhost:2220/login', {
method: 'POST',
headers: {
'Content-Type': 'application/json',},
body: JSON.stringify({ email, password, secterkey }),});
const result = await response.json();
console.log(result);
alert(result.message);
if (result.success) {
window.location.href = 'upload.html';}
} catch (error) {
console.error('Error during login:', error);}});
```

Наступним було реалізовано серверний алгоритм обробки запитів авторизації користувачів.

```
app.post('/login', async (req, res) => {
const { email, password, secterkey } = req.body;
try {
const existingUser = await User.findOne({ email });
if (!existingUser) {
return res.status(401).json({ success: false, message: 'Неправильна електронна адреса або пароль' });}
const isPasswordValid = await bcrypt.compare(password, existingUser.password);
```

```
if (!isPasswordValid) {
return res.status(401).json({ success: false, message: 'Неправильна електронна адреса або пароль' });}
function verifyToken(secretKey, token) {
return speakeasy.totp.verify({ secret: secretKey, encoding: 'base32', token: token });}
const isTokenValid = verifyToken(existingUser.secretBase32, secterkey);
if (isTokenValid) {
const user = { ...existingUser.toObject(), isPasswordValid};
req.session.user = user;
console.log("Session set:", req.session.user);
return res.json({ success: true, message: Ви успішно увійшли в свій обліковий запис });
} else {
return res.status(401).json({ success: false, message: 'Неправильний токен' });}
} catch (error) {
console.error('Error during login:', error);
res.status(500).json({ success: false, message: 'Помилка авторизації користувача' });}});
```

Даний алгоритм містить всі необхідні перевірки для авторизації, раніше зареєстрованих користувачів, з надсиланням відповіді клієнту.

Для того щоб після успішної авторизації користувачів створювалась сесія, було прописано відповідний код, який використовує бібліотеку Express-Session.

```
app.use(session({
secret: 'secret',
resave: false,
saveUninitialized: true,}));
```

Після цього було прописано клієнтський код, який відповідає за надсилання запиту на вихід з облікового запису.

```
document.getElementById('logoutBtn').addEventListener('click', function() {
fetch('/logout')
.then(response => response.json())
.then(data => {
if (data.success) {
window.location.href = '/login.html';
} else {
console.error('Error destroying session:', data.message);}})
.catch(error => {
console.error('Error destroying session:', error);});});
```

Потім було прописано серверний алгоритм виходу користувачів з свого облікового запису.

```
app.get('/logout', (req, res) => {
req.session.destroy(error => {
if (error) {
console.error('Error destroying session:', error);
res.status(500).json({ success: false, message: 'Помилка видаленя сесії' });
} else {
res.json({ success: true, message: 'Сесію успішно видалено' });}});});
```

Після цього було створено файл «upload.html», прописано клієнтський інтерфейс завантаження файлів, тобто інтерфейс головної сторінки корпоративного файлообмінника.

```
<div class="container-block">
<h1>Завантаження файлів</h1>
```

```
<button id="logoutBtn">Logout</button>
<form id="fileupload">
<input type="file" id="file" name="file" required><br>
<label for="email">Description:</label>
<input type="text" id="description" name="description" required><br>
<button type="submit">Upload</button>
</form>
<div class="file-list">
<h2>Список файлів</h2>
<ul id="files"></ul>
</div>
</div>
```

Наступним було прописано клієнтський код, який при завантаженні сторінки відправляє запит на сервер для перевірки сесії користувача, і надання доступу до захищеної сторінки корпоративного файлообмінника.

```
window.onload = function() {
fetch('/checkSession')
.then(response => response.json())
.then(data => {
if (!data.success) {
alert("Спочатку потрібно увійти");
window.location.href = '/login.html';}})
.catch(error => {
console.error('Помилка перевірки сесії', error);});};
```

Потім було створено на сервері алгоритм самої перевірки сесії

```
app.get('/checkSession', (req, res) => {
if (req.session.user) {
res.json({ success: true, message: 'Сесія активна' });
} else {
res.json({ success: false, message: 'Сесія не активна' });}});
```

Після цього в клієнтському файлі було прописано код який відповідає за збір всіх даних з клієнтського боку, і відправкою запиту для обробки на сервер, та вивід завантажених файлів на сторінку.

```
document.addEventListener('DOMContentLoaded', () => {
const fileupload = document.getElementById('fileupload');
const fileList = document.getElementById('files');
fileupload.addEventListener('submit', async (event) => {
event.preventDefault();
const file = document.getElementById('file').files[0];
const description = document.getElementById('description').value;
try {
const formData = new FormData();
formData.append('file', file);
formData.append('description', description);
const response = await fetch('http://localhost:2220/upload', {
method: 'POST',
body: formData,});
const result = await response.json();
console.log(result);
alert(result.message);
window.location.reload();
} catch (error) {
console.error('Error during file upload:', error);}});
```

```
async function fetchFiles() {
const response = await fetch('http://localhost:2220/files');
const files = await response.json();
files.forEach(file => {
const listItem = document.createElement('li');
const link = document.createElement('a');
const img = document.createElement('img');
img.src = 'icons8-file-48.png';
link.textContent = ${file.description} (Uploaded by: ${file.uploadedBy});
link.href = /download/${file._id};
link.setAttribute('download', file.filename);
listItem.appendChild(img);
listItem.appendChild(link);
const deleteBtn = document.createElement('button');
deleteBtn.classList.add('deleteBtn');
deleteBtn.textContent = 'Видалити';
deleteBtn.dataset.fileId = file._id;
data-file-id
listItem.appendChild(deleteBtn);
fileList.appendChild(listItem);
fileList.insertBefore(listItem, fileList.firstChild);});}
fetchFiles();});
```

Після чого на серверній стороні, було прописано модель збереження файлів в базі даних.

```
const File = mongoose.model('File', {
filename: String,
description: String,
content: Buffer,
uploadedBy: String});
```

Після чого було прописано серверний алгоритм обробки запитів, завантаження файлів.

```
app.post('/upload', upload.single('file'), async (req, res) => {
try {
const uploadedBy = req.session.user.name;
const newFile = new File({
filename: req.file.originalname,
description: req.body.description,
content: req.file.buffer,
uploadedBy: uploadedBy});
await newFile.save();
res.json({success: true, message: 'Ви успішно завантажили файл'});
} catch (error) {
console.error(error);
res.status(500).send('Помилка завантаження файлу');}});
```

Також було прописано клієнтський код який відповідає за надсилання запиту на сервер про видалення файлів.

```
document.addEventListener('click', async (event) => {
if (event.target.classList.contains('deleteBtn')) {
const fileId = event.target.dataset.fileId;
try {
const response = await fetch(`http://localhost:2220/files/${fileId}`, {
method: 'DELETE',});
const result = await response.json();
if (result.success) {
```

```
event.target.parentNode.remove();
alert(result.message);
} else {
alert('Помилка видалення файлу');}
} catch (error) {
console.error('Помилка видалення файлу:', error);
alert('Помилка видалення файлу');}}});
```

Після чого було прописано серверний алгоритм, який обробляє запит на видалення файлів.

```
app.delete('/files/:id', async (req, res) => {
try {
const fileId = req.params.id;
const file = await File.findById(fileId);
if (!file) {
return res.status(404).json({ success: false, message: 'Файл не знайдено' });}
if (file.uploadedBy !== req.session.user.name) {
return res.status(403).json({ success: false, message: 'Ви не маєте доступу до видалення цього файлу' });}
const deletedFile = await File.findByIdAndDelete(fileId);
if (!deletedFile) {
return res.status(404).json({ success: false, message: 'Файл не знайдено' });}
res.json({ success: true, message: 'Файл успішно видалено' });
} catch (error) {
console.error('Помилка видалення файлу:', error);
res.status(500).json({ success: false, message: 'Помилка видалення файлу' });}});
```

Таким чином було практично реалізовано захищений корпоративний файлообмінник з двофакторною автентифікацією, за допомогою мови програмування JavaScript і Node.js, та додаткових модулів, бібліотек і фреймворків.

3.3 Тестування розробленого захищеного корпоративного файлообмінника з двофакторною автентифікацією

Заключним етапом в розробці захищеного корпоративного файлообмінника з двофакторною автентифікацією, є його тестування.

Для початку проведення тестування розробленого захищеного корпоративного файлообмінника, було запущено сервер (рис. 3.1).

```
vitaliivolos@MacBook-Air-Vitalij test-3 % node app.js
(node:21967) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effe
ct since Node.js Driver version 4.0.0 and will be removed in the next major version
(Use `node --trace-warnings ...` to show where the warning was created)
(node:21967) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has n
o effect since Node.js Driver version 4.0.0 and will be removed in the next major version
Сервер працює і слухає порт 2220
Підключено до бази даних MongoDB
```

Рисунок 3.1 – запущений сервер

Сервер працює і базу даних було підключено правильно.

Після чого було проведено тестування доступу до захищеної сторінки неавторизованим користувачам (рис. 3.2).

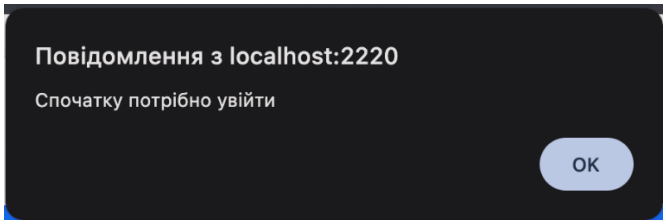


Рисунок 3.2 – повідомлення з розробленого захищеного корпоративного файлообмінника з двофакторною автентифікацією

Доступ неавторизованим користувачам до захищеної сторінки заборонено, при натиснені на кнопку «ОК», автоматично відкривається сторінка авторизації.

Після чого було проведено тестування реєстрації, для цього на відповідній сторінці, було введено дані для реєстрації з електронною адресою, якої немає в списку дозволених для реєстрації (рис. 3.3).

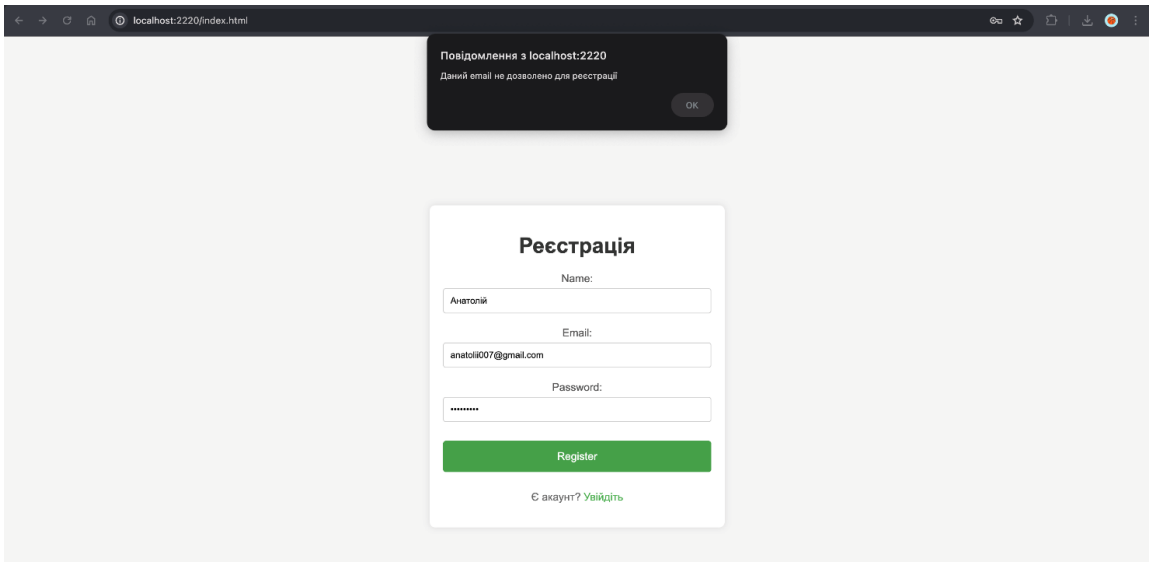


Рисунок 3.3 – зовнішній вигляд розробленого захищеного корпоративного файлообмінника з двофакторною автентифікацією

Алгоритм перевірки дозволених електронних адрес, спрацював чудово, і не дозволив зареєструватися даному користувачу.

Після цього було проведено тестування реєстрації з електронною адресою яка є в списку дозволених (рис. 3.4).

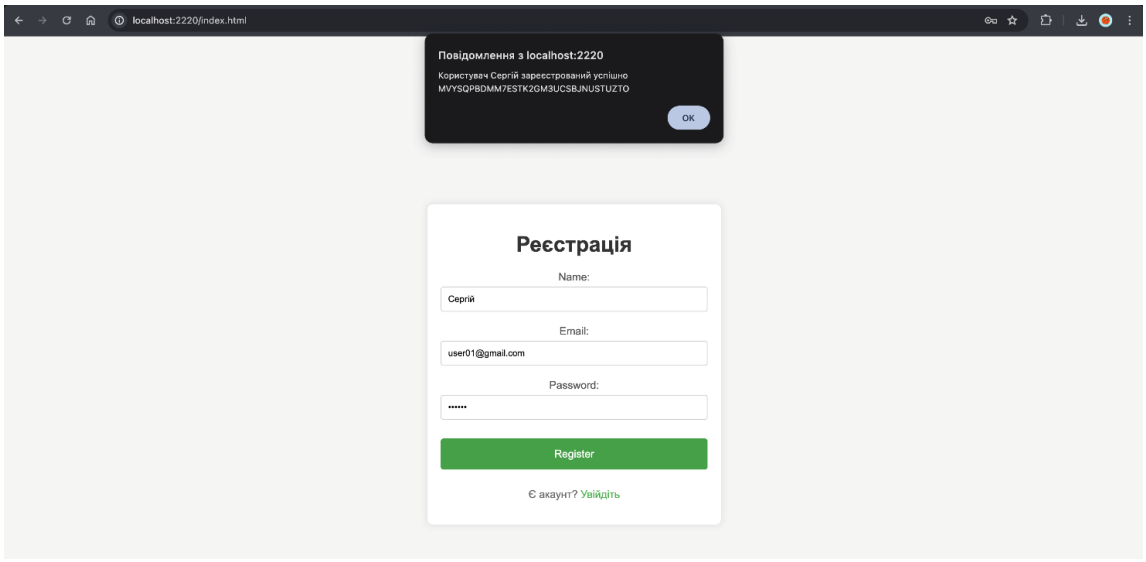


Рисунок 3.4 – зовнішній вигляд розробленого захищеного корпоративного файлообмінника з двофакторною автентифікацією

Користувач який дозволений для реєстрації, успішно зареєстрований, йому було надане відповідне повідомлення з токеном для двофакторної автентифікації.

Для перевірки чи дійсно користувач зареєструвався, було перевірено базу даних (рис. 3.5).

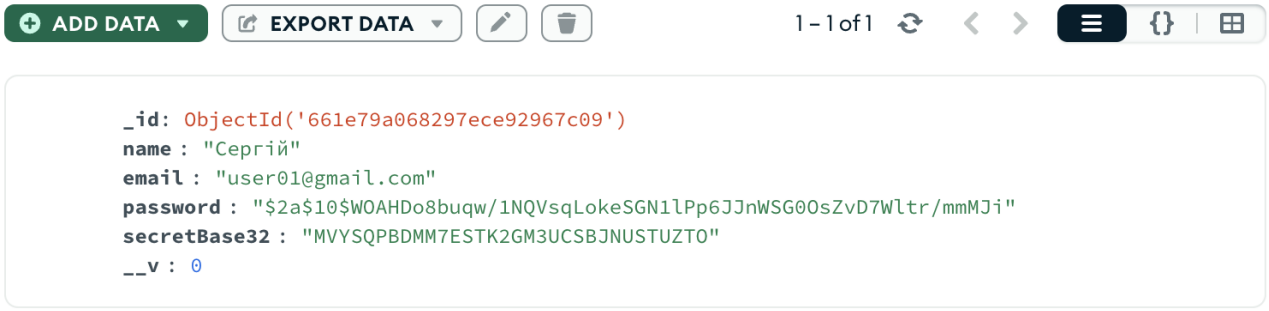


Рисунок 3.5 – зареєстрований користувач, збереження в базі даних

Користувача дійсно було успішно зареєстровано, з збереженням його імені, електронної адреси, захешованого пароля, і токеном для двофакторної автентифікації.

Після чого було проведено тестування на повторну реєстрацію зареєстрованого користувача, для цього було введено ту ж саму електронну адресу яка вже зареєстрована (рис.3.6).

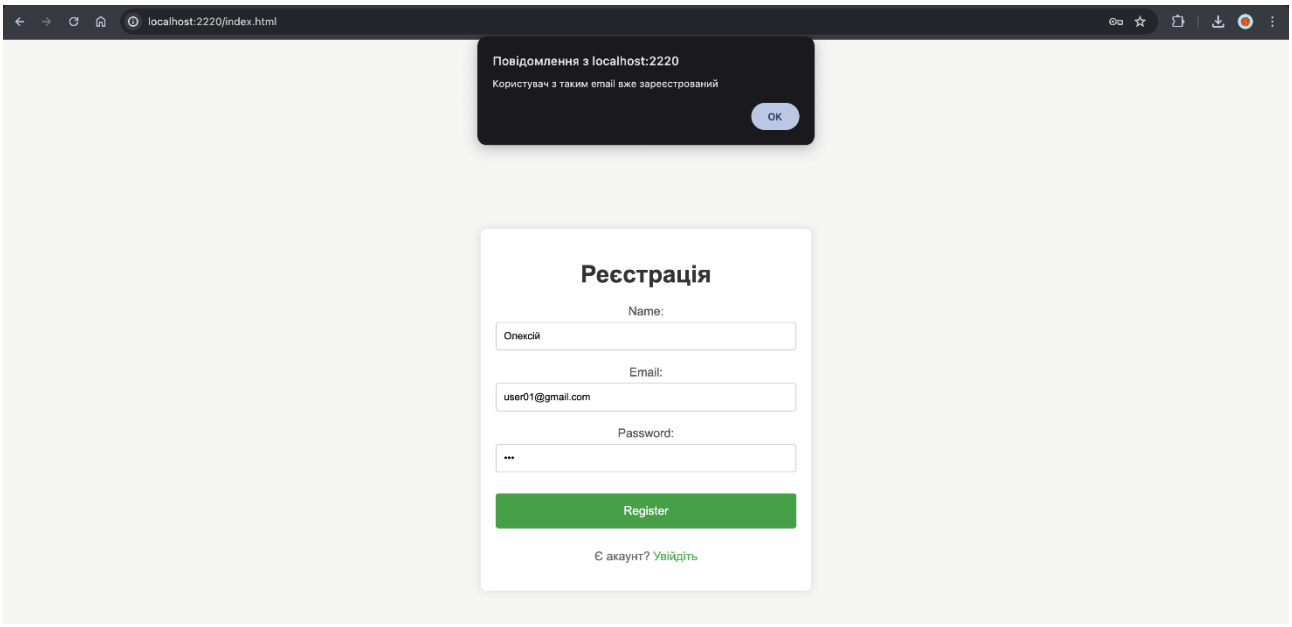


Рисунок 3.6 – зовнішній вигляд розробленого захищеного корпоративного файлообмінника з двофакторною автентифікацією

Користувача який уже зареєстрований не допущено до повторної реєстрації, йому виведено відповідне повідомлення.

Наступним було проведено тестування авторизації користувача, для цього було додано токен двофакторної автентифікації в застосунок Google Authenticator, для генерації тимчасового ключа (рис. 3.7) [41-42].

Рисунок 3.7 – тимчасовий ключ в програмі Google Authenticator, для двофакторної автентифікації

Після чого було здійснено спробу авторизації раніше зареєстрованого користувача, з використанням даного ключа (рис. 3.8).

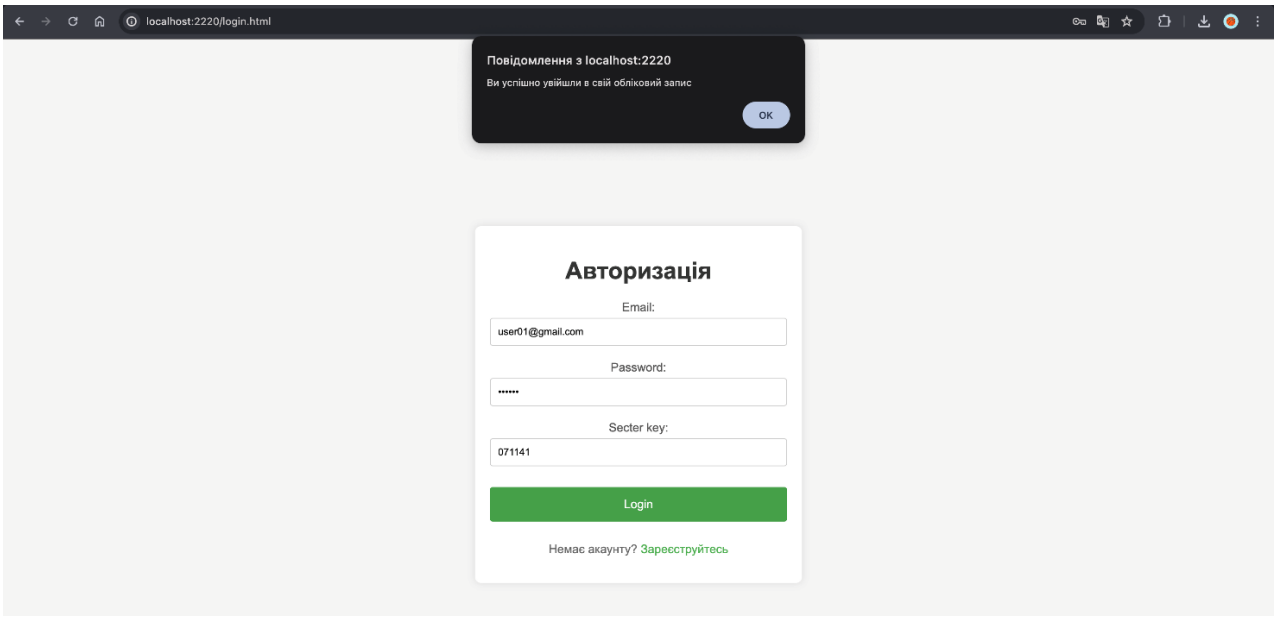


Рисунок 3.8 – зовнішній вигляд розробленого захищеного корпоративного файлообмінника з двофакторною автентифікацією

В результаті користувач успішно авторизувався, також йому було виведено відповідне повідомлення. Після успішної авторизації користувачу було створено сесію (рис. 3.9).

```
Session set: {
  _id: new ObjectId('661e79a068297ece92967c09'),
  name: 'Сергій',
  email: 'user01@gmail.com',
  password: '$2a$10$WOAHDo8buqw/1NQVsqLokeSGN1lPp6JJnWSG0OsZvD7Wltr/mmMJi',
  secretBase32: 'MVYSQPBDMM7ESTK2GM3UCSBJNUSTUZTO',
  __v: 0,
  isPasswordValid: true
}
```

Рисунок 3.9 – створена сесія користувача

Після успішної авторизації автоматично відкривається захищена сторінка корпоративного файлообмінника, де зберігаються самі файли (рис. 3.10).

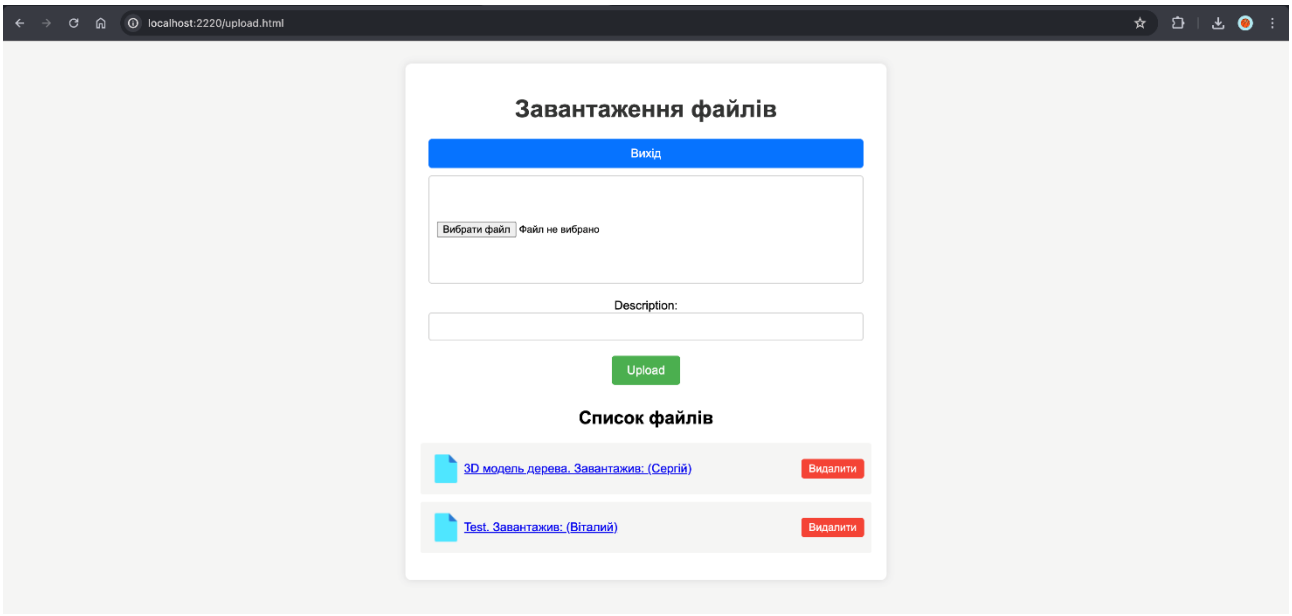


Рисунок 3.10 – зовнішній вигляд розробленого захищеного корпоративного файлообмінника з двофакторною автентифікацією

Наступним було проведено тестування завантаження файлів, для цього було обрано файл, додано йому опис, і натиснено відповідну кнопку для завантаження (рис. 3.11).

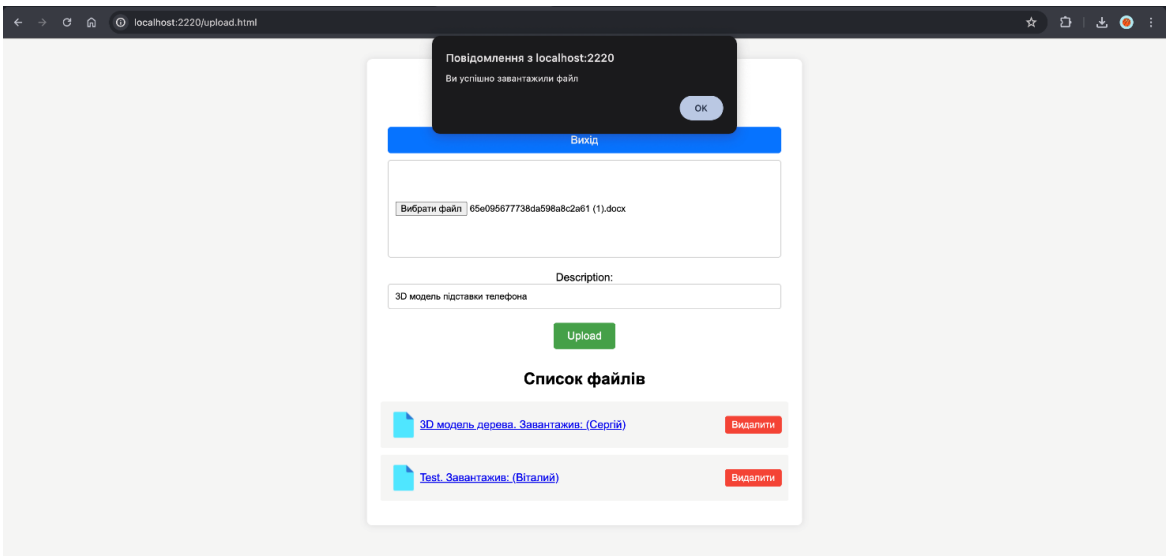


Рисунок 3.11 – зовнішній вигляд розробленого захищеного корпоративного файлообмінника з двофакторною автентифікацією

Файл успішно завантажено, і додано в базу даних (рис. 3.12).

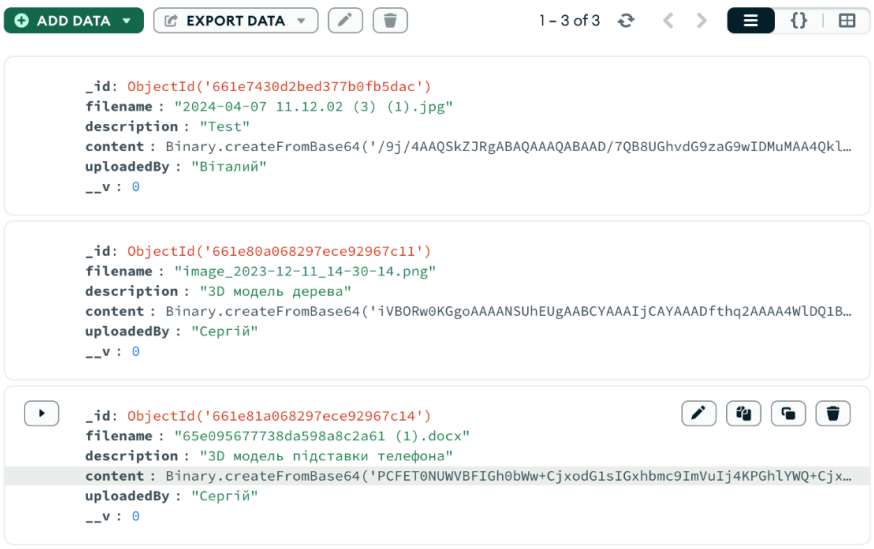


Рисунок 3.12 – збережений файл в базі даних

Аналізуючи результати проведених тестувань, можна дійти висновку що розроблений захищений корпоративний файлообмінник з двофакторною автентифікацією, працює бездоганно, забезпечуючи безпечний обмін конфіденційними файлами 3D моделей між працівниками підприємства яке займається 3D друком.

3.4 Висновок до розділу 3

В даному розділі було обрано і обґрунтовано мову програмування, фреймворки, бібліотеки та середовище розробки. Було програмно реалізовано захищений корпоративний файлообмінник з двофакторною автентифікацією, а саме було розроблено як клієнтську, так і серверну частини. Також було проведено тестування розробленого захищеного корпоративного файлообмінника, та сформовано висновок за результатами тестувань.

В результаті було отримано повноцінний, абсолютно робочий і готовий до використання захищений корпоративний файлообмінник з двофакторною автетифікацією.

Даний корпоративний файлообмінник, допоможе підприємству яке займається 3D друком, підвищити рівень захисту 3D моделей, які є їх власністю, під час обміну між інженерами.

ВИСНОВКИ

Отже, в ході написання даної бакалаврської роботи, було здійснено огляд і аналіз існуючих на ринку, сучасних корпоративних файлообмінників, проаналізовано їх переваги та недоліки, сформовано відповідні висновки. Також було визначено і сформовано поняття автентифікації і двофакторної автентифікації, та визначено відмінність від поняття авторизації. Було здійснено аналіз сучасних методів двофакторної автентифікації, таких як одноразові коди які надходять у вигляді SMS, електронною поштою, чи дзвінком, одноразові коди з програми автентифікації, та біометричні методи двофакторної автентифікації, такі як відбиток пальця, обличчя, або голосу, проаналізовано їх переваги та недоліки.

Було розроблено запропоновані алгоритми роботи системи захищеного корпоративного файлообмінника з двофакторною автентифікацією, а саме алгоритм реєстрації, авторизації з додаванням сесій користувачів, алгоритм перевірки сесії користувачів, з наданням їм доступу до захищеної сторінки, та алгоритм виходу користувачів з облікових записів. Також було розроблено алгоритми які відповідають за завантаження в базу даних з клієнтського боку, та алгоритм видалення файлів з перевіркою власності файлу. Крім того було побудовано блок-схеми даних алгоритмів. Також було обрано і спроектовано базу даних для збереження даних користувачів і завантажених файлів. Крім того було спроектовано користувацький інтерфейс, на основі аналізу розроблених алгоритмів роботи захищеного корпоративного файлообмінника з двофакторною автентифікацією.

Також було обрано і обґрунтовано мову програмування, фреймворки, бібліотеки, та середовище розробки, за допомогою яких буде програмно реалізовано корпоративний файлообмінник. Крім того, було програмно реалізовано захищений корпоративний файлообмінник з двофакторною автентифікацією, а саме було розроблено як клієнтську, так і серверну частини, з зручним інтерфейсом. Також було проведено тестування розробленого

захищеного корпоративного файлообмінника, особливу увагу було зосереджено на захищеності розробленої системи. В результаті проведених тестувань, було сформовано відповідні висновок.

В результаті виконаної роботи було досягнуто запланованого результату, а саме, отримано повноцінний, повністю працездатний і готовий до роботи, захищений корпоративний файлообмінник з двофакторною автентифікацією, для забезпечення захищеного обміну конфіденційними файлами 3D моделей між працівниками підприємства яке займається 3D друком.

ПЕРЕЛІК ПОСИЛАНЬ

1. Визначення файлообмінника, або файлового хостингу. URL: https://dlan.dp.ua/ua/filesoobmenik (Дата звернення: 14.04.2024).

2. Dropbox. Загальні відомості. URL: https://vchymo.com/application/Dropbox (Дата звернення: 29.04.2024).

3. Синхронізація Dropbox. URL: https://techcrunch.com/2008/03/11/dropbox-the-online-storage-solution-weve-been-waiting-for (Дата звернення: 29.04.2024).

4. Переваги Dropbox. URL: https://vchymo.com/application/Dropbox (Дата звернення: 17.04.2024).

5. Пробний період будь якого тарифу Dropbox. URL: https://www.dropbox.com/try/teams (Дата звернення: 18.04.2024).

6. Безкоштовний тарифний план Dropbox. URL: https://www.dropbox.com/plans (Дата звернення: 17.04.2024).

7. Офіційний сайт Google Drive. URL: https://myaccount.google.com (Дата звернення: 25.04.2024).

8. Інструкція з користування і налаштування Google Drive. URL: https://www.cloudwards.net/how-does-google-drive-work (Дата звернення: 03.04.2024).

9. Офіційний сайт OneDrive. URL: https://www.microsoft.com/uk-ua/microsoft-365/onedrive/online-cloud-storage (Дата звернення: 23.03.2024).

10. Основні відомості щодо поняття автентифікації. URL: https://www.techtarget.com/searchsecurity/definition/authentication (Дата звернення: 15.03.2024).

11. Основні поняття авторизації. URL: https://auth0.com/intro-to-iam/what-is-authorization (Дата звернення: 15.03.2024).

12. Відмінність між авторизацією і автентифікацією. URL: https://www.sailpoint.com/identity-library/difference-between-authentication-and-

authorization/#:~:text=Simply%20put%2C%20authentication%20is%20the,people%20can%20come%20on%20board (Дата звернення: 04.04.2024).

13. Контроль доступу. URL: https://www.ibm.com/docs/en/wca/3.0.0?topic=security-authentication-versus-access-control (Дата звернення: 09.04.2024).

14. Важливість використання автентифікації в організаціях. URL: https://nordvpn.com/uk/blog/what-is-user-authentication/#:~:text=User%20authentication%20is%20important%20because,both%20private%20and%20business%20accounts (Дата звернення: 09.04.2024).

15. Основні завдання автентифікації. URL: https://www.ibm.com/docs/en/atlas-policy-suite/6.0.3?topic=tasks-authentication-task (Дата звернення: 03.04.2024).

16. Основні поняття, і визначення двофакторної автентифікації. URL: https://www.techtarget.com/searchsecurity/definition/two-factor-authentication (Дата звернення: 03.05.2024).

17. Фактор знань, в двофакторній автентифікації. URL: https://www.ibm.com/topics/2fa#:~:text=In%20most%202FA%20implementations%2C%20a,security%20questions%20are%20also%20typical (Дата звернення: 22.04.2024).

18. Фактор володінь в двофакторній автентифікації. URL: https://www.techtarget.com/searchsecurity/definition/two-factor-authentication#:~:text=A%20possession%20factor%20is%20something,in%20the%20user's%20physical%20self (Дата звернення: 22.04.2024).

19. Фактор приналежності в двофакторній автентифікації. URL: https://www.paidmembershipspro.com/two-factor-authentication/ (Дата звернення: 18.03.2024).

20. Фактор розташування в двофакторній автентифікації. URL: https://duo.com/product/multi-factor-authentication-mfa/two-factor-authentication-2fa#:~:text=Location%20Factor,identity%20of%20the%20new%20user (Дата звернення: 17.03.2024).

21. Фактор часу в двофакторній автентифікації. URL: https://help.peopleforce.io/en/articles/6628189-two-factor-authentication (Дата звернення: 12.04.2024).

22. Метод двофакторної автентифікації на основі одноразових кодів підтвердження. URL: https://www.kaspersky.com/blog/types-of-two-factor-authentication/48446/ (Дата звернення: 03.04.2024).

23. Метод двофакторної автентифікації за допомогою спеціального тимчасового, одноразового коду з програми автентифікації. URL: https://www.simplilearn.com/tutorials/cyber-security-tutorial/what-is-two-factor-authentication (Дата звернення: 26.04.2024).

24. Метод двофакторної автентифікації на основі біометричних даних. URL: https://jumpcloud.com/blog/biometric-totp-2fa#:~:text=Biometric%202FA%2C%20or%20biometric%20authentication,depresses%20keys%20on%20their%20keyboard (Дата звернення: 13.04.2024).

25. Результати тестування роботи систем Face ID, від компанії Apple. URL: https://support.apple.com/en-us/102381 (Дата звернення: 03.05.2024).

26. Результати тестування роботи систем Touch ID, від компанії Apple. URL: https://support.apple.com/en-us/105095 (Дата звернення: 01.05.2024).

27. Правила, і принципи побудови блок-схеми. URL: https://nulab.com/learn/design-and-ux/keep-it-simple-follow-flowchart-rules-for-better-diagrams/ (Дата звернення: 10.04.2024).

28. Офіційний сайт бази даних MongoDB. URL: https://www.mongodb.com/why-use-mongodb#:~:text=MongoDB%20is%20a%20document%20database,development%20teams%20using%20agile%20methodologies (Дата звернення: 08.04.2024).

29. Переваги бази даних MongoDB. URL: https://www.mongodb.com/advantages-of-mongodb (Дата звернення: 07.04.2024).

30. Опис мови програмування JavaScript. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (Дата звернення: 07.04.2024).

31. Офіційний сайт мови програмування Node.js. URL: https://nodejs.org/en (Дата звернення: 08.04.2024).

32. Інструкція з використання вреймворка Express. URL: https://expressjs.com/uk/ (Дата звернення: 26.04.2024).

33. Інструкція з встановлення, використання і налаштування бібліотеки Dotenv. URL: https://www.npmjs.com/package/dotenv (Дата звернення: 28.03.2024).

34. Офіційний сайт бібліотеки Mongoose. URL: https://mongoosejs.com/ (Дата звернення: 07.05.2024).

35. Інструкція з встановлення, використання і налаштування бібліотеки Speakeasy. URL: https://www.npmjs.com/package/speakeasy (Дата звернення: 23.04.2024).

36. Інструкція з встановлення, використання і налаштування бібліотеки Express-Session. URL: https://www.npmjs.com/package/express-session (Дата звернення: 24.04.2024).

37. Інструкція з встановлення, використання і налаштування бібліотеки Bcryptjs. URL: https://www.npmjs.com/package/bcryptjs (Дата звернення: 19.04.2024).

38. Інструкція з встановлення, використання і налаштування бібліотеки Multer. URL: https://www.npmjs.com/package/multer (Дата звернення: 19.04.2024).

39. Офіційний сайт середовища розробки Visual Studio Code. URL: https://code.visualstudio.com/ (Дата звернення: 19.05.2024).

40. Документація Visual Studio Code. URL: https://code.visualstudio.com/docs (Дата звернення: 06.05.2024).

41. Встановлення програми Google Authenticator на мобільні пристрої з операційною системою IOS. URL: https://apps.apple.com/ru/app/google-authenticator/id388497605 (Дата звернення: 03.05.2024).

42. Встановлення розширення Google Authenticator для браузера Chrome. URL: https://chromewebstore.google.com/detail/authenticator (Дата звернення: 09.05.2024).

ДОДАТКИ

Додаток А. Технічне завдання

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ
Голова секції "Управління
інформаційною
безпекою" кафедри МБІС
д.т.н., професор
_____ Юрій ЯРЕМЧУК
" 22 " березня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до бакалаврської дипломної роботи на тему:
Розробка захищеного корпоративного файлообмінника з двофакторною автентифікацією

08-72.БДР.005.00.084.ТЗ

Керівник бакалаврської дипломної роботи
к.т.н., доцент
Карпінець В. В.
" 22 " Березня 2024 р.

Вінниця – 2024 р.

1. Найменування та область застосування

Розробка захищеного корпоративного файлообмінника з двофакторною автентифікацією.

Область застосування: захист інформаційних ресурсів від несанкціонованого доступу.

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ № 80 від 11.03.2024р.

3. Мета та призначення розробки

3.1 Мета розробки: розробка захищеного корпоративного файлообмінника з двофакторною автентифікацією, з метою захисту інформації від несанкціонованого доступу.

3.2 Призначення: розроблений програмний засіб виконує захист від несанкціонованого доступу.

4. Джерела розробки

4.1. Ахрамович В. М. Ідентифікація й аутентифікація, керування доступом // Сучасний захист інформації. – 2016. №4.– С. 47-51.

4.2. Бурячок В.Л. Політика інформаційної безпеки: підручник. / В.Л.Бурячок, Р.В.Грищук, В.О.Хорошко / За заг. ред. докт. техн. наук, проф. В.О. Хорошка. – К.: ПВП «Задруга», 2014. – 222 с.

4.3. Єсін В.І. Безпека інформаційних систем і технологій / В.І.Єсін, О.О. Кузнецов, Л.С. Сорока. – Харків: ХНУ імені В.Н. Каразіна, 2013. – 632 с.

4.4. ZakariaOmar, ZangooeiToomaj, MohdAfiziMohdShukran. Enhancing Mixing Recognition-Based and Recall-Based Approach in Graphical Password Scheme. IJACT, Vol. 4, No. 15, pp. 189-197, 2012.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний засіб повинен мати зручний, легкий у використанні інтерфейс користувача;

5.1.2 Реалізація методу не повинна вимагати спеціальних ліцензійних програмних додатків;

5.1.3 Програмний засіб повинен виконувати процес автентифікації користувачів у системі.

5.2 Вимоги до надійності:

5.2.1 Програмний засіб повинен працювати без помилок, у випадку виникнення критичних ситуацій необхідно передбачити виведення відповідних повідомлень;

5.2.3 Програмний засіб повинен виконувати свої функції.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Pentium 1500 МГц і подібні до них;

– оперативна пам’ять – не менше 512 Mb;

– середовище функціонування – операційна система сімейство Windows і MacOS;

– вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп’ютерною технікою.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів і тестування розробленого продукту, наведена у пункті 3.3

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист розроблюваного програмного засобу від несанкціонованого використання.

7.2 Неможливість отримання доступу незареєстрованих користувачів до інформаційних ресурсів.

8. Техніко-економічні показники

8.1 Цінність результатів використання даного проекту повинна перевищувати витрати на його реалізацію.

8.2 Має бути реалізований таким чином, щоб підходити для використання широкого загалу.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської дипломної роботи	Початок	Закінчення
1.	Визначення напрямку бакалаврської роботи, формулювання теми	23.03.2024	28.03.2024
2.	Аналіз предметної області обраної теми	02.04.2024	08.04.2024
3.	Розробка запропонованих алгоритмів роботи системи	09.04.2024	15.04.2024
4.	Проектування бази даних	17.04.2024	21.04.2024
5.	Проектування користувацького інтерфейсу	26.04.2024	29.04.2024
6.	Програмна реалізація захищеного корпоративного файлообмінника з двофакторною автентифікацією	01.05.2024	14.05.2024
7.	Тестування розробленого захищеного корпоративного файлообмінника з двофакторною автентифікацією	15.05.2024	19.05.2024
8.	Попередній захист бакалаврської дипломної роботи	23.05.2024	23.05.2024
9.	Виправлення, уточнення, корегування бакалаврської дипломної роботи	24.05.2024	29.05.2024
10.	Захист бакалаврської дипломної роботи	12.06.2024	17.06.2024

10. Порядок контролю та прийому

10.1 До приймання бакалаврської дипломної роботи надається:

– ПЗ до бакалаврської дипломної роботи;

- демонстрація результату бакалаврської дипломної роботи;
- презентація;
- відзив керівника роботи;
- відзив рецензента

Технічне завдання до виконання прийняв _____ Волос В. С.

Додаток Б. Серверний код

```
const express = require('express');
const dotenv = require('dotenv');
const cors = require('cors');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const path = require('path');  // Додано модуль path
const speakeasy = require('speakeasy');
const qrcode = require('qrcode');
const bcrypt = require('bcryptjs');
const session = require('express-session');
dotenv.config();
const app = express();
const PORT = 2220;
mongoose.connect(process.env.MONGO_URI, { useNewUrlParser: true, useUnifiedTopology: true });
const db = mongoose.connection;
db.on('error', console.error.bind(console, 'Помилка підключення до бази даних MongoDB:'));
db.once('open', () => {
  console.log('Підключено до бази даних MongoDB');
});
const userSchema = new mongoose.Schema({
  name: String,
  email: String,
  password: String,
  secretBase32: String,
});
const User = mongoose.model('User', userSchema);
app.use(cors());
app.use(bodyParser.json());
app.use(express.static(path.join(__dirname, 'public')));
app.use(session({
  secret: 'secret',
  resave: false,
  saveUninitialized: true,
}));
app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'index.html'));
});
const allowedEmails = [
  'user01@gmail.com',
  'user02@gmail.com',
  'user03@ gmail.com'
];
app.post('/register', async (req, res) => {
  const { name, email, password } = req.body;
  try {
    if (!allowedEmails.includes(email)) {
      return res.status(400).json({ success: false, message: 'Даний email не дозволено для реєстрації' });
    }
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(400).json({ success: false, message: 'Користувач з таким email вже зареєстрований' });
    }
    const hashedPassword = await bcrypt.hash(password, 10);
    const secret = speakeasy.generateSecret({ length: 20 });
    const secretBase32 = secret.base32;
    const newUser = new User({ name, email, password: hashedPassword, secretBase32 });
```

```
      await newUser.save();
      res.json({ success: true, message: Користувач ${name} зареєстрований успішно ${secretBase32} });
    } catch (error) {
      console.error('Error during registration:', error);
      res.status(500).json({ success: false, message: 'Помилка реєстрації користувача' });
    }
});
app.post('/login', async (req, res) => {
    const { email, password, secterkey } = req.body;
    try {
      const existingUser = await User.findOne({ email });
      if (!existingUser) {
        return res.status(401).json({ success: false, message: 'Неправильна електронна адреса або пароль' });
      }
      const isPasswordValid = await bcrypt.compare(password, existingUser.password);
      if (!isPasswordValid) {
        return res.status(401).json({ success: false, message: 'Неправильна електронна адреса або пароль' });
      }
      function verifyToken(secretKey, token) {
        return speakeasy.totp.verify({ secret: secretKey, encoding: 'base32', token: token });
      }
      const isTokenValid = verifyToken(existingUser.secretBase32, secterkey);
if (isTokenValid) {
        const user = { ...existingUser.toObject(), isPasswordValid };
        req.session.user = user;
        console.log("Session set:", req.session.user);
        return res.json({ success: true, message: Ви успішно увійшли в свій обліковий запис });
      } else {
        return res.status(401).json({ success: false, message: 'Неправильний токен' });
      }
    } catch (error) {
      console.error('Error during login:', error);
      res.status(500).json({ success: false, message: 'Помилка авторизації користувача' });
    }
});
app.get('/logout', (req, res) => {
    req.session.destroy(error => {
      if (error) {
        console.error('Error destroying session:', error);
        res.status(500).json({ success: false, message: 'Помилка видаленя сесії' });
      } else {
        res.json({ success: true, message: 'Сесію успішно видалено' });
      }
    });
});
app.get('/checkSession', (req, res) => {
    if (req.session.user) {
      res.json({ success: true, message: 'Сесія активна' });
    } else {
      res.json({ success: false, message: 'Сесія не активна' });
    }
});
const File = mongoose.model('File', {
    filename: String,
    description: String,
    content: Buffer,
    uploadedBy: String
});
const multer = require('multer');
const storage = multer.memoryStorage();
```

```
const upload = multer({ storage: storage });
app.post('/upload', upload.single('file'), async (req, res) => {
  try {
    const uploadedBy = req.session.user.name;
    const newFile = new File({
      filename: req.file.originalname,
      description: req.body.description,
      content: req.file.buffer,
      uploadedBy: uploadedBy
    });
    await newFile.save();
    res.json({ success: true, message: 'Ви успішно завантажили файл' });
  } catch (error) {
    console.error(error);
    res.status(500).send('Помилка завантаження файлу');}
});
app.get('/files', async (req, res) => {
  try {
    const files = await File.find({}, 'filename description uploadedBy');
    res.json(files);
  } catch (error) {
    console.error('Error fetching files:', error);
    res.status(500).json({ success: false, message: 'Помилка отримання списку файлів' });
  }
});
app.get('/download/:id', async (req, res) => {
  try {
    const fileId = req.params.id;
    const file = await File.findById(fileId);
    if (!file) {
      return res.status(404).json({ success: false, message: 'Файл не знайдено' });}
    res.set({
      'Content-Type': 'application/octet-stream',
      'Content-Disposition': attachment; filename="${file.filename}",
    });
    res.send(file.content);
  } catch (error) {
    console.error('Error fetching file:', error);
    res.status(500).json({ success: false, message: 'Помилка отримання файлу' }}});
app.delete('/files/:id', async (req, res) => {
  try {
    const fileId = req.params.id;
    const file = await File.findById(fileId);
    if (!file) {
      return res.status(404).json({ success: false, message: 'Файл не знайдено' });
    }
    if (file.uploadedBy !== req.session.user.name) {
      return res.status(403).json({ success: false, message: 'Ви не маєте доступу до видалення цього файлу' });
    }
    const deletedFile = await File.findByIdAndDelete(fileId);
if (!deletedFile) {
      return res.status(404).json({ success: false, message: 'Файл не знайдено' });
    }
    res.json({ success: true, message: 'Файл успішно видалено' });
  } catch (error) {
    console.error('Помилка видалення файлу:', error);
    res.status(500).json({ success: false, message: 'Помилка видалення файлу' });}});
app.listen(PORT, () => {
  console.log(`Сервер працює і слухає порт ${PORT}`);});
```

Додаток В. Клієнтський код завантаження файлів

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="upload.css">
  <title>File Upload</title>
</head>
<body>
  <div class="container-block">
    <h1>Завантаження файлів</h1>
    <button id="logoutBtn">Вихід</button>
    <form id="fileupload">
      <input type="file" id="file" name="file" required><br>
      <label for="email">Description:</label>
      <input type="text" id="description" name="description" required><br>
      <button type="submit">Upload</button>
    </form>
    <div class="file-list">
      <h2>Список файлів</h2>
      <ul id="files"></ul>
    </div>
  </div>
  <script>
    window.onload = function () {
      fetch('/checkSession')
        .then(response => response.json())
        .then(data => {
          if (!data.success) {
            alert("Спочатку потрібно увійти");
            window.location.href = '/login.html';
          }
        })
        .catch(error => {
          console.error('Помилка перевірки сесії', error);
        });
    };
    document.getElementById('logoutBtn').addEventListener('click', function () {
      fetch('/logout')
        .then(response => response.json())
        .then(data => {
          if (data.success) {
            window.location.href = '/login.html';
          } else {
            console.error('Error destroying session:', data.message);
          }})
        .catch(error => {
          console.error('Error destroying session:', error);
        });
    });
    document.addEventListener('DOMContentLoaded', () => {
      const fileupload = document.getElementById('fileupload');
      const fileList = document.getElementById('files');
      fileupload.addEventListener('submit', async (event) => {
        event.preventDefault();
        const file = document.getElementById('file').files[0];
```

```
            const description = document.getElementById('description').value;
            try {
                const formData = new FormData();
                formData.append('file', file);
                formData.append('description', description);
                const response = await fetch('http://localhost:2220/upload', {
                    method: 'POST',
                    body: formData,
                });
                const result = await response.json();
                console.log(result);
                alert(result.message);
                window.location.reload();
            } catch (error) {
                console.error('Error during file upload:', error);
            }
        });
        async function fetchFiles() {
            const response = await fetch('http://localhost:2220/files');
            const files = await response.json();
files.forEach(file => {
                const listItem = document.createElement('li');
                const link = document.createElement('a');
                const img = document.createElement('img');
                img.src = 'icons8-file-48.png';
                link.textContent = ${file.description}. Завантажив: (${file.uploadedBy});
                link.href = /download/${file._id};
                link.setAttribute('download', file.filename);
                listItem.appendChild(img);
                listItem.appendChild(link);
                const deleteBtn = document.createElement('button');
                deleteBtn.classList.add('deleteBtn');
                deleteBtn.textContent = 'Видалити';
                deleteBtn.dataset.fileId = file._id;
                listItem.appendChild(deleteBtn);
                fileList.appendChild(listItem);
                fileList.insertBefore(listItem, fileList.firstChild);
            });}
        fetchFiles();});
    document.addEventListener('click', async (event) => {
        if (event.target.classList.contains('deleteBtn')) {
            const fileId = event.target.dataset.fileId;
            try {
                const response = await fetch(`http://localhost:2220/files/${fileId}`, {
                    method: 'DELETE',
                });
                const result = await response.json();
                if (result.success) {
                    event.target.parentNode.remove();
                    alert(result.message);
                } else {
                    alert('Помилка видалення файлу');}
            } catch (error) {
                console.error('Помилка видалення файлу:', error);
                alert('Помилка видалення файлу');}}
    });
  </script>
</body>
</html>
```

Додаток Г. Ілюстративний матеріал

Розробка захищеного корпоративного файлообмінника з двофакторною автентифікацією

Виконав студент групи 1КІТС-20б Волос Віталій Сергійович
Науковий керівник: к.т.н., доцент каф. МБІС Карпінець Василь Васильович

Актуальність роботи

Розробка та використання захищеного корпоративного файлообмінника з двофакторною автентифікацією стає критично важливою для сучасних компаній. Використання такого рішення, дозволить забезпечити безпеку даних які зберігаються, на рівні корпоративного обміну між працівниками підприємства яке займається 3D друком, зменшуючи ризик їх витоку, чи небажаного несанкціонованого доступу.

Мета роботи

Метою роботи є розробка захищеного корпоративного файлообмінника з двофакторною автентифікацією, який буде сприяти підвищенню конфіденційності, цілісності і доступності даних, які циркулюють на підприємстві яке займається 3D друком.

Завдання

- здійснити аналіз існуючих аналогів корпоративних файлообмінників;
- здійснити аналіз поняття двофакторної автентифікації;
- здійснити аналіз існуючих методів двофакторної автентифікації;
- розробити запропоновані алгоритми роботи системи;
- обрати, та спроектувати базу даних;
- здійснити проектування користувацького інтерфейсу;
- здійснити вибір мови програмування, фреймворків, та бібліотек, здійснити вибір середовища розробки;
- здійснити програмну реалізацію спроектованого захищеного корпоративного файлообмінника з двофакторною автентифікацією;
- здійснити тестування роботи розробленого додатку.

Аналіз існуючих корпоративних файлообмінників

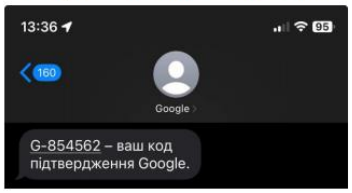
Google Drive

OneDrive for Business

Dropbox Business

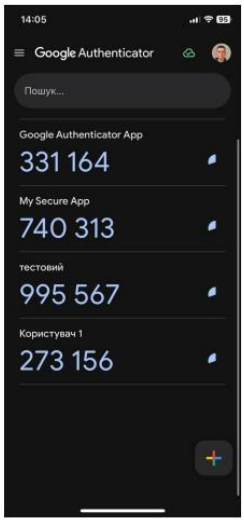
Аналіз методів двофакторної автентифікації

Одноразові коди які надходять SMS, електронною поштою, чи дзвінком.

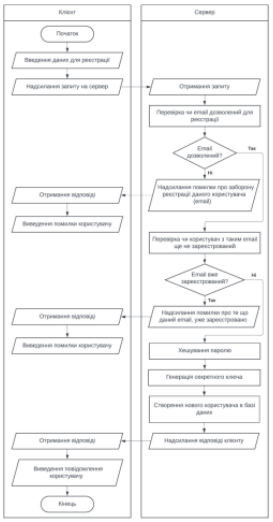


Біометрія: відбиток пальця, обличчя, або голосу

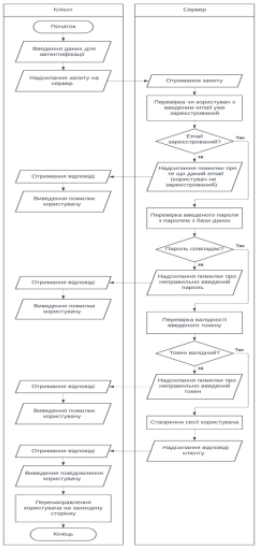
Одноразові коди з програми автентифікації.



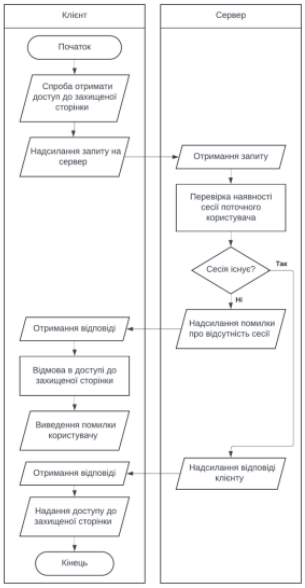
Розроблені алгоритми роботи системи



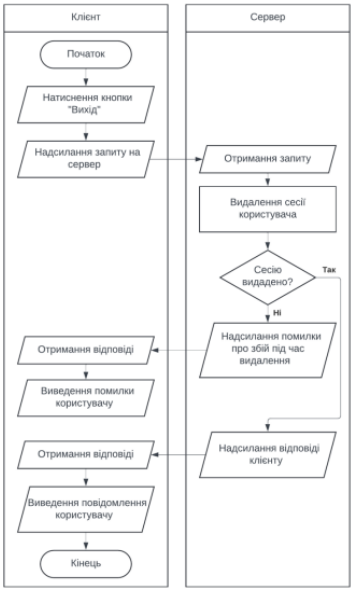
розроблена блок-схема запропонованого алгоритму реєстрації користувачів



розроблена блок-схема запропонованого алгоритму авторизації користувачів

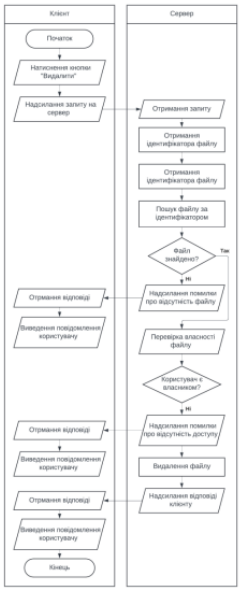


розроблена блок-схема запропонованого алгоритму перевірки сесій користувачів



розроблена блок-схема запропонованого алгоритму виходу користувачів, з облікових записів

розроблена блок-схема запропонованого алгоритму завантаження файлів



розроблена блок-схема запропонованого алгоритму видалення файлів

Використані технології

JavaScript

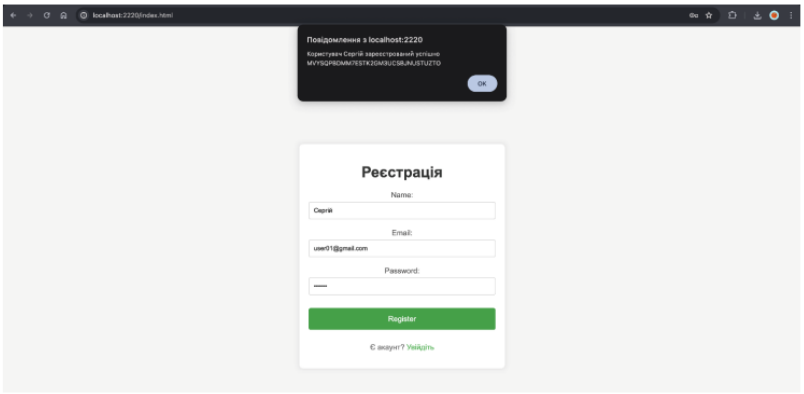
MongoDB

Node.js

Visual Studio Code

Процес реєстрації

Зовнішній вигляд сторінки реєстрації, розробленого захищеного корпоративного файлообмінника з двофакторною автентифікацією.

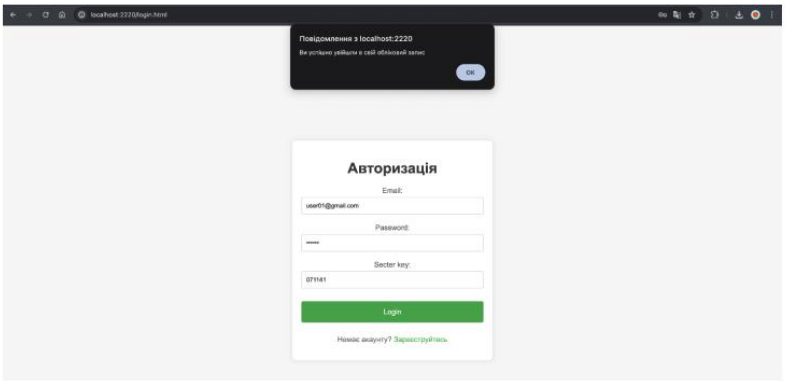


Збереження в базі даних, зареєстрованого користувача.

Процес авторизації

Тимчасовий ключ в програмі Google Authenticator, для двофакторної автентифікації

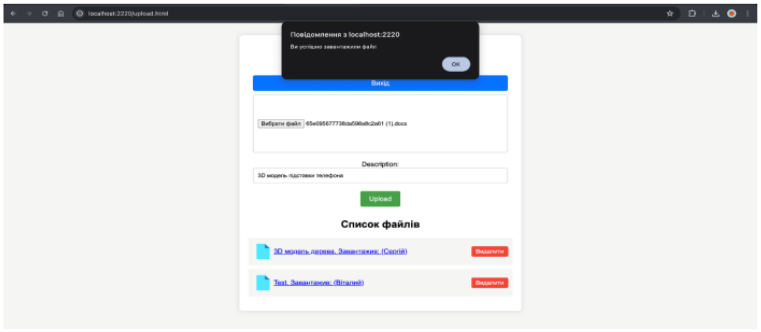
Зовнішній вигляд сторінки авторизації розробленого захищеного корпоративного файлообмінника з двофакторною автентифікацією



Створена сесія користувача, після успішної авторизації

Процес завантаження файлів

Зовнішній вигляд сторінки завантаження файлів, розробленого захищеного корпоративного файлообмінника з двофакторною автентифікацією.



Збереження завантажених файлів в базі даних.

Висновки

В результаті виконання роботи, було здійснено:

- Аналіз існуючих аналогів корпоративних файлообмінників;
- Аналіз існуючих методів двофакторної автентифікації;
- Розробка запропонованих алгоритмів роботи системи;
- Проектування бази даних;
- Проектування інтерфейсу;
- Програмно реалізовано додаток;
- Тестування розробленого додатку.

Отже, результаті виконаної роботи було досягнуто запланованого результату, а саме, отримано повноцінний, повністю працездатний і готовий до роботи, захищений корпоративний файлообмінник з двофакторною автентифікацією, для забезпечення захищеного обміну конфіденційними файлами 3D моделей між працівниками підприємства яке займається 3D друком.

Дякую за увагу!

Додаток Д. Протокол перевірки на антиплагіат

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ
ЗАПОЗИЧЕНЬ

Назва роботи: Розробка захищеного корпоративного файлообмінника з двофакторною автентифікацією

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ: Кафедра менеджменту та безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 93 % Схожість 7 %

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 _____ Коваль Н.П.
(підпис) (прізвище, ініціали)

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

Автор роботи _____ Волос В.С.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Карпінець В.В.
(підпис) (прізвище, ініціали)