НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра автоматизованих систем обробки інформації і управління

«На правах рукопису»

«До захисту допущено»

В.о. завідувача кафедри

УДК _____

_____ *О.А.Павлов*

(підпис) (ініціали, прізвище)

Магістерська дисертація

зі спеціальності *121 «Інженерія програмного забезпечення»*

на тему: «*Архітектурне рішення для безпечного обміну файлами між мобільними пристроями в корпоративній мережі*»

Виконала: студентка VI курсу, групи *ІП-82мп*

Глушкова Тетяна Олександрівна _____

(прізвище, ім'я, по батькові) (підпис)

Науковий керівник *доц., доц., к.т.н. Фіногенов О.Д.* _____

(посада, науковий ступінь, , вчене звання, прізвище та ініціали) (підпис)

Консультант *доц., к.т.н., Ліщук К.І.* _____

(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент *доц., к.т.н., доц. каф. НГІ та КГ ФМФ Яблонський П.М.* _____

(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»

Факультет *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації та управління*
(повна назва)

Рівень вищої освіти *другий (магістерський) за освітньо-професійною програмою*

Спеціальність *121 «Інженерія програмного забезпечення»*
(код і назва)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *О.А. Павлов*
(підпис) (ініціали, прізвище)
«___» _____ 201 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Глушковій Тетяні Олександрівні
(прізвище, ім'я, по батькові)

1. Тема дисертації *Архітектурне рішення для безпечного обміну файлами між мобільними пристроями в корпоративній мережі*

науковий керівник дисертації *к.т.н., доц. Фіногенов О.Д.*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "____" _____ 20__ р. № _____

2. Строк подання студентом дисертації "____" _____ 20__ р.

3. Об'єкт дослідження *безпека даних*

4. Предмет дослідження *убезпечення передачі та збереження даних при віддаленому доступі до корпоративної мережі*

5. Перелік завдань, які потрібно розробити *Дослідження методів безпечної передачі даних, методів безпечного збереження даних, методів віддаленого*

доступу; розробка архітектурного рішення.

6. Перелік графічного матеріалу

Схема варіантів використання системи

Схема роботи алгоритму першого доступу до системи

7. Орієнтовний перелік публікацій *Аналіз методів віддаленого доступу до корпоративної мережі для BYOD пристроїв. Механізм доступу до внутрішнього серверу у середовищі для безпечного обміну файлами між мобільними пристроями за межами корпоративної мережі.*

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання “ *01* ” *вересня* 20 *19* р

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	*Дослідження літератури та документації*	*18.04.2019*	
2	*Огляд існуючих рішень та архітектур*	*04.06.2019*	
3	*Розробка архітектурного рішення та алгоритмів*	*20.08.2019*	
4	*Проектування системи*	*05.09.2019*	
5	*Розробка серверного та клієнтського програмного забезпечення*	*23.10.2019*	
7	*Тестування та виправлення помилок*	*12.11.2019*	
8	*Оформлення документації*	*01.12.2019*	
9	*Подання роботи на попередній захист*	*05.12.2019*	
10	*Подання роботи на основний захист*	*16.12.2019*	

Студент _____ _____
(підпис) (ініціали, прізвище)

Науковий керівник _____ _____
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Актуальність теми. Надзвичайно швидке впровадження мобільних технологій, зокрема BYOD, організаціями викликає необхідність у більш стійких засобах безпеки та більш складних системах захисту. Тому виникла необхідність появи засобів для ефективного захисту даних на мобільних пристроях, обміну файлами маж ними (електронний документообіг), а також для безпечного доступу до внутрішньої мережі компанії.

Мета дослідження. Метою є побудова архітектурного рішення, що надає безпечний віддалений доступ до корпоративної мережі мобільним клієнтам з убезпеченням передачі та збереження даних, а також реалізація програмного додатку на основі пропонованої архітектури.

Для реалізації поставленої мети були сформульовані **наступні завдання**:

- дослідити методи безпечної передачі даних;
- дослідити методи безпечного збереження даних (файлів та даних користувачів на серверах);
- дослідити методи віддаленого доступу з публічного інтернету до корпоративної мережі;
- розробити експериментальне архітектурне рішення для убезпечення передачі даних та віддаленого доступу;
- розробити основні алгоритми взаємодії компонентів системи (алгоритм першого доступу до системи, алгоритм передачі та шифрування файлів при завантаженні, алгоритми взаємодії між компонентами);
- розробити мінімально необхідні клієнтські та серверні компоненти для підтвердження концепції.

Об’єкт дослідження – безпека даних.

Предмет дослідження – убезпечення передачі та збереження даних при віддаленому доступі до корпоративної мережі.

Наукова новизна: Запропоновано архітектурне рішення, що призначене для безпечного обміну файлами між мобільними пристроями за межами корпоративної мережі, що на відміну від більшості існуючих систем відокремлює сховище файлів від їх метаданих, не надає прямого доступу до сховища файлів з мобільного клієнта та удосконалює існуючі методи віддаленого доступу.

Запропоновано використання селективного шифрування важливих даних системи (інформація про користувачів, метадані тощо), що підвищує захищеність системи.

Практичне значення отриманих результатів визначається тим, що розроблено програмний продукт, який демонструє основні переваги запропонованого архітектурного рішення.

Зв'язок роботи з науковими програмами, планами, темами: дисертація виконувалась в рамках ініціативної теми кафедри АСОІУ «Методи та технології в задача пошуку та збереження даних». Державний реєстраційний номер 0117U000925.

Апробація: основні положення роботи доповідались і обговорювались на 3-й Всеукраїнській науково-практичній конференції молодих вчених та студентів «Інформаційні системи та технології управління» (ІСТУ-2019).

Публікації:

Аналіз методів віддаленого доступу до корпоративної мережі для BYOD пристроїв / Глушкова Т.О., Ісаченко Г.В // X Міжнародна науково-технічна конференція молодих вчених «Інформаційні технології: економіка, техніка, освіта». – 2019 р. - С. 205-206

Механізм доступу до внутрішнього серверу у середовищі для безпечного обміну файлами між мобільними пристроями за межами корпоративної мережі / Глушкова Т.О., Ісаченко Г.В. // 3 Всеукраїнська науково-практична конференція молодих вчених та студентів «Інформаційні системи та технології управління» (ІСТУ-2019). – 2019р.

Ключові слова. КОРПОРАТИВНА МЕРЕЖА, МОБІЛЬНІ ПРИСТРОЇ, ОБМІН ФАЙЛАМИ, БЕЗПЕКА ДАНИХ, BYOD, EFSS.

SUMMARY

Actuality of theme. Extremely rapid adoption of mobile technologies, such as BYOD, requires organizations to have more robust security and more sophisticated security systems. Therefore, there was a need for tools to effectively protect data on mobile devices, share files between them (electronic workflow), as well as secure access to the company's internal network.

The aim of the research. The purpose is to build an architectural solution that provides secure remote access to the corporate network to mobile clients with the security of data transmission and storage, as well as the implementation of a software application based on the proposed architecture.

To achieve this goal, the **following tasks** were formed:

- investigate secure data transmission methods;
- investigate methods for secure storage of data (files and user data on servers);
- investigate methods for remote access from the public Internet to the corporate network;
- develop an experimental architectural solution to secure data transmission and remote access;
- develop basic algorithms for interaction of system components (first access system algorithm, file transfer and encryption algorithm, algorithms for interaction between components);
- develop minimum required client and server components as proof of concept.

The object of the research is data security.

The subject of the research is transmission and storage data securing along with remote access to the corporate network.

Scientific Novelty: An architectural solution designed to securely share files between mobile devices outside of the corporate network, which, unlike most existing

systems, separates file storage from their metadata, does not provide direct access to file storage from a mobile client, and improves existing remote access methods.

Suggested usage of selective encryption of important system data (user information, metadata, etc.), which increases the security of the system.

The practical value of the results is a software product which demonstrates the main advantages of the proposed architectural solution.

Relationship with scientific programs, plans, topics: the dissertation was carried out in the scope of initiative theme of the ASOIU department “Methods and technologies in the task of searching and storing data”. State registration number 0117U000925.

Testing: the main points of the work were reported and discussed at the 3rd All-Ukrainian Scientific and Practical Conference of Young Scientists and Students "Information Systems and Management Technologies" (ISTU-2019).

Articles:

Analysis of remote access methods to the corporate network for BYOD devices / Hlushkova T., Isachenko G // X International Scientific and Technical Conference of Young Scientists "Information Technologies: Economics, Technology, Education". - 2019 - pp. 205-206

The mechanism of access to the internal server in the environment for secure file sharing between mobile devices outside the corporate network / Hlushkova T., Isachenko G. // 3 All-Ukrainian Scientific and Practical Conference of Young Scientists and Students "Information Systems and Management Technologies" (ISTU-2019). - 2019

Keywords. CORPORATE NETWORK, MOBILE DEVICES, FILE SHARING, DATA SECURITY, BYOD, EFSS.

ЗМІСТ

СПИСОК УМОВНИХ СКОРОЧЕНЬ

AES - Advanced Encryption Standard - широко використовуваний симетричний алгоритм блокового шифрування

API – Application programming interface – прикладний програмний інтерфейс

BYOD - Bring your own device - це IT-політика, згідно з якою співробітникам дозволено використовувати особисті мобільні пристрої

CA – Certificate Authority - довірений центр сертифікації

CSR – Certificate signing request - запит на підпис сертифікату

ECDSA – Elliptic Curve Digital Signature Algorithm – алгоритм цифрового підпису на еліптичних кривих

EFSS – Enterprise file sync and sharing - програмне забезпечення, яке дозволяє організаціям надійно синхронізувати та обмінюватися документами

HMAC – Hash-based Message Authentication Code – хеш-код аутентифікації повідомлень

HOTP – HMAC-Based One-Time Password Algorithm – алгоритм одноразових паролів на основі HMAC

HTTPS – Hyper Text Transfer Protocol Secure - протокол передачі гіпертекстових документів з підтримкою шифрування

JWT – JSON Web Token – Веб токен на основі JSON

PoC – Proof of Concept - мінімальний додаток для підтвердження працездатності концепції

REST – Representational State Transfer – архітектурний підхід до побудови веб сервісів

SaaS – Software as a service – програмне забезпечення як сервіс

SSL – Secure Sockets Layer – протокол для шифрування трафіку

SSO – Single Sign-On – технологія єдиного входу

TLS – Transport Layer Security – захист на транспортному рівні

ВСТУП

Мобільні технології стрімко заполонили світ за останні декілька років. За статистичними даними близько 5.13 мільярдів людей мають мобільні девайси у 2019 році, що відповідно складає близько 66.53% світу [1].

Виходячи із стрімкого розвитку мобільних технологій та такої великої кількості приватних девайсів, серед організацій набула популярності концепція Bring Your Own Device (BYOD). BYOD дозволяє співробітникам приносити та використовувати для роботи власні пристрої: смартфони, планшети та ноутбуки. Ці пристрої використовується і у рамках комп'ютерної мережі компанії і поза нею. Таким чином, меншим стає відсоток пристроїв, що належать компанії. Численні корпорації та організації взяли на себе лідерство у прийнятті BYOD, а саме Intel, Citrix Systems, Unisys, Apple.

Переваги BYOD зрозумілі: працівникам легше та зручніше користуватися власними пристроями, в той же час роботодавці економлять гроші, не платячи за високоефективні пристрої. Проте використання BYOD вимагає використання певних політик безпеки.

Ефективна політика BYOD повинна чітко визначати всі цілі та обмеження, пов'язані з використанням активів компанії та конфіденційних даних. Політика описує всі дії, дозволені на пристроях, коли вони працюють в корпоративній мережі, а також за її межами.

Надзвичайно швидке впровадження мобільних технології, зокрема BYOD, викликає необхідність у більш стійких засобах безпеки та більш складних системах. BYOD може спричинити багато ризиків і проблем, і найбільший з них - втрата даних. А з ростом технологій та кількістю мобільних девайсів росте кількість втрачених та викрадених даних. З'являються нові площини та вектори атаки. Існуючи мобільні системи мають купи вже відомих вразливостей, а також вразливості нульового дня.

Вживання власних девайсів для роботи призводить до того, що конфіденційні дані компанії можуть бути втрачені, якщо працівник випадково

або невипадково залишить їх на мобільному пристрої незахищеними. Не зважаючи на це, документообіг або обмін файлами є важливим аспектом функціонування будь-якого підприємства. Працівнику у певний момент вдома або під час відпустки може знадобитися доступ до спільних (але конфіденційних) файлів або документів. Управління спільними файлами та правила доступу до них може бути складною та схильною до помилок процедурою. Було багато випадків, коли люди мимоволі розкривали приватні дані або нехтували спільним доступом до приватних файлів у процесі комунікації чи командній роботі.

Але в той же час, при спільний або віддаленій роботі над задачею людям необхідний доступ до спільних ресурсів компанії.

Зазвичай, спільний доступ до файлів та обмін ними визначають як діяльність із надання певних файлів доступними для окремої людини або групи людей з можливістю надання відправником конкретних дозволів для людей або груп людей (наприклад, можливості перегляду файлу директором компанії або бухгалтером, видалення та редагування). Ці задачі вимагають деякої взаємодії з набором файлів або файловою системою. Приклади спільного використання файлів включають надсилання вкладень електронною поштою або завантаження файлу у загальну папку та надання конкретним користувачам права читати та редагувати цей файл.

Основні компоненти (актори) системи файлообміну: відправник (власник), одержувач (и), файл (и), механізм обміну (публікації), дії, які можна здійснити з файлом.

Найбільш популярними зараз є наступні методи файлообміну:

- вкладення у електронній пошті;
- фізичні девайси (флеш накопичувач);
- публічні сервіси обміну файлами;
- месенджери.

Відсоток людей, що використовує ці методи складає 99, 97, 81 та 77 відповідно [2]. Проте усі перелічені методи є небезпечними. Електрона пошта часто використовується для фішингу та розповсюдження шкідливого програмного забезпечення, флеш накопичувачі можуть буди вкрадені або зіпсовані, публічні сервіси підходять лише до приватного користування (не корпоративного), більшість месенджерів не є безпечними та часто співпрацюють зі спецслужбами країн.

Отже звичні для людини методи використовувані у повсякденному житті не є прийнятними для підприємств і організацій. Тому виникла категорія програмного забезпечення, що має назву синхронізація та обмін файлами на підприємстві (Enterprise file synchronization and sharing, також відомий як EFSS), яка відноситься до програмних служб, що дозволяють організаціям безпечно синхронізувати та обмінюватися документами, фотографіями, відео та файлами з декількох пристроїв з працівниками та зовнішніми клієнтами та партнерами. Організації часто застосовують ці технології, щоб не дозволяти працівникам використовувати у робочих цілях додатки для персонального використання. Через використання сторонніх засобів, зберігання, доступ та керування корпоративними даними перебувають за межами контролю та видимості IT-відділу.

Пропоноване архітектурне рішення ставить на меті надати організації централізовану систему файлообміну підконтрольну IT-відділу.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз особливостей корпоративної мережі

Своєчасний обмін інформацією всередині учасників колективу - важлива складова успішної роботи будь-якої компанії, незалежно від її специфіки і масштабу. Поширення цифрових технологій у всіх галузях виробництва сприяє широкому впровадженню корпоративних мереж на різних рівнях бізнесу, від невеликих фірм до холдингів.

Популярність корпоративних мереж обумовлена низкою їх переваг. Зменшення часу простою системи в разі апаратних, програмних і технічних помилок надає стабільний, безперервний обмін даними між усіма учасниками. Спеціальні програми і гнучкі налаштування прав доступу до окремих документів, функцій і розділів знижують ризик витоку інформації, втрати конфіденційних даних. Крім цього, порушників легко відстежити за допомогою програмних рішень.

Процес проектування корпоративної мережі включає об'єднання локальних мереж департаментів всередині компанії і створення матеріально-технічної бази для подальшого планування, організації та управління профільною діяльністю підприємства.

Побудова корпоративної мережі заснована на узгодженій архітектурі даних, платформ і додатків, за допомогою яких забезпечується обмін інформацією між користувачами. Отримання функціонуючої корпоративної мережі додатково передбачає розробку засобів обслуговування і захисту баз даних.

VPN, або Virtual Private Network - варіант створення віртуальної мережі всередині підприємства, яка використовує можливості глобальної мережі. Особливість побудови такої мережі - можливість доступу до Internet з будь-якої точки світу за допомогою зареєстрованого логіна і пароля. Рішення популярне серед IT-компаній, дизайнерських бюро та інших підприємств, які наймають співробітників для віддаленої роботи. Недолік цього способу організації

локальної мережі - загроза несанкціонованого доступу і втрати даних користувачів при неналежному налаштуванні [3].

Wi-Fi - більш технологічний і сучасний варіант створення корпоративної мережі, не прив'язаної до апаратних потужностей і фізичного розташуваннюя користувачів. За допомогою роутерів налаштовується доступ в мережу для всіх співробітників, при цьому «потрапити» в мережу можна з будь-якого пристрою. Головна перевага Wi-Fi - легка інтеграція і масштабування створеної мережі на будь-яку кількість користувачів. За допомогою Wi-Fi здійснюється динамічний перерозподіл пропускної здатності мережі між окремими вузлами, в залежності від рівня застосовуваної навантаження.

У контексті корпоративної мережі, важливу роль має віддалений доступ до внутрішньої мережі компанії, що зазвичає реалізується за допомогою VPN. Подібне рішення безпечно завдяки можливості гнучкого налаштування прав доступу до даних, що зберігаються на серверах компанії.

Несанкціонований доступ до даних, що зберігаються на корпоративних серверах і загроза їх втрати - дві основні небезпеки, від яких необхідно захистити мережу підприємства [4]. Для цих цілей використовуються:

- антивірусні системи;
- системи моніторингу;
- оперативне блокування несанкціонованих доступів вручну;
- гручке налаштування VPN мереж.

Особливості корпоративної мережі

До найбільш істотних особливостей корпоративних мереж можна віднести наступні:

– масштабність системи – корпоративна мережа включає велику кількість вузлів (комп’ютерів) складно пов'язаних один з одним і розподілених, зазвичай, на великій території світу;

– гетерогенність – неоднорідність обладнання, протоколів, операційних систем, додатків;

– інтегрованість – неоднорідні частини і підмережі корпоративної мережі повинні працювати як єдине ціле, надаючи користувачам по можливості прозорий доступ до всіх необхідних ресурсів;

– підвищені вимоги до надійності – в корпоративній мережі виконуються стратегічно важливі для роботи підприємства додатки і зберігаються такої же важливості дані, тому мережа повинна мати мінімально можливий час простоїв основних компонентів через збої і відмови, а критична інформація не повинна губитися;

– підвищені вимоги до керованості мережі – масштабність мережі вимагає розвинених багатофункціональних засобів управління мережею, інакше витрати експлуатації мережі величезним штатом фахівців набагато перевищать принесені вигоди. У корпоративних мережах користувачі мають дуже жорсткі вимоги до часу усунення відмови обладнання, тому апаратна надмірність і планування відновлення після відмов є дуже важливими;

– широта охоплення технічних проблем – при проектуванні корпоративної мережі розробники мають справу з найширшим колом технічних питань;

– потреба в наявності на підприємстві фахівців різних профілів високої кваліфікації – створення корпоративної мережі та управління нею вимагає наявності проектувальників мережі, інсталяторів мережі і адміністраторів мережі.

Виходячи з описаних особливотей, можна прийти до висновку що якщо існує корпоративна мережа компанії, то скоріше за все вона буде добре підтриманованою та ефективною.

У додаток до описаних особливостей існує класифікація рівнів компаній CIS Controls: організацією CIS було розроблено пріоритизований набір дій щодо захисту організації та даних від відомих векторів кібератаки. Організації в усьому світі покладаються на CIS Controls (рис. 1.1) для вдосконалення свого кіберзахисту [5]. На основі контролей розроблено так звані групи впровадження,

що дозволяють класифікувати компанію і зосередити ресурси компанії на проблемних або не реалізованих аспектах.

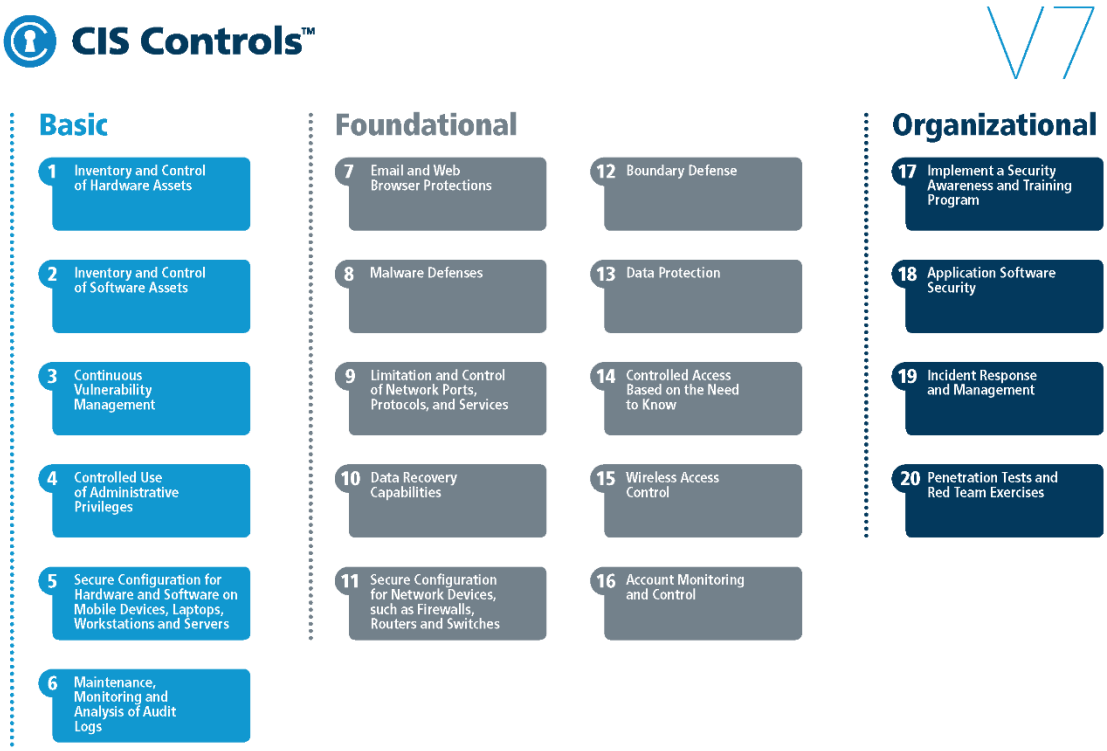


Рисунок 1.1 – Рівні зрілості компаній (CIS Controls)

1.2 Аналіз підходів до реалізації віддаленого доступу до закритої мережі

Віддалений доступ – це можливість доступу до комп'ютера або мережі віддалено через мережеве з'єднання. Віддалений доступ дозволяє користувачам отримувати доступ до необхідних їм систем, коли вони фізично не можуть безпосередньо підключатися. Користувачам надають доступ до систем віддалено за допомогою телекомунікацій або підключення до Інтернету [6].

Іншими словами, віддалений доступ дозволяє користувачам отримувати доступ до файлів та інших системних ресурсів на будь-яких пристроях або серверах, підключених до мережі в будь-який час, підвищуючи продуктивність працівників і дозволяючи їм краще співпрацювати з колегами по всьому світу.

Стратегія віддаленого доступу також надає організаціям можливість наймати найкращих знавців своєї справи незалежно від їх місця розташування, сприяти співпраці між командами та офісами.

Протягом багатьох років для організацій була загальноприйнятою практика надання віддаленого доступу та можливсоті дистанційної роботи на особистих обчислювальних пристроїв працівників, підрядників, ділових партнерів та постачальників. Остання тенденція, Bring Your Own Device, розширює цю концепцію дистанційної роботи, щоб дозволити пристроям безпосередньо підключатися до корпоративних мереж організації. Це призводить до значного ризику для організації, якщо пристрої розміщені в внутрішніх мережах організації, оскільки пристрої BYOD, якими керують самі користувачі, зазвичай мають набагато нижчий рівень захищенності ніж власні пристрої організації [7].

Проте цей ризик можна значною мірою пом'якшити, створивши окрему дротову або бездротову мережу на підприємстві, призначеному для пристроїв BYOD. Ця мережа BYOD повинна бути зовнішньою (наприклад, від DMZ організації) і не надавати більше доступу до ресурсів підприємства, ніж користувачі вже мають у рамках віддаленого доступу до мережі. Тому, організації повинні забезпечити створення окремої, зовнішньої, спеціалізованої мережі для використання BYOD у межах підприємства. Ця мережа повинна бути захищена і контролюватися у спосіб, який узгоджується з тим, як сегменти віддаленого доступу захищені і контролюються.

Ризики, пов'язані з BYOD та клієнтськими пристроями сторонніх виробників, дуже схожі з тими, що стосуються загальної дистанційної роботи та віддаленого доступу. Ці ризики наступні:

- недостатність фізичного контролю над пристроєм (при втраті або викраденні);
- не убезпечені публічні мережі, до яких приєднується девайс;
- інфіковані девайси у внутрішніх мережах;

- зовнішній доступ до внутрішніх ресурсів з пристроїв.

Організації мають багато варіантів для забезпечення віддаленого доступу до своїх обчислювальних ресурсів. Методи віддаленого доступу також можуть використовуватися для забезпечення доступу до внутрішніх ресурсів для BYOD і клієнтських пристроїв, керованих третьою стороною, приєднаних до бездротових BYOD мереж.

Фахівці з технічної підтримки також використовують віддалений доступ для підключення до комп'ютерів користувачів із віддалених місць, щоб допомогти їм вирішити проблеми зі своїми системами або програмним забезпеченням.

Сервер віддаленого доступу - це сервер що надає віддалений доступ до внутрішньої мережі, використовуючи методи описані нижче у підрозділі.

Безпека серверів віддаленого доступу є особливо важливою, оскільки вони надають можливість зовнішнім хостам отримувати доступ до внутрішніх ресурсів, а також захищеному, ізольованому середовищу дистанційної роботи для клієнтських пристроїв (видані організацією, пристрої сторонніх виробників, BYOD). На додаток до несанкціонованого доступу до корпоративних ресурсів і віддалених пристроїв клієнта, скомпрометований сервер може бути використаний для підслуховування розмов і маніпулювання працівниками.

Особливо важливо для організацій забезпечити, щоб сервери віддаленого доступу були максимально оновлені (patch) і керовані лише з довірених хостів авторизованими адміністраторами. У більшості випадків сервер повинен розміщуватися в периметрі мережі організації, щоб він діяв як єдина точка входу в мережу.

Існує багато загроз для клієнтських пристроїв при дистанційній роботі, включаючи віруси, втрати або крадіжки пристроїв. Оскільки пристрої дистанційної роботи, як правило, піддаються більшому ризику в зовнішніх середовищах, ніж у корпоративних середовищах, рекомендовано використовувати додаткові засоби безпеки, такі як шифрування конфіденційних даних, що зберігаються на пристроях. Клієнтські пристрої для дистанційної

роботи повинні мати всі засоби безпеки, що передбачаються і надаються організацією у рамках політики безпеки.

Дозвіл безпосередньо підключатися до корпоративних мереж власним пристроєм працівника додає значний ризик, якщо пристрої мають доступ до внутрішньої мережі організації, оскільки ці пристрої часто не захищені в тій же мірі, як власні пристрої організації. Однак цей ризик можна значною мірою пом'якшити, створивши окрему дротову або бездротову мережу на підприємстві, присвячену цим пристроям. Мережа повинна бути зовнішньою (наприклад, поза демілітаризованою зоною організації) і не надавати більше доступу до ресурсів підприємства, ніж користувачі вже мають через віддалений доступ. Ця мережа повинна бути захищена і контролюватися у спосіб, узгоджений з тим, як сегменти віддаленого доступу захищені і контролюються.

Отже, технології дистанційного доступу та віддаленого доступу часто потребують додаткового захисту, оскільки їхня природа зазвичай пов'язує їх із зовнішніми загрозами більше, ніж технології, доступні лише з самої організації.

Проаналізовані методи віддаленого доступу, які найчастіше використовуються для віддаленої роботи, були розділені на чотири категорії на основі їх архітектур: тунелювання, портали, доступ до віддаленого робочого столу та прямий доступ до програм. Методи віддаленого доступу усіх чотирьох категоріях мають деякі спільні риси:

– залежать від фізичної безпеки клієнтських пристроїв;

– можуть використовувати кілька типів серверів і механізмів аутентифікації користувачів. Така гнучкість дозволяє деяким методам віддаленого доступу працювати з існуючими механізмами аутентифікації організації, такими як паролі або сертифікати;

– можуть використовувати криптографію для захисту даних, що передаються між клієнтським пристроєм та організацією. Криптографічний захист притаманний VPN і тунелюванню в цілому, але ця опція доступна в більшості систем віддаленого доступу до комп'ютерів і додатків;

– дозволяють працівникам зберігати дані на своїх клієнтських пристроях.

Тунелювання (Tunneling)

Багато методів віддаленого доступу забезпечують безпечний тунель зв'язку, через який інформація може передаватися через публічну мережу (Інтернет). Тунелі зазвичай встановлюються за допомогою технологій віртуальної приватної мережі (VPN). Після встановлення VPN тунелю між віддаленим клієнтським пристроєм та VPN сервером організації, працівник може отримати доступ до багатьох обчислювальних ресурсів організації. Щоб використовувати VPN, користувачі повинні або мати відповідне програмне забезпечення що реалізує потрібний VPN протокол на своїх клієнтських пристроях. На рисунку 1.2 клієнт VPN встановлений на кожному з клієнтських пристроїв, і є один серверний компонент, на якому запущено серверне програмне забезпечення VPN.

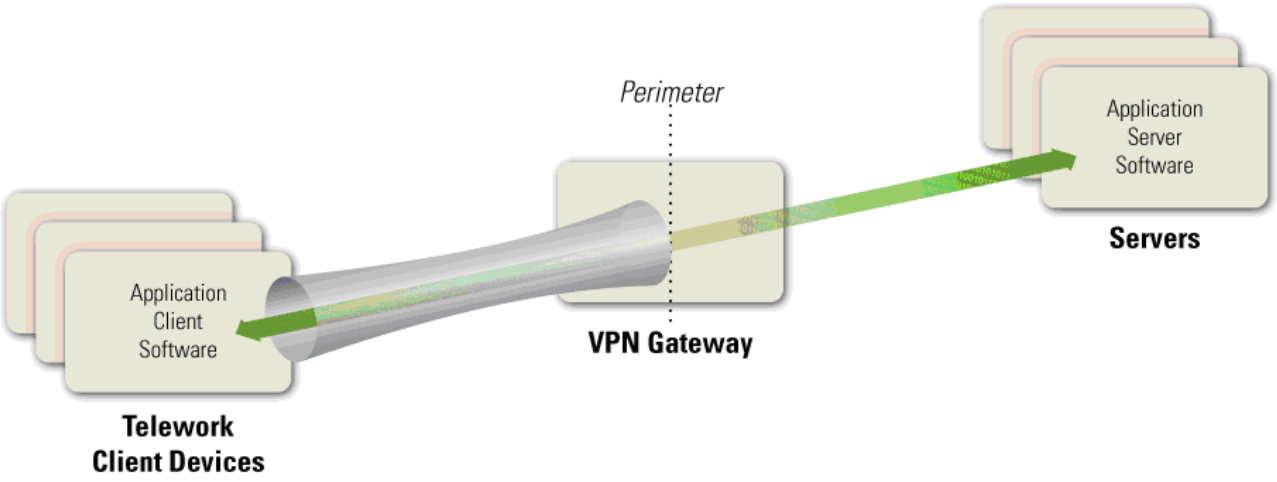


Рисунок 1.2 – Архітектура тунелювання

Через цей тунель, клієнтське програмне забезпечення (наприклад, клієнт електронної пошти, текстовий процесор, веб-браузер, клієнт баз даних), встановлене на клієнтському пристрої, спілкується з програмним забезпеченням серверів, що знаходиться на серверах організації.

Тунелі використовують шифрування для захисту конфіденційності та цілісності переданої інформації між клієнтським пристроєм і сервером. Тунелі

також можуть аутентифікувати користувачів, забезпечувати контроль доступу (наприклад, обмежувати, які протоколи можуть використовуватися або які внутрішні хости можуть бути досягнуті за допомогою віддаленого доступу) і виконувати інші функції безпеки. Клієнтське програмне забезпечення і завантажені дані перебувають на клієнтському пристрої, тому вони не захищені тунельним рішенням і повинні бути захищені іншими засобами.

Типи VPN, які найчастіше використовуються для віддалених працівників - це Internet Protocol Security (IPsec), OpenVPN і Secure Sockets Layer (SSL) тунелі. Тунелювання також може бути досягнуто за допомогою Secure Shell (SSH), хоча це менш часто використовується і часто є більш складним у налаштуванні і підтримці, ніж IPsec або SSL.

Шлюз VPN може керувати доступом до частин мережі та типами доступу, які отримує віддалений клієнт після аутентифікації. Наприклад, VPN може дозволити користувачеві мати доступ лише до однієї підмережі або лише запускати окремі програми на певних серверах захищеної мережі. Таким чином, незважаючи на те, що криптографічний тунель закінчується шлюзом VPN, шлюз може додавати додаткову маршрутизацію до трафіку віддаленого клієнта, щоб дозволити доступ лише до деяких частин внутрішньої мережі.

Портали додатків (Application Portals)

Наступна категорія рішень віддаленого доступу - портали. Портал - це сервер, який надає доступ до одного або більше додатків через єдиний централізований інтерфейс. Працівник використовує клієнтський додаток порталу на своєму пристрої для дистанційного доступу до порталу. Більшість порталів є веб додатками - для них клієнт порталу є звичайним веб-браузером. На рисунку 1.3 выдображено базову архітектуру рішення порталу.

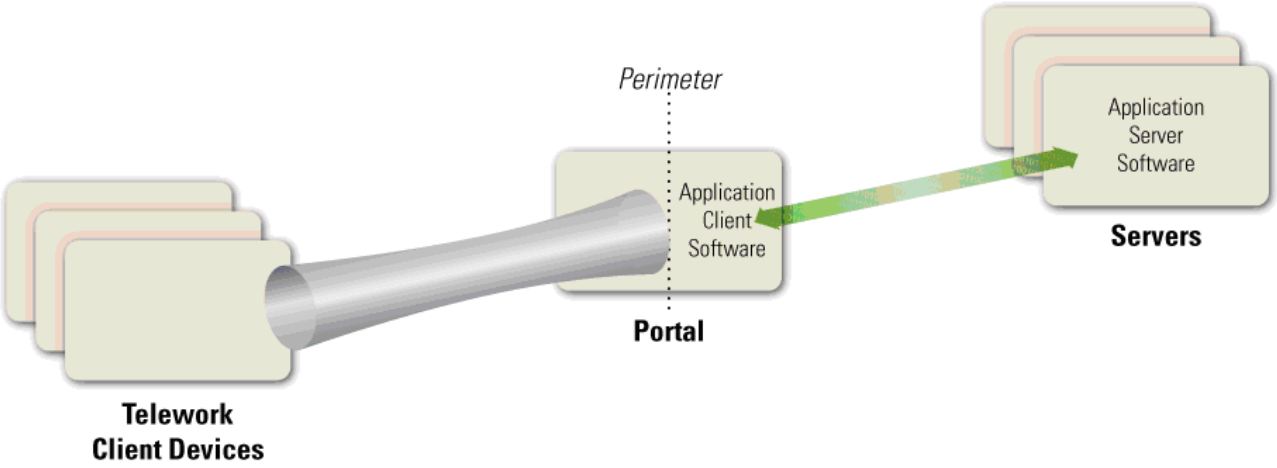


Рисунок 1.3 – Архітектура порталу додатків

Програмне забезпечення клієнтського додатка встановлюється на сервері порталу, і він здійснює зв'язок з програмним забезпеченням сервера додатків на серверах організації.

З точки зору безпеки, портали мають більшість тих самих характеристик, що й тунелі: портали захищають інформацію між клієнтськими пристроями та порталом, а також можуть забезпечувати аутентифікацію, контроль доступу та інші функції безпеки. Однак існує важлива відмінність між тунелями та порталами - розташуванням клієнтського програмного забезпечення та пов’язаних даних. У випадку тунеля програмне забезпечення та дані знаходяться на клієнтському пристрої; у порталі вони знаходяться на сервері порталу. Сервер порталу передає дані на клієнтський пристрій у вигляді зображень робочого столі або веб-сторінок, але дані що зберагіється на клієнтському пристрої є тимчасовими. Однак портали можуть бути налаштовані так, щоб дозволити клієнтам завантажувати вміст з порталу та зберігати його на клієнтському пристрої або в інших місцях поза безпечним середовищем.

Наявність централізованого клієнтського програмного забезпечення надає організації більше контролю над тим, як програмне забезпечення та дані захищені, на відміну від більш розподілених рішень віддаленого доступу.

Портали обмежують доступ, який працівник має до конкретних додатків,

що входять до складу порталу. Ці додатки в свою чергу обмежують доступ робітника до серверів всередині мережі.

Існує кілька типів портальних рішень, які зазвичай використовуються для віддаленого доступу. Веб-портал надає користувачеві доступ до декількох веб-додатків з одного веб-сайту порталу. Портал SSL VPN є звичайною формою веб-порталу. Іншим типом портального рішення є доступ до термінального сервера, який надає кожному працівнику доступ до окремого віртуального робочого столу. Сервер терміналів імітує зовнішній вигляд операційної системи для комп'ютерів і забезпечує доступ до додатків. Доступ до сервера терміналів вимагає від віддаленого працівника або встановлення на клієнтському пристрої спеціального клієнтського додатку термінального сервера, або використання веб-інтерфейсу, часто за допомогою плагіна браузера або іншого додаткового програмного забезпечення, наданого організацією.

Інший подібний метод віддаленого доступу, що називається інфраструктурою віртуального робочого столу (virtual desktop infrastructure, VDI), передбачає підключення користувача до системи, яка містить віртуальні образи стандартизованих, не імітованих операційних систем і настільних комп'ютерів. Коли сеансом віддаленого доступу закінчего, віртуальне зображення знищуеється, так що наступний користувач матиме чистий віртуальний робочий стіл. VDI є особливо корисним для захисту на пристроях BYOD та інших пристроях, які не відповідають вимогам організації.

Незважаючи на те, що доступ до термінальних серверів і технологій VDI в першу чергу призначений для комп'ютерів дистанційної роботи, існує нова технологія, що забезпечує подібні можливості для мобільних пристроїв: віртуальна мобільна інфраструктура (VMI). Так само, як рішення VDI поставляє безпечний віртуальний робочий стіл на ПК для дистанційної роботи, так і VMI забезпечує безпечне середовище віртуальних мобільних пристроїв для мобільного пристрою дистанційної роботи. Технологія корисна для організацій, які надають можливість використання мобільних пристроїв для дистанційної

роботи, зокрема BYOD або мобільних пристроїв, що контролюються третіми сторонами.

Віддалений доступ до робочого столу (Remote Desktop Access)

Рішення для віддаленого доступу до робочого столу дає можливість дистанційно керувати певним комп'ютером в організації, найчастіше власним комп'ютером в офісі організації з віддаленого клієнта. Працівник керує клавіатурою і мишею віддаленого комп'ютера і бачить екран комп'ютера на екрані локального підключеного клієнта. Віддалений доступ до робочого столу дозволяє користувачеві отримувати доступ до всіх програм, даних та інших ресурсів, які зазвичай доступні з комп'ютера в офісі. Рисунок 1.4 відображає базову архітектуру доступу до віддаленого робочого столу. Клієнтська програма для роботи з віддаленим робочого столом або плагін веб-додатку встановлюється на кожному клієнтському пристрої віддаленої роботи, і він підключається безпосередньо до відповідної внутрішньої робочої станції працівника у внутрішній мережі організації.

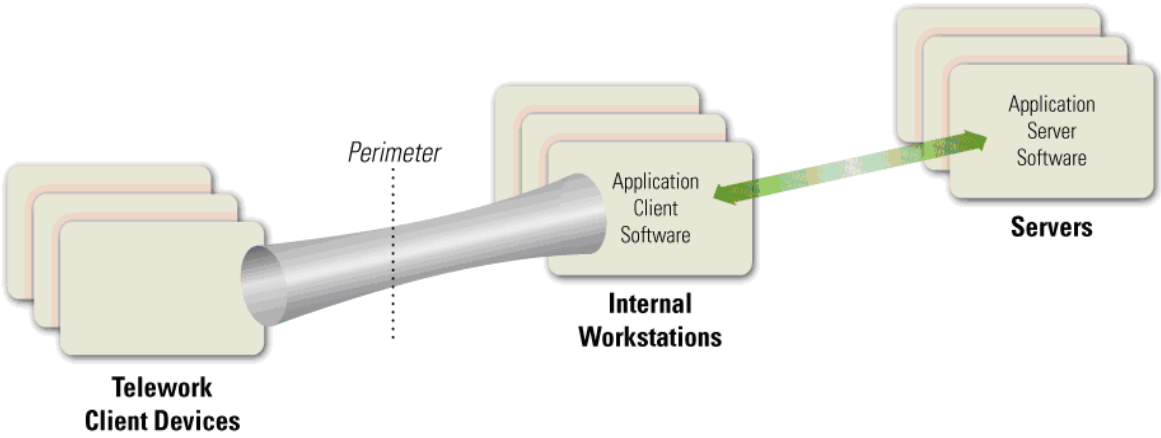


Рисунок 1.4 – Архітектура віддаленого доступу до робочого стола

Існує два основних типу доступу до віддаленого робочого стола: прямий (між працівником та внутрішньою робочою станцією), а також непрямий (через надійну проміжну систему). Однак прямий доступ часто неможливий, оскільки він забороняється багатьма брандмауерами. Наприклад, якщо внутрішня робоча станція стоїть за брандмауером, що виконує трансляцію мережевих адрес (NAT), клієнтський пристрій не може ініціювати контакт з внутрішньою робочою

станцією, якщо тільки NAT не дозволяє такого контакту, або ж внутрішня робоча станція ініціює зв'язок із зовнішнім клієнтським пристроєм дистанційної роботи наприклад, періодично перевіряти з клієнтським пристроєм, чи хоче він підключитися).

Непрямий доступ до віддаленого робочого столу здійснюється через проміжний сервер. Цей сервер іноді є частиною брандмауера організації, але частіше за все керується довіреною комерційним або безкоштовним сервісои сторонніх розробників за межами периметра мережі організації. Зазвичай існують окремі з'єднання між клієнтським пристроєм дистанційної роботи і проміжним сервером, а також між проміжним сервером і внутрішньою робочою станцією. Безпека цього проміжного сервера є дуже важливою, оскільки він відповідає за належну аутентифікацію працівників і запобігання доступу до незашифрованого трафіку від неавторизованих сторін.

Програмне забезпечення для прямого віддаленого доступу до робочого столу захищає конфіденційність і цілісність зв'язку, а також аутентифікує користувача, щоб переконатися, що ніхто інший не підключається до внутрішньої робочої станції. Проте, оскільки це передбачає end-to-end шифрування у мережі організації, вміст зв'язку приховано від систем контролю та моніторингу, таких як брандмауери та системи виявлення вторгнень. Для багатьох організацій підвищений ризик, викликаний цим, не є виправданим та не коштує переваг, тому прямі з'єднання від зовнішніх клієнтських пристроїв до внутрішніх робочих станцій заборонені.

Інша серйозна проблема безпеки з програмним забезпеченням для віддаленого робочого столу полягає в тому, що вона децентралізована; замість того, щоб організація мала захищати один сервер шлюзу VPN або сервер порталу, організація має замість цього забезпечити кожну внутрішню робочу станцію, до якої можна отримати доступ за допомогою доступу до віддаленого робочого столу. Оскільки ці внутрішні робочі станції можуть бути доступні з Інтернету, безпосередньо або опосередковано, вони зазвичай повинні бути захищені майже так само, як повноцінні сервери віддаленого доступу, але

звичайні робочі станції, як правило, не розраховані на такий ступінь безпеки. Застосування контролів для кожної робочої станції для підвищення її безпеки до прийнятного рівня часто передбачає значну кількість часу та ресурсів, а також придбання додаткових засобів безпеки. Крім того, рішення для аутентифікації, такі як можливості двофакторної аутентифікації, можуть потребувати розгортання на кожній внутрішній робочій станції.

Як правило, рішення для доступу до віддалених робочих столів, такі як протокол Microsoft Remote Desktop, або Virtual Network Computing (VNC), повинні використовуватися лише у виняткових випадках після ретельного аналізу ризиків безпеки.

Інші типи рішень віддаленого доступу, описані в цьому розділі, забезпечують кращий рівень безпеки.

Прямий доступ до додатку (Direct Application Access)

Віддалений доступ до додатку можна здійснити без використання спеціалізованого програмного забезпечення. Працівник може отримати доступ до окремої програми напряму, додаток сам забезпечує власну безпеку (шифрування зв'язку, аутентифікація користувача тощо). Рисунок 1.5 відображає архітектуру високого рівня для прямого доступу до програм. Програмне забезпечення клієнтського додатка, встановлене на клієнтському пристрої дистанційної роботи, ініціює з'єднання з сервером, який зазвичай розташований на периметрі організації (наприклад, в демілітаризованій зоні, DMZ) або в хмарі.

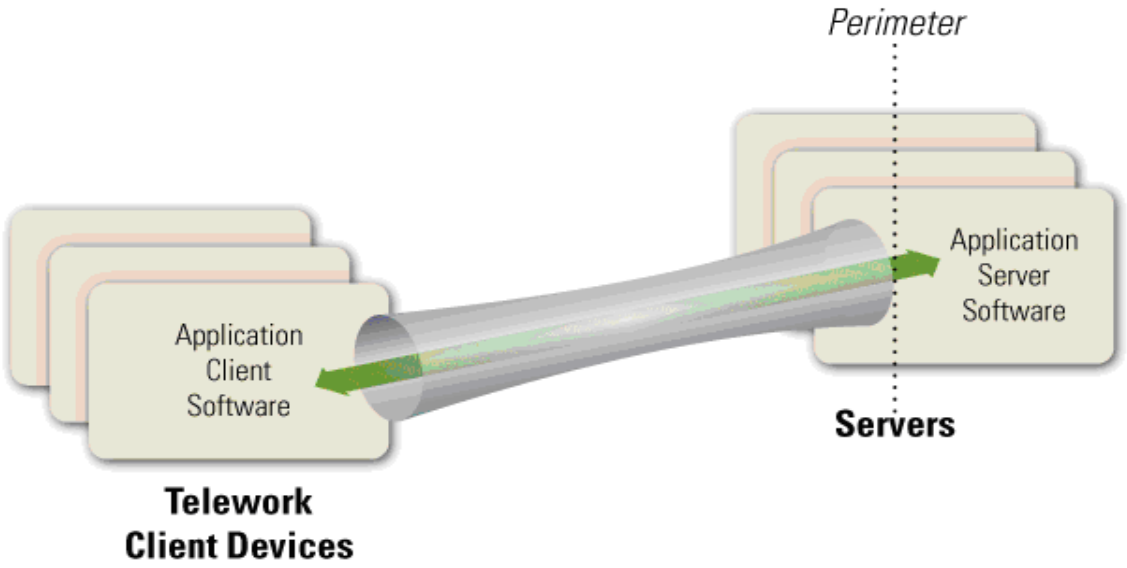


Рисунок 1.5 – Архітектура прямого доступу до додатку

Одним з найбільш поширених прикладів прямого доступу до програм є веб-пошта. Працівник запускає веб-браузер і підключається до веб-сервера, який надає доступ до електронної пошти. Веб-сервер запускає HTTP через TLS (HTTPS), щоб захистити комунікацію, а програма веб-пошти на сервері аутентифікує працівника перед тим, як надати доступ до його електронної пошти. Для випадків, таких як веб-пошта, що використовує розповсюджений клієнт (наприклад, веб-браузер), прямий доступ до програм надає дуже гнучке рішення віддаленого доступу, яке можна використовувати практично з будь-якого клієнтського пристрою. Іншим типовим прикладом прямого доступу до додатків є додаток для смартфонів (клієнтське програмне забезпечення), яке підключається до сервісу, що надається одним з серверів організації через HTTPS.

Архітектура прямого доступу до додатків, як правило, є прийнятною лише в тому випадку, якщо сервери, до яких звертаються працівники віддаленого доступу, розташовані на периметрі мережі організації або в хмарі, але не в внутрішніх мережах. Сервери, які безпосередньо доступні з Інтернету, повинні бути добре захищені, щоб зменшити ймовірність компрометації. Багато організацій надають прямий доступ до додатків лише кількома програмам з низьким рівнем ризику, які широко використовуються, наприклад електронною

поштою, і використовують тунельні або портальні методи для забезпечення доступу до інших програм, особливо тих, які були б надто великими, якщо вони були б доступні прямо з Інтернету.

1.3 Аналіз методів обміну файлами

Обмін файлами – це загальнодоступний чи приватний обмін даними в мережі з окремими рівнями доступності. Залежно від рівнів доступу, обмін файлами дає змогу певній кількості людей читати, переглядати або редагувати файл на основі рівня доступу, наданого розпорядником файлу. Служби обміну файлами, як правило, є масовими (тобто не комерційними рішеннями), публічно доступними і мають обмеження на максимальний об’єм сховища.

Існує безліч різних типів обміну файлів. Ось деякі з найпоширеніших способів зберігання та розповсюдження файлів в Інтернеті: програми на основі протоколу передачі файлів (FTP), peer-to-peer мережі, знімні носії інформації, онлайн сервіси [8].

Найпоширеніша система передачі файлів в Інтернеті на сьогоднішній день відома як File Transfer Protocol або FTP. FTP використовується для обміну файлами між комп'ютерами по локальній мережі і інтернету. Має вбудовану аутентифікацію користувачів. Користувачі можуть отримати доступ до файлів, що спільно використовуються, на сайті FTP-сервера або через будь-який FTP клієнт.

Peer-to-peer мережа надає можливість комп'ютерному обладнанню та програмному забезпеченню спілкуватися без необхідності створення центрального сервера. Файли розміщені на одному комп'ютері та спільно використовуються іншими членами мережі. Немає централізованого серверу збереження даних.

Знімні носії інформації - це передбачає все, що можна від’єднати з пристрою чи комп’ютера. Користувач може передавати або вставляти файли з свого пристрою на знімний носій, а потім фізично передавати його тому, з ким

необхідно поділитися файлами. Вони можуть містити FTP-сервер для цілей безпеки, вимагати наявності дійсного логіну та пароля для доступу.

Сервіси для онлайн обміну даними включають веб-сервіси, які дозволяють користувачам зберігати чи ділитися даними в Інтернеті для особистого або професійного використання. Один учасник може завантажувати фотографії, документи, PDF-файли тощо на платформу для обміну файлами онлайн, яка дозволяє іншим завантажувати ці файли собі за допомогою цієї ж платформи.

Важливо розуміти ризики для підприємств, пов'язані з використання вже існуючих сервісів та додатків для обміну файлами. Ризики включають:

- завантаження файлу, що містить шкідливе програмне забезпечення;
- доступ до файлів поза внутрішньою мережою підприємства;
- використання програм для обміну файлами, яка потребує вимкнути служби брандмауера;
- випадкове розміщення конфіденційних файлів у публічному доступі;
- публічні сервіси не є досить захищеними, оскільки направлені на роботу з обміном файлами, а не їх безпекою;
- публічні сховища є відкрити і можуть бути скомпрометовані.

При використанні публічного сховища з’являються наступні ризики:

– компроментація ключів файлів;

– компроментація метаданих файлів з подальшим таргетованим пошуком даних за метаданими;

– втрата бази даних з ключами та даними користувачів.

1.4 Методи збереження даних

Послуги мережевого зберігання даних зазвичай класифікуються за способом отриманням даних та взаємодією з клієнтом. Існують три види сховищ: файлове, блокове, та об'єктне [9].

Найбільш традиційним типом є спільна файлова система, або просто «сховище файлів», що, як випливає з назви, пропонує багатьом клієнтам можливість доступу до однієї спільної папки. Два найпопулярніші протоколи спільної файлової системи, що використовуються сьогодні, це NFS і SMB/CIFS.

Блок-сховище – це сховище даних, що зазвичай використовується в середовищі мережі зберігання даних (SAN), де дані зберігаються в томах, які також називаються блоками. Кожному блоку присвоюється довільний ідентифікатор, за допомогою якого він може бути збережений і витягнутий. База даних - найпростіший приклад блочного зберігання даних. Кожен блок діє як окремий жорсткий диск і налаштовується адміністратором сховища. Ці блоки керуються серверною операційною системою, і зазвичай мають доступ до протоколів Fiber Channel, iSCSI або Fibre Channel over Ethernet.

Об'єктне сховище є відносно новим типом зберігання, призначеним для неструктурованих даних, таких як медіа, документи, журнали, резервні копії, бінарні файлі програм і зображення VM. Концептуально сховище є схожим на постійний ключ / значення. Зазвичай, доступ до даних (об’єктів) реалізується через виклик REST API (при GET операції повертається ідентифікатор). Більшість об’єктних сховищ дозволяють зберігати метадані об'єктів і агрегувати їх у контейнери (або набори даних, так звані “відра”).

Щодо фізичних носіїв даних: нині світовий бізнес прагне до нової концепції - об'єднання глобального попиту і розсіяної пропозиції. Дана модель стала дуже популярною і довела свою працездатність. Сьогодні вже майже кожен знає про сервіси Uber, Airbnb, Avito, Ebay і т.д. Всі ці сервіси вже давно перейшли на хмарні технології, давши їм непоганий розвиток.

Хмарні обчислення і зберігання даних - метод надання технічних ресурсів в необхідній кількості. Іншими словами, клієнт може зберігати велику кількість даних на віддаленому сервері, не хвилюючись за збереження даних, дороге обладнання та ін.

Багато хто оцінює можливості хмарних сховищ за обсягом даних, який може там зберігається, але хмарні сховища також характеризуються іншими факторами:

- швидкість доступу до даних;
- надійність доступу;
- максимальний обсяг імпортованого або експортованого файлу;
- можливості впровадження сторонніх додатків та модулей;
- рівень безпеки зберігання та передачі даних;
- збереження файлів на певний термін перед їх видаленням;
- гарантіїі страхування безперебійності роботи;
- технічна підтримка.

Існують також і перешкоди, які сповільнюють процес переходу - це, наприклад, страх втрати контролю над даними та неприйняття переваг хмарного сховища вищим керівництвом. Але є і цілком серйозні причини, такі як незадовільна якість інтернет-зв'язку між постачальником послуг і офісами клієнта, висока ціна установки системи, а також деякі дії регуляторів ринку, створюють складності при маніпуляціях з даними;

Особливо важливим фактором є тип сховища. Типи внутрішньої будови системи хмарного сховища бувають наступні:

- файлове - підходить для роботи з документами, таблицями тощо;
- об'єктне - підходить для роботи з неструктурованих даними;
- блочне - підходить для роботи з фінансовою або іншою інформацією, що вимагає складних обчислень;
- агрегатне сховище - варіант сховища блоків, підходить для роботи з інформацією малих обсягів.

«Хмари» бувають декількох типів: приватні сховища, публічні сховища і гібридні сховища.

Приватна хмара – орендований сервіс або ресурси в умовах приватної мережі. Іншими словами, використання індивідуальної, створеною компанією, хмари. У такому випадку компанія отримує більш високий рівень захисту даних, вищу обчислювальна потужність та повне управління над системою. У противагу цьому, при ораганізації власної хмари, компанія погоджується да обслуговування власними силами та на великі витрати на придбання та установку обладнання.

Приватне хмарне сховище варто обирають тоді, коли організації важлива максимальна безпека і конфіденційність зберігання даних.

Публічна «хмара» - це є оренда «чужих» ресурсів. Тобто, компанія звертаєтеся до певного постачальника послуг. Переваги такого підходу - це порівняно невелика вартість. До того ж компанія лише використовуєте сервера, обслуговуванням займаються інші люди, через це публічну хмару досить легко впровадити і використовувати. Але в той же час, така хмара має менш високий рівень захисту, стан системи (доступніть та час відклику) напряму залежить від якості інтернет-з’єднання. А також, компанія не маєте повного контролю над серверами з даними.

Публічна «хмара» найбільше підходить компаніям малого чи середнього бізнесу, у яких є необхідність забезпечити колективний доступ до певних додатків, даних або є потреба щось протестувати, автоматизувати бізнес-процеси.

Гібридне сховище – найсучасніший варіант з усіх перерахованих вище, поєднує в собі як публічне зберігання, так і приватне, що дозволяє дуже гнучко налаштовувати роботу системи, забезпечує простоту використання, підвищує рівень безпеки, в порівнянні з публічним зберіганням. Конфіденційні дані зберагаться у приватній частині хмари з максимальним рівнем безпеки, а відкриті дані зберігаються у її піблічній частині. До недоліків гібридного сховища можна віднести витрати на створення необхідної інфраструктури та установки хмарного сховища. Для підтримки сховища потрібні досвідчені IT-співробітники для обслуговування систем.

Гібридна «хмара» для бізнесу є чимось середнім, перехідним моментом між публічним сховищем і приватним або навпаки.

Часто постає питання про вибір хмарного сховища або звичайного локального сервера. Звичайно, завжди краще мати і локальне і хмарне сховище, щоб бізнес-процеси працювали безперервно, а дані було у безпеці.

Отже, якщо порівняти хмарне та локальні сховища, отримаємо наступне. Хмарні сховища мобільні та доступні завжди і скрізь, при наявності інтернет-зв’язку, їх робота безперевна, вони мають вбудовані функції версіонування файлів та їх резервного копіювання. Завдяки «хмарам» на підприємстві легко організувати спільну роботу.

Локальні сховища є більш простими в установці і використанні, а найголовніше - відносно недорогими. Локальні сховища надійні, компанія має повний контроль над системою. Можливі преривання у роботі.

Таким чином, краще обирати локальне сховище, у випадку невеликиого бюджету, якщо питтання спільної роботи наж файлами не є критичним і об’єми даних що потрібно зберігати є малими (50 - 5000 Гб). У випадку, коли потрібно зберігати велику кількість даних (5 - 100 Тб) або забезпечити мобільність і безперервність роботи, то слід вибрати хмарне сховище.

Проте для реалізації файлового серверу для корпоративної мережі найбільш відповідним варіантом є гібридне сховище, оскільки воно забезпечує конфіденційність даних та сталість процесів, або локальне сховище с реплікацією даних та кластеризацією для підвищення найдійності системи. Якщо компанія має достатньо ресурсів, можна обрати приватне сховище. У будь-якому разі, варінтом який не відповідає вимогам файлового серверу є публічна хмара.

1.5 Аналіз існуючих EFSS платформ

Пошук найбільш безпечного рішення для зберігання даних є дуже актуальною та серйозною темою. Компаніїї занепокоєні убезпеченням корпоративних даних з огляду на те, скільки кіберзлочинів у новинах сьогодні.

Наприклад, витік данних Dropbox 2012 року, взлом iPhone в 2014 році і нанесення атак вірусами типу WannaCry і Petya.

На додаток до витоку даних і вірусів, атаки «людина-в-середині» і «людина в хмарі», які є формами підслуховування передачі даних, зростають, що робить безпеку передачі та збереження даних важливішою, ніж будь-коли.

В даний час існує три моделі захисту для BYOD пристроїв - управління мобільними пристроями (MDM), управління мобільними додатками (MAM) та управління мобільною інформацією (MIM). Ці системи допомагають контролювати пристрої, що належать працівникам, в їхніх особистих і робочих цілях.

Системи для обміну та збереження даних спочатку відносилося до MDM чи MIM систем. MDM та MIM системи зазвичай є досить важкими і складними, а використання не комерційних систем для обміну даними є неприйнятним та банально небезпечним, тому з часом відокремився та сформувався новий сегмент ринку програмного забезпечення, що має назву синхронізація та обмін файлами на підприємстві (Enterprise file synchronization and sharing, також відомт як EFSS).

Сьогодні більшість інструментів EFSS включають шифрування як в режимі передачі даних, так і статичне, щоб утримувати зловмисників від читання приватних файлів. Однак, незважаючи на те, що навіть криптостійкі шифри є вразливими до атаки перебором (brute force), є й інші способи компрометації даних, починаючи зі слабких паролів співробітників.

Платформи EFSS для убезпечення даних пропонують цілий ряд функцій, які допоможуть зберегти цілісність, конфіденційність та увтентичність даних. Такі функції включають двофакторну аутентифікацію, віддалене очищення девайсу, убезпечені центри обробки даних та наскрізне шифрування (end-to-end encryption), такими як Boxcryptor.

Розглянемо наступні популярні рішення для обміну файлами:

- Box Business;
- Dropbox Business;

- Citrix ShareFile;
- Egnyte Connect;
- IBM Connections.

Спершу розглянемо цінову політику застосунків (табл. 1.1). Ціна приблизно однакова, проте напряму залежить від кількості користувачів. Така політика базується на тому, що застосунки надають компаніям лімітовані сховища (або не лімітовані, в залежності від плану) використання яких аналізується та моніториться компанією-постачальником послуги.

Таблиця 1.1 – Цінова політика найпопулярніших EFSS

Назва	Ціна у $ (за одного робітника за місяць)
Box Business	10-25
Dropbox Business	12-20
Citrix ShareFile	15-25
Egnyte Connect	8-15
IBM Connections	6-10

Box Business

Команда Box спеціалізується на бізнес-клієнтах і спеціалізується на забезпеченні максимальної безпеку інтелектуальної власності під час зберігання в хмарі.

Box використовує TLS для передачі даних, що передаються від пристрою до центру обробки даних. Box зберігає ваші дані на серверах в зашифрованому вигляді. Метод шифрування – 256-бітний AES.

Шифрування при передачі даних є ключовим моментом, оскільки він допомагає попередити MITM (атаки man-in-the-middle), які в основному є засобом онлайн-підслуховування.

Box також використовує огортання ключів (key wrapping), що означає не тільки шифрування вмісту, але і шифрування ключа. Для цього Box також використовує 256-бітний AES.

Box інтегрується з Boxcryptor, що дозволяє налаштувати наскрізне шифрування. Наявна інтеграція з єдиним входом (SSO), що дозволяє користувачам входити в декілька бізнес-систем з однаковими обліковими даними, також доступна.

Box дозволяє користувачам використовувати двофакторну аутентифікацію для захисту від слабких паролів. Це змушує користувачів вводити код безпеки, що надсилається на вказаний номер під час входу в Box з незнайомого пристрою.

Box надає можливість встановлювати вимоги до користувацьких паролів. Таким чином, можна запобігти створенню слабких паролів, які можуть мати погані наслідки для бізнесу.

Box також підтримує закріплення пристроїв, що дозволяє обмежити кількість пристроїв, які можна синхронізувати. Можна навіть перевірити, які пристрої використовують ваші користувачі. Таким чином, точно відомо, які пристрої мають доступ до вашого хмарного сховища.

На жаль, від'єднання пристрою лише скорочує доступ до вашого сховища, але будь-який вміст на жорсткому диску пристрою не видаляється, тому це не є ефективним засобом захисту себе в разі втрати або крадіжки пристрою.

В цілому, сервіс забезпечує гарну безпеку, але резервує деякі з найкращих можливостей для своїх великих клієнтів, а не малих бізнесів.

Dropbox Business

Вміст файлів у хмарі Dropbox шифрує за допомогою 256-бітного шифрування AES. Проте метадані файлу залишаються у вигляді звичайного тексту. На жаль, це норма в більшості EFSS, оскільки метадані використовуються для індексування та прискорення роботи.

Dropbox підтримує додаткові функції для убезпечення даних. Перше – двохфакторна аутентифікація (лише у бізнес плані), що вимагає від користувачів введення шестизначного захисного коду для входу при підключенні нового пристрою.

Крім того, у випадку крадіжки синхронізованого пристрою, адміністратор Dropbox може виконати віддалене очищення цього пристрою. Це закриває

доступ до синхронізованої папки і видаляє будь-який вміст всередині неї. Проте будь-які файли, шо були переміщені за межі папки синхронізації, все одно залишаться на пристрої.

Найголовнішою проблемою Dropbox є те, що його було зламано і понад 68 мільйонів адрес електронної пошти користувачів і паролів просочилися в Інтернет. Атака відбулася у 2012 році. У той час Dropbox повідомив, що було вкрадено адреси електронних пошт користувачів, проте він не повідомив про те, що паролі також були вкрадені. З тих пір Dropbox зробив кроки для забезпечення кращої безпеки даних. Для користувачів Dropbox Business ситуація набагато краща, ніж у користувачів Dropbox Plus.

Citrix ShareFile

Файли у хмарі ShareFile зберігаються в затверджених центрах обробки даних, призначених для протидії природним катаклізмам, фізичним атакам і віртуальним атакам.

На серверах ShareFile дані шифруються за допомогою 256-бітової AES. Шифрування використовується при збереженні на даних на сервері (at-rest), ключі зберігаються на окремому сервері, тому при компроментації сховища дані залишуться зашифрованими.

Файли шифруються перед відправленням до сховища ShareFile, кожен з яких має свій унікальний ключ шифрування. Передача файлів також використовує протокол TLS.

ShareFile надає можливість конфігурації вимог до паролів. Також підтримує двофакторну автентифікації та контроль кількості невдалик спроб входу в систему.

Останньою функцією безпеки є те, що ShareFile дозволяє віддалено видаляти дані на пристрої. За допомогою цієї функції можна вимкнути синхронізацію для будь-якого пристрою користувача, запобігаючи будь-якому доступу до вмісту хмари з викраденого пристрою.

ShareFile не підтримує так званий “закріплений пристрій". Закріплення пристрою дозволяє обмежити кількість пристроїв, у яких користувач може

встановлювати ShareFile, що, в свою чергу, дозволяє знизити ризик доступу неавторизованих користувачів до вашого вмісту.

ShareFile також не пропонує шифрування на стороні клієнта і не інтегрується з будь-якими інструментами шифрування сторонніх виробників, такими як Boxcryptor.

Egnyte Connect

Egnyte використовує 256-бітовий AES для шифрування даних. Дані залишаються захищенними під час передачі між пристроями та хмарними серверами Egnyte завдяки TLS, а HTTPS та FTPS допомагають додатково захистити подальші зв'язки. Файли також шифруються за допомогою протоколу шифрування AES-256 під час передачі даних (завантаження, синхронізація) та для зберіганні на хмарі. Багато фінансових установ та урядових організацій використовують лише 128-бітове шифрування, хоча 256- бітове шифрування є стандартним серед послуг хмарного сховища та підвищує стійкість алгоритму.

Після прибуття до центру обробки даних більшість служб EFSS розшифрують файли, щоб витягти його метадані, перш ніж заново зашифрувати вміст файлу. Egnyte не має можливості так робити, оскільки використовується end-to-end шифрування. Egnyte зберігає ваш пароль, тому він називається платформою з нульовим знанням (zero-knowledge platform.). Egnyte дозволяє захистити вміст від загрози слабких паролів з можливістю:

- встановлення обов'язкових вимог щодо міцності пароля;
- вимоги скидання стандартного пароля;
- обмеження кількості невдалих спроб аутентифікації;
- двофакторної аутентифікація.

Egnyte Connect також інтегрується з Boxcryptor (опціонально), який дозволяє шифрувати файли самостійно, перш ніж надсилати їх у хмару. Таким чином, Egnyte не може використовувати клієнтські ключі шифрування, які він зберігає, для розшифрування файлів. Це, у свою чергу, перешкоджає службам передавати ваші приватні дані уряду для виконання програм спостереження. Для більшого контролю за паролями вможна також інтегрувати Egnyte з

послугою "єдиний вхід" (single-sign-on, SSO). Якщо ви використовуєте інші платформи, які інтегруються з SSO, співробітники можуть використовувати той самий пароль для всіх платформ.

Egnyte також дозволяє відстежувати пристрої працівників, які синхронізуються з вашим хмарним сховищем і розривають це з'єднання. Таким чином, якщо пристрій втрачено або викрадено, можна звести до мінімуму шанси на несанкціонований доступ до даних.

Нарешті, Egnyte Connect дозволяє налаштувати гібридне сховище. За допомогою гібридної пам'яті можна зберігати деякі дані в хмарі та деякі дані на власних приватних серверах. Таким чином, якщо компанія має особливо конфіденційну інформацію, вона можете підтримувати повний контроль над вмістом сховища і встановлювати власні протоколи безпеки.

IBM Connections

Будь-які дані, що зберігаться в IBM Connections Cloud, реплікуються на резервних серверах. Самі сервери зберігаються в убезпечених центрах обробки даних, здатні витримувати пожежі, землетруси та інших катастрофи.

Перед тим, як перенести дані з вашого пристрою в хмару, дані шифруються в 128-бітному AES. Хоча 256-бітна AES є більш безпечною і більш поширеною в секторі хмарних сховищ, немає реальних причин лякатися рішенням IBM про використання 128-бітного AES.

Теоретично, для того, щоб зламати 128-бітові AES шифр найсучаснішим суперкомп'ютером у світі - потрібно дуже багато часу.

IBM Connections забезпечує шифрування в режимі передачі даних та at-rest шифрування (захищає ваші дані від потенційних порушень цілісності сервера).

Однією з недоліків є те, що IBM Connections не підтримує двофакторну аутентифікацію (2FA) та конфігурацію вимог до пароля.

При порівняльному аналізі увага звертається не на весь аспект функцій додатку, а на складову убезпечення даних.

Такі функції як закріплення девайсу, управління паролями та віддалене очищення девайсу відносятся до категорії Mobile Device Management (MDM) платформ, проте знаходять своє використання і у EFSS системах.

У таблиці 1.2 розглянуто критерії порівняння існуючих систем. Для кожного крітерію проставлено приорітет у рамках розв'язуваної проблеми виходячи із ризиків віддаленого доступу розглянутих у пункті 1.2 та ризиків використання публічних (не комерційних) рішень пункту 1.3 [10]. Було вирішено у пропонованій архітектурі покладатися на перші 7 пріоритетних критеріїв.

Таблиця 1.2 – Порівняння функцій найпопулярніших EFSS

	Box Business	Dropbox Business	Citrix ShareFile	Egnyte Connect	IBM Connections	Пріоритет критерію
Шифрування (At-rest, In- transit)	AES 256	AES 256	AES 256	AES 256	AES 128	3
Шифрування даних про користувачів (не файлів)	-	-	-	+	-	2
Відокремлення сховища даних та ключів	-	+	+	-	+	1
Прямий доступ до сховища даних	-	+	+-	-	+	6
Двох-факторна аутентифікаці я	+	+	+	+	-	11
Користувацькі налаштування паролів	+	-	+	+	-	13

Продовження таблиці 1.2

Збереження відкритих метаданих	+	+	+	+	+	4
Віддалене видалення даних	-	+	+	+	-	8
Закріплення головного девайсу	+	+	-	-	-	12
SSO	+	-	+	+	-	9
End-to-end шифрування (Boxcryptor)	+	+	–	+	-	5
Wrapping key	+	-	-	-	-	10
Очищення завантажених файлів	-	-	+	-	-	7

1.6 Висновки до розділу

Виходячи з ризиків віддаленого доступу розглянутих у пункті 1.2 та ризиків використання публічних (не комерційних) та не спеціалізованих рішень з пункту 1.3 приходимо до висновку про необхідність додаткових систем забезпечення файлообміну компанії та доступу до її внутрішньої мережі - EFSS систем.

Для зменшення розглянутих ризиків система обміну даними має задовольняти наступним критеріям:

– відокремлення сховища даних (файлів) від сховища даних про користувачів;

– статичне шифрування даних про користувачів;

– зберігання метаданих файлів окремо від самих файлів;

– наскрізне шифрування;

– відсутність прямого доступ до сховища даних.

Існуючі системи задовольняють описані критерії лише частково, тому необхідно розробити власне архітектурне рішення для вирішення задачі безпечного обміну файлами, що відповідає заданим критеріям.

Шифрування статичних даних та відокремлення метаданих від файлів збільшить час пошуку та завантаження файлів, проте це свідоме рішення, оскільки основною задачею системи є саме убезпечення даних.

Проаналізувавши особливості корпоративної мережі, можна прийти до висновку що такі мережі реалізують можливості для віддаленого доступу до них, проте, технології дистанційного доступу та віддаленого доступу побудовані на можливостях такої мережі часто потребують додаткового захисту, оскільки природа корпоративних мереж зазвичай пов'язує їх із зовнішніми загрозами більше, ніж технології, доступні лише з самої організації.

Виходячи з проаналізованих існуючих методів віддаленого доступу, для задачі обміну даними у рамках корпративної мережі найбільш підходящим є тунелювання. Тунелі використовують шифрування для захисту конфіденційності та цілісності переданої інформації між клієнтським пристроєм і сервером. Тунелі також можуть аутентифікувати користувачів, забезпечувати контроль доступу (наприклад, обмежувати, які протоколи можуть використовуватися або які внутрішні хости можуть бути досягнуті за допомогою віддаленого доступу) і виконувати інші функції безпеки. Клієнтське програмне забезпечення і завантажені дані перебувають на клієнтському пристрої, тому вони не захищені тунельним рішенням і повинні бути захищені іншими засобами. Тому цей підхід необхідно модифікувати, надавши спеціалізоване клієнтське забезпечення, що може працювати зі з’єднання через тенулювання, наприклад VPN (різні його протоколи) або Reverse Proxy. Клієнтський додаток має виконувати роль шифруючого контейнеру, який не дозволяє завантаженим файлам залишати межі контейнеру.

Сервери віддаленого доступу, які безпосередньо доступні з Інтернету, повинні бути добре захищені, щоб зменшити ймовірність компрометації.

Виходячи з проаналізованих існуючих методів обміну файлами, найбільш підходящим є онлайн сервіс, проте він має розташовуватись у периметрі корпоративної мережі з обмеженим брандмауером доступом.

Для отпимальної швидкості роботи рішення нам необхідний об’єктний тип сховища, а для максимальної захищенності даних необхідно розгорнути це сховище або у гібридній моделі сховища або у локальному сховищі.

РОЗДІЛ 2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

У першому розділі було розглянуто існуючі системи та оцінено їх функції, виділено найважливіші критерії. Деталі архітектурних рішень системи та, особливо, протоколи взаємодіїї між компонентами системи є комерційною таємницею, тому вони не розголошуються. Через це при реалізації власної системи необхідно власноруч виділити основні компоненти системи, визначити прокололи взаємодії та обрати засоби криптографічного захисту.

2.1. Засоби криптографічного захисту

TLS з клієнтською авторизацією

Для авторизації пропонується використати TLS з клієнтською авторизацією.

TLS (англ. Transport layer security – Протокол захисту транспортного рівня), як і його попередник SSL (англ. Secure sockets layer - шар захищених сокетів), – криптографічний протокол, що забезпечує захищену передачу даних між вузлами в мережі Інтернет. TLS і SSL використовують асиметричне шифрування для аутентифікації, симетричне шифрування для конфіденційності та коди автентичності повідомлень для збереження цілісності повідомлень [11].

Сертифікати SSL / TLS зазвичай використовуються як для шифрування, так і для ідентифікації сторін.

Підтвердження автентичності клієнта – це взаємна автентифікація, заснована на сертифікаті, де клієнт надає сертифікат клієнта серверу для підтвердження своєї особи. Оскільки у контексті розроблюваної архітектури клієнтський додаток має бути авторизований, необхідно використовувати TLS з підтвердженням автентичності клієнта.

Клієнтський сертифікат – це X.509 цифровий сертифікат, такий самий, як звичайний серверний сертифікат.

Аутентифікація клієнта SSL / TLS, як випливає з назви, призначена для клієнта, а не для сервера. За допомогою серверних сертифікатах клієнт (браузер) перевіряє особу сервера. Якщо сертифікат є валідним (підписаний довіреним CA), браузер продовжує роботу - обирає версію протоколу, алгоритми шифрування та підпису повідомлень, встановлює зашифроване з'єднання. Весь процес відбувається під час рукостискання SSL/TLS (рис. 2.1).

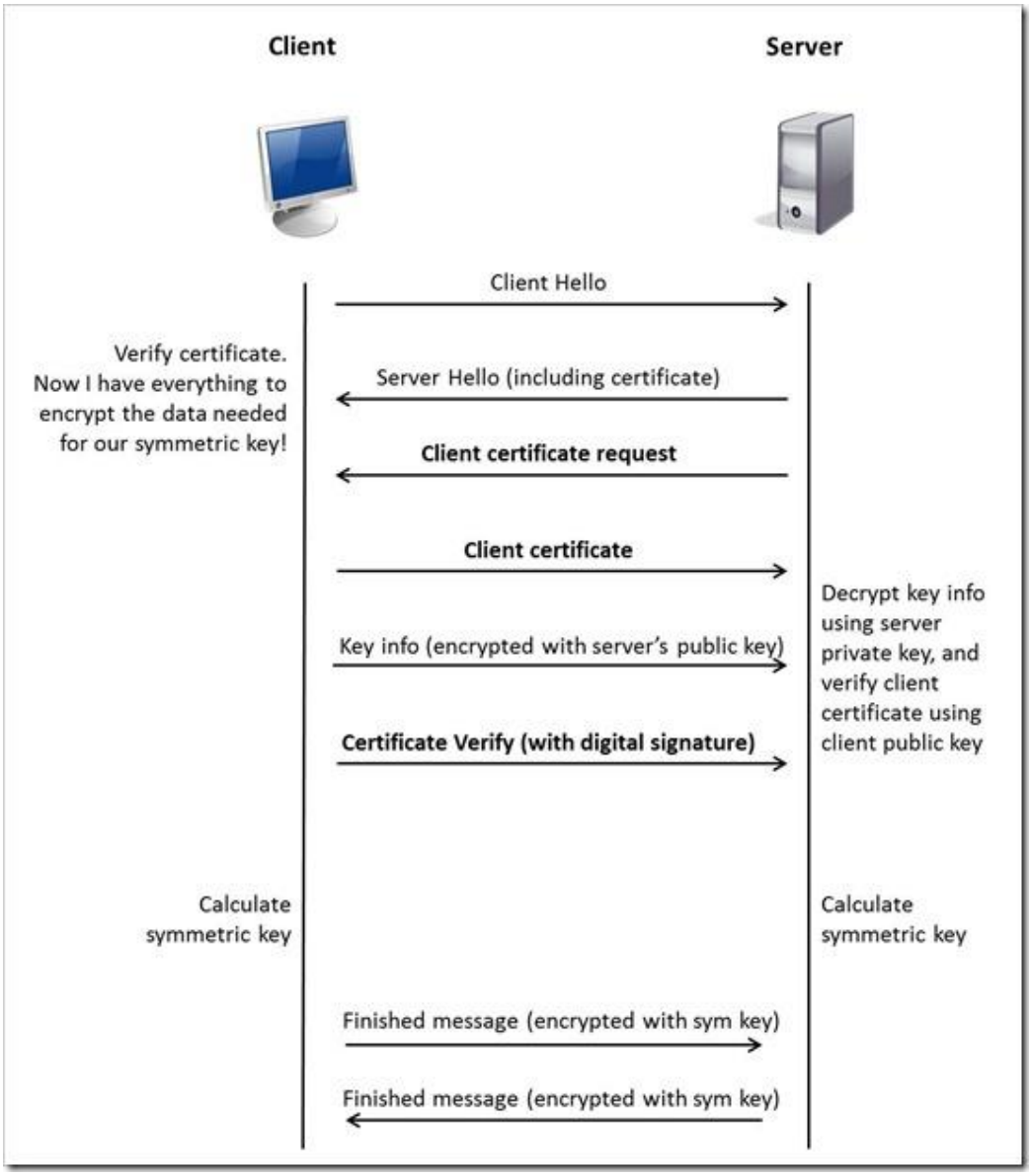


Рисунок 2.1 – Рукостискання при двосторонньому TLS

Типовим застосуванням клієнтської автентифікації є ситуації, де потрібно обмежити доступ до серверу лише колу аутентифікованих користувачів. Це дуже корисно проти атак, що із сторонніх джерел (неаутентифікованих

користувачів). Зловмисники, як правило, грають у імітаційну гру, викрадаючи дані користувачів. Саме тому паролів для аутентифікації недостатньо. Ось чому такі технології, як двофакторна аутентифікація, набирають популярності.

У контексті TLS лише у клієнта є приватний ключ, тому необхідність пароля може бути усунена для деяких систем. Однак це не рекомендований підхід. Використання обох разом може забезпечити високу безпеку, яку надзвичайно важко зламати. Саме такий підхід бажано використати для реалізації доступу до внутрішньої мережі для мобільних клієнтів.

Json Web Token

JSON веб-токен (JWT) – авторизаційний токен призначений для передачі підписаних «заявок» (claims) між службами (як зовнішніми, так і внутрішніми для застосунку/сайту). «Заявки» - частина інформації, яку інші можуть переглядати та/або перевіряти, але не змінювати. Щоб ідентифікувати/здійснити автентифікацію користувача у застосунку, треба розмістити стандартизований токен у заголовок або url сторінки. Варіант з заголовками є більш прийнятним та зручним. Таким чином засвідчується, що користувач авторизувався та може переглядати бажаний контент [12].

Токен JWT складається з трьох частин: заголовок (header), корисне навантаження (payload) і підпис або дані шифрування. Перші два елементи - це JSON об'єкти певної структури. Третій елемент обчислюється на підставі перших і залежить від обраного алгоритму (у випадку використання не підписаного JWT може бути опущений). До заголовку і корисного навантаження застосовується алгоритм кодування Base64-URL, після чого додається підпис і всі три елементи розділяються крапками. Отже токен виглядає наступним чином: eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NSIsIm5hbWUiOiJKb2huIEdvbGQiLCJhZG1pbiI6dHJ1ZX0.LIHjWCBORSWMEibq-tnT8ue_deUqZx1K0XxCOXZRrBI

Розглянемо детальніше складові JWT:

– заголовок (Header);
– корисне навантаження (Payload);

– підпис (Signature).

Перша частина JWT — рядок, що закодовує звичайний JSON об'єкт, який описує тип токену, а також використаний алгоритм хешування.

Друга частина JWT формує основу токена. Довжина корисного навантаження пропорційна кількості даних, збережених у JWT. Загальне правило: зберігати мінімум у JWT.

Третя і кінцева частина JWT — підпис, згенерований на основі заголовка та корисного навантаження підписаний або зашифрований алгоритмом вказаним у заголовках. Використовується для перевірки валідності та оригінальності JWT.

Зазвичай використувається наступна інформація у тілі токену:

– iss – постачальник токена;

– exp – дата закінчення терміну дії (якщо термін дії вийшов — токени будуть відхилені). За специфікацією зазначається в секундах;

– iat – час створення токена. Можна використовувати для визначення віку JWT;

– nbf – «not before» (не раніше). Зазначення моменту активації токена у майбутньому;

– jti – унікальний ідентифікатор JWT. Потрібен, щоб запобігти повторному використанню JWT;

– sub – описуваний об'єкт (рідко використовується);

– aud – одержувачі (рідко використовується).

На відміну від JWT, більшість веб додатків використовують Cookie (кукі) - невеликий фрагмент даних, відправлений веб-сервером, який зберігається бразуером на комп'ютері користувача. Веб-клієнт (зазвичай веб-браузер) кожен раз при спробі відкрити сторінку відповідного сайту пересилає цей фрагмент даних веб-сервера в складі HTTP-запиту і застосовується для:

– аутентифікації користувачів;

– зберігання персональних налаштувань користувача;

– відстеження стану сеансу доступу користувача;

– ведення статистики про користувачів.

JWT має ряд переваг над кукі:

– при використанні кукі, сервер повинен зберігати інформацію про виданих сесіях, в той час як використання JWT не вимагає зберігання додаткових даних про виданні токени: все, що повинен зробити сервер – це перевірити підпис;

– сервер може не займатися створенням токенів, а надати це зовнішнім сервісам;

– у JSON токенах можна зберігати додаткову корисну інформацію про користувачів. Як наслідок - більш висока продуктивність. У разі використання кукі іноді необхідно здійснювати запити на отримання додаткової інформації. При використанні JWT ця інформація може бути передана в самому токені;

– JWT уможливлює надання одночасного доступу до різних доменів і сервісів;

– сервер не має зберігати сесію користувача.

Тунелювання

Виходячи з проаналізованих методів віддаленого доступу до мережі у першому розділі найоптимальнішим для поставленої задачі є тунелювання. Для підтвердження пропонованої концепції найпростішим способом тунелювання є VPN. Існує багато реалізацій VPN, але які найчастіше використовувані для віддалених працівників - це Internet Protocol Security (IPsec), OpenVPN і Secure Sockets Layer (SSL) тунелі [13].

OpenVPN – безкоштовне рішення з відкритим вихідним кодом, яке, за визнанням більшості фахівців, є найкращим на сьогоднішній день для створення приватної віртуальної мережі (VPN). OpenVPN не входить до складу стандартних дистрибутивів сучасних операційних систем, тому вимагає установки додаткового програмного забезпечення, проте додатки працюють на всіх сучасних операційних системах: Windows, macOS, Linux, Android, iOS. При правильному налаштуванні його не зможуть розшифрувати ні спецслужби, ні зловмисники [14].

Сьогодні VPN-провайдери пропонують 128 і 256-бітові ключі. Математики-криптологи станом на 2018 рік вважали, що ключ довжиною 128 біт можна розшифрувати тільки в спеціальних датацентрах на дуже великих потужностях і це займе дуже багато часу. Комп'ютер, здатний розшифрувати дані, зашифровані ключем 256 біт, на думку вчених, поки що не винайдено. За твердженнями багатьох VPN-провайдерів, ключ довжиною 128 біт є оптимальним рішенням у плані безпеки і продуктивності. При використанні 128-бітного ключа шифрування відбувається швидше, ніж при ключі в 256 біт, створює менше навантаження на сервер і на пристрій користувача.

OpenVPN має бути налаштований так, щоб авторизувати клієнта за сертифікатом підписаним довіреним CA.

Відповідно мобільному додатку необхідні лише ip-адреса VPN серверу та його тип.

Зберігання паролів

На сьогоднішній день найбільш стійким варіантом зберігання паролів у реляційних базах даних є використання хеш функції з криптографічною сіллю.

У криптографії хеш-функція являє собою математичний алгоритм, який відображає дані будь-якого розміру у бітовий рядка фіксованого розміру. Вихід хеш-функції називається хешем. Відомо багато алгоритмів калькуляції хешів наприклад MD5 чи SHA-1. Наведемо приклади хешів для паролю “password”, використовуючи ці алгоритми:

– SHA1: 5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8;
– MD5: 5f4dcc3b5aa765d61d8327deb882cf99.

Проте на сьогоднішній день ці алгоритми вважаються не стійкими та не рекомендованими для використання [15].

Критографічна сіль – рядок даних, який передається хеш-функції разом з паролем. Сіль має генеруватися випадковим чином використовуючи криптостійкий алгоритм геренації випадкових даних.

Сіль використовується для ускладнення перебору по словнику і атак з використанням райдужних таблиць, а також приховування однакових паролів.

Однак сіль не впливає на складність повного перебору для кожного окремого пароля. Сіль зберігається у відкритому вигляді поряд із паролем.

Щоб зловмисник не міг створити таблицю пошуку для кожної можливої солі, сіль повинна бути довгою. Хорошим правилом є використання солі такого ж розміру, що і вихід хеш-функції. Наприклад, вихід SHA256 становить 256 біт (32 байти), тому сіль повинна бути не менше 32 випадкових байтів.

Багато користувачів мережі користуються тим самим паролем для більшості веб-сайтів та ресурсів. Це означає, що якщо веб-сайт, що зберігає ваш пароль у простому тексті, скомпрометований, хакери оримують доступ не лише до облікового запису скомпроментованого ресурсу, а й до інших ресурсів з ідентичними даними аутентифікації.

Існують великі бази даних з відкритими паролями, а також райдужні таблиці співпадіння паролів та їх хешей. Тому при компроментації бази даних з захешованими паролями, прообраз хуш функції може бути підібраний.

Природа криптографічного хешу означає, що зловмисники не можуть знайти прообраз хешу, але при виборі поганого пароля він може бути підібраний.

Швидкість підбору паролів обмужується лише потужністю процесора.

Хакери можуть спробувати підбирати пароль зі словника найбільш вживаних паролів (наприклад файл rockyou з 14 млн записами) із відповідною сіллю з бази даних.

Словники паролів, або алгоритми для створення паролів для підбору, як правило, організовані так, щоб найбільш часто використані паролі опиняються якомога раніше у словнику. Це означає, що користувачі зі слабкими паролями будуть скомпрометовані надзвичайно швидко.

Проте навіть при мільйоні тестів хешування паролів за секунду, правильно підібраний пароль залишатиметься недоступним майже невизначений час. На основі набору символів A-Za-z0-9 існує більше тисячі мільйонів паролів з 12 символами.

До того ж, має сенс уповільнити атаки в режимі офлайн, запустивши алгоритм хешування паролів як цикл, який вимагає тисяч окремих хеш-розрахунків або використовувати навмисно повільну хеш функцію.

Набільш відомими хеш-функціями є PBKDF2, bcrypt, scrypt. Рекомендованим є PBKDF2 (рис 2.2), оскільки він заснований на хеш примітивах, які відповідають багатьом національним та міжнародним стандартам.

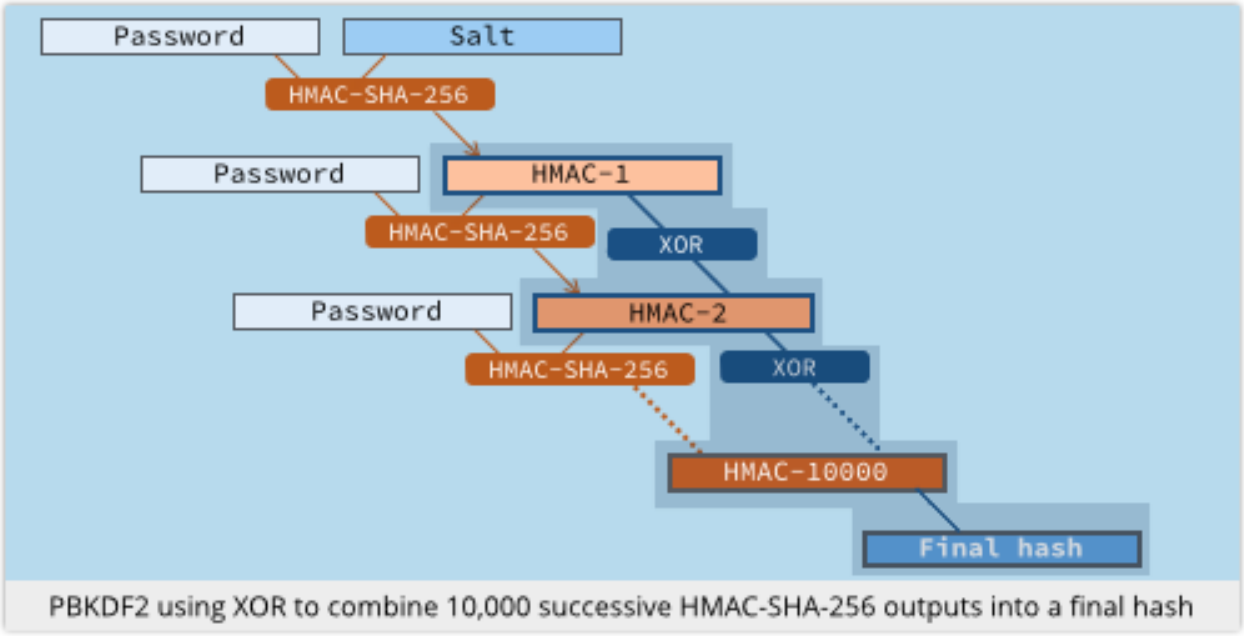


Рисунок 2.2 – Циклічне хешування на прикладі PBKDF2

Отже, мінімальні рекомендації щодо безпечного зберігання паролів користувачів у системі, наведено нижче [16]:

– сильний генератор випадкових чисел, щоб створити сіль у 16 байт або більше;

– HMAC-SHA-256 в якості основного хешу всередині PBKDF2;

– 80 000 ітерацій або більше;

– використати 32 байти (256 біт) виводу з PBKDF2 як кінцевий хеш пароля;

– зберігання кількісті ітерацій, солі та кінцевого хеш у базі даних паролів;

– регулярно збільшувати кількість ітерацій, щоб не відставати від швидших інструментів для підбору пароля.

2.2 Розробка архітектурного рішення

Побудова власної архітектури для реалізації EFSS є достатньо складною та творчою задачою. Існуючі системи не розголошують деталі реалізації, оскільки це є приватною та інтелектуальною власністю компанії-представника послуг. До того ж, задачу завжди можна вирішити декількома способами, які є більш або менш оптимальними в залежності від умови та задач системи.

Виходячи з проаналізованих існуючих методів обміну файлами, найбільш підходящим є онлайн сервіс, проте він має розташовуватись у периметрі корпоративної мережі з обмеженим брандмауером доступом.

Виходячи з аналізу можливих варіантів доступу до корпоративної мережі, найбільше вимогам відповідає тунелювання. Для демонстрації концепції вирішено використовувати VPN.

Для отпимальної швидкості роботи рішення нам необхідний об’єктний тип сховища, а для максимальної захищенності даних необхідно розгорнути це сховище або у гібридній моделі сховища або у локальному сховищі.

Для реалізації поставленої задачі необхідно мати три сервери (не включаючи сервер віддаленого доступу - VPN): AuthServer, DistributionServer та FileServer. Також можна розширити цей список або власним Certificate Authority сервісом для реалізації власної інфрастуктури відкритих ключів, або інтеграцією функціональності CA у DistributionServer.

Пропонована архітектура має на меті нівелювати чотири основні ризики пов’язані з BYOD пристроями: відсутність засобів фізичної безпеки, незахищені публічні мережі, заражені пристрої у внутрішніх мережах, зовнішній доступ до внутрішніх ресурсів.

У контексті пропонованої архітектури видалення файлу можливе лише його власником, а редагування - фактично є копіюванням файлу з подальшою

його зміною іншою людиною, шо стає власником нової копії файлу, тим самим зберігаючи історію файлу.

Для реалізації поставленої задачі файлообміну між мобільними клієнтами за межами корпоративної мережі пропонується мікросервісна архітектура, що включає три сервіси (не включаючи сервер віддаленого доступу): AuthServer, DistributionServer та FileServer .

Пропонована система передбачає шість учасників: користувач мобільного додатку (власник або користувач даних), адміністратор, FileServer, AuthServer, DistributionServer, сервер для віддаленого доступу. На рисунку 2.3 зображено пропоновані компоненти та їх взаємодію.

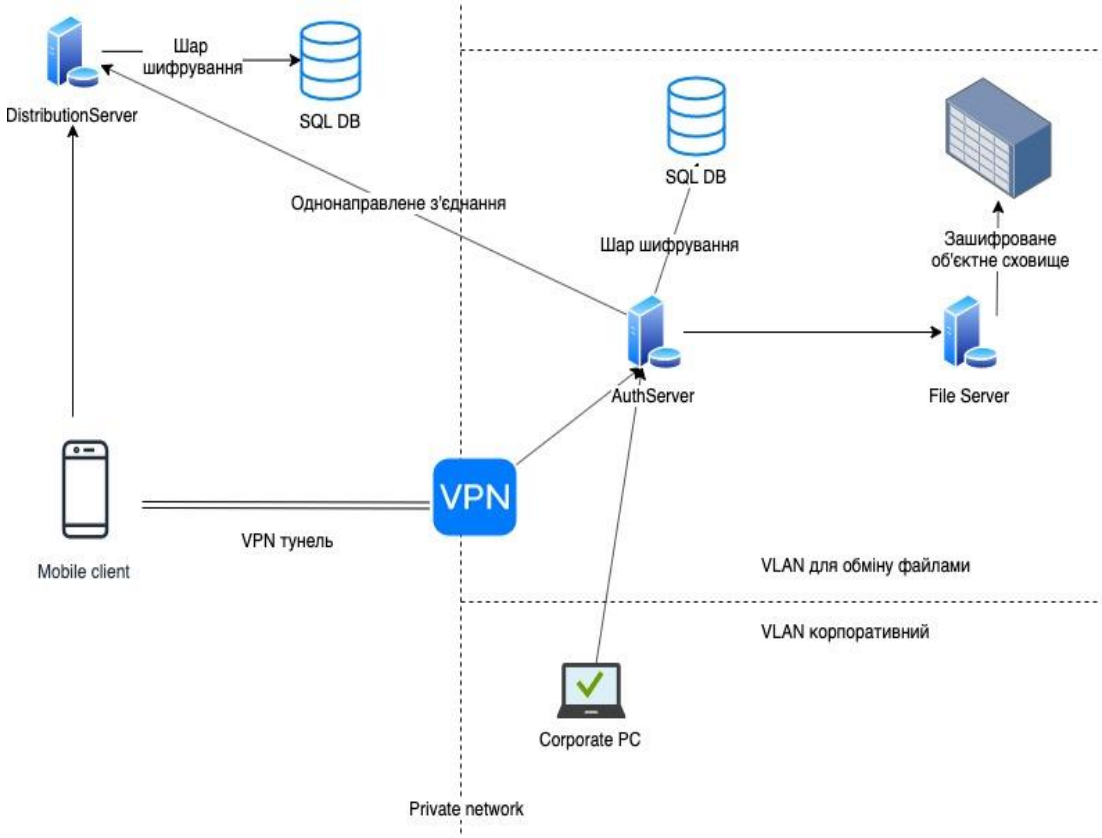


Рисунок 2.3 – Загальний огляд пропонованої архітектури

FileServer

FileServer – сховище даних - сервер єдиною задачею якого є збереження зашифрованих даних за певним ідентифікатором та повернення цих даних за вимогою. Сервер не має жодної інформації про файли та їх власників. Не зберігає метадані. Жодний девайс не має прямого доступу до файлового сервісу та не знає

про його існування та розташування. Комунікує лише з AuthServer по REST API з двостороннім TLS.

Розташування серверу: за брандмауером всередині корпоративної мережі.

AuthServer

AuthServer – сервіс управління системою, з яким напряму працює адміністратор системи. Сервер використовується для адміністрування користувачів, управління доступом, управлінням ключами та шифруванням файлів. Ще одна задача сервісу - відправляти сповіщення користувачам на вказану електронну пошту, наприклад, при реєстрації або при отриманні нових доступів до файлів. Лише адміністратор реєструє нових користувачів або розширює їх права. Комунікація з базою даних відбувається через проксі бази даних із сильним селективним шифруванням [17].

AuthServer виступає медіатором і перенаправляє зашифровані дані від мобільних клієнтів до FileServer. Шифрування файлів є наскрізним. Алгоритм шифрування та розшифрування не розглядається у поточної статті.

Розташування серверу: за брандмауером всередині корпоративної мережі.

AuthServer має дані для з’єднання з FileServer, а також зберігає налаштування серверу віддаленого доступу (має реалізовувати тунелювання, для спрощення реалізації використовується VPN) для доступу у корпоративну мережу.

DistributionServer

DistributionServer – сервер що відповідає за направлення мобільного клієнта у відповідну мережу організації.

Розташування серверу: публічний інтернет.

Цей сервер є необхідним, оскільки встановлений мобільний клієнт не має жодної інформації про підприємство, до якого відноситься майбутній користувач. Система має однозначно виявити організацію за наданими користувачем даними.

Саме тому DistributionServer є спільним для всіх підприємств та попередньо відомий мобільному клієнту та усім AuthServer-ам. DistributionServer

видає ліцензії AuthServer-ам, реєструє нові AuthServer-и та зберігає дані про підключення до них, по запиту клієнта авторизує його та повертає дані про сервер та параметри з’єднання (VPN налаштування). Комунікація з базою даних відбувається через проксі бази даних із сильним селективним шифруванням.

Також включає у себе функціонал Certificate Authority для підписання сертифікатів мобільних додатків [18].

2.3 Протокол комунікації між компонентами

У розглянутій архітектурі відбувається комунікація між наступними компонентами:

- DistributionServer - AuthServer;
- DistributionServer - мобільний додаток;
- AuthServer - FileServer;
- AuthServer - мобільний додаток.

Оскільки AuthServer та FileServer знаходяться у межах однієї мережі, їм достатньої побудованого двостороннього TLS каналу.

Додаткового убезпечення вимагає канал з’єднання між DistributionServer з AuthServer та між DistributionServer з мобільним додатоком.

У обох випадках сервером виступає DistributionServer. Діаграму послідовності алгоритму взаємодії між DistributionServer та AuthServer продемонстровано на рисунку 2.4. Для авторизації клієнтів пропонується використати підхід підпису повідомлень. Для підпису запитів пропонується використовуватся Ed25519 (Edwards-curve Digital Signature Algorithm) з передачею підпису у хедері запиту. Публічний ключ надається DistributionServer у момент першого з’єднання (коли AuthServer та мобільним додаток можна ідентифікувати за іншими даними). У відповідь DistributionServer видає унікальний ідентифікатор клієнта, для того, щоб знаходити відповідність між підписом та його власником.

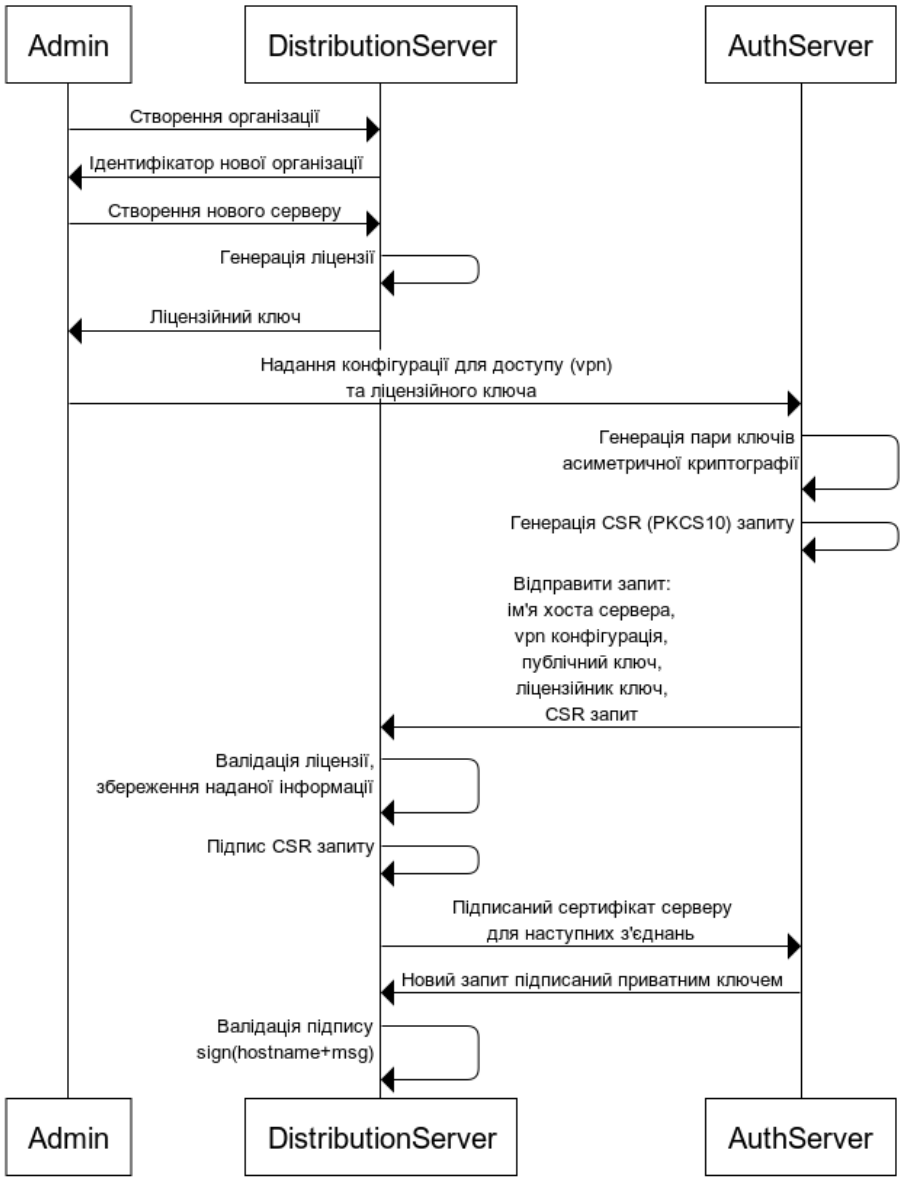


Рисунок 2.4 – Діаграма послідовності взаємодії між серверами

2.4 Алгоритм публікації та завантаження файлів

Для завантаження та публікації фалів найважливішим є принцип шифрування та розшифрування даних. Для підтвердження концепції використовується AES у режимі CTR, проте спроектована система має підтримувати різні алгоритми шифрування.

Було розглянуто алгоритм використання асиметричного ключа з розділенням його навпіл між автором та отримувачем [19]. Алгоритм має недоліки, такі як відсутність синхронізації ключів, а також неможливість

реалізації наскрізного шифрування. Тому було розроблено власний алгоритм передачі та зберігання файлу.

Алгоритм завантаження та шифрування виглядає наступним чином:

– користувач обирає файл для завантаження;

– клієнтський додаток ініціює обмін ключами з AuthServer;

– клієнтський додаток шифрує файл використовуючи дані з попереднього етапу і передає файл на Auth Server;

– клієнтський додаток окремо пересилає метадані файлу та інформацію для доступу;

– Auth Server отримає файл, розшифровує його ключем з пункту 2;

– Auth Server генерує новий ключ для симетричного шифрування, шифрує файл та відправляє його на FileServer;

– мобільний клієнт надає інформацію про доступи до файлу;

– Auth Server повідомляє усіх користувачів про нове надходження файлу з унікальним ідентифікатором для його завантаження.

На рисунку 2.5 наведено діаграму послідовності алгоритму.

Хоча перешифрування файлу є не продуктивним, проте таким чином реалізується наскрізне шифрування, а також надається додатковий шар шифрування для переданих даних, тільки авторизований мобільний клієнт розшифрує файл [20].

Під обміном ключами мається на увазі наступний підхід: за допомогою алгоритму Диффі-Хелмана (Diffie-Hellman) на еліптичних кривих, використовуючи Curve25519 клієнт та сервер обмінюються спільним ключем. Проте цей ключа не можна використовувати одразу, тому для використання AES необхідно надати ключ відповідного розміру. Це можна реалізувати за допомогою функції виведення ключа на основі ключа (KBKDF), оскільки одна з задач такої функції є виведення ключа, придатного для використання в якості введення в алгоритм шифрування.

Прикладом таких функцій є PBKDF2HMAC або HKDF.

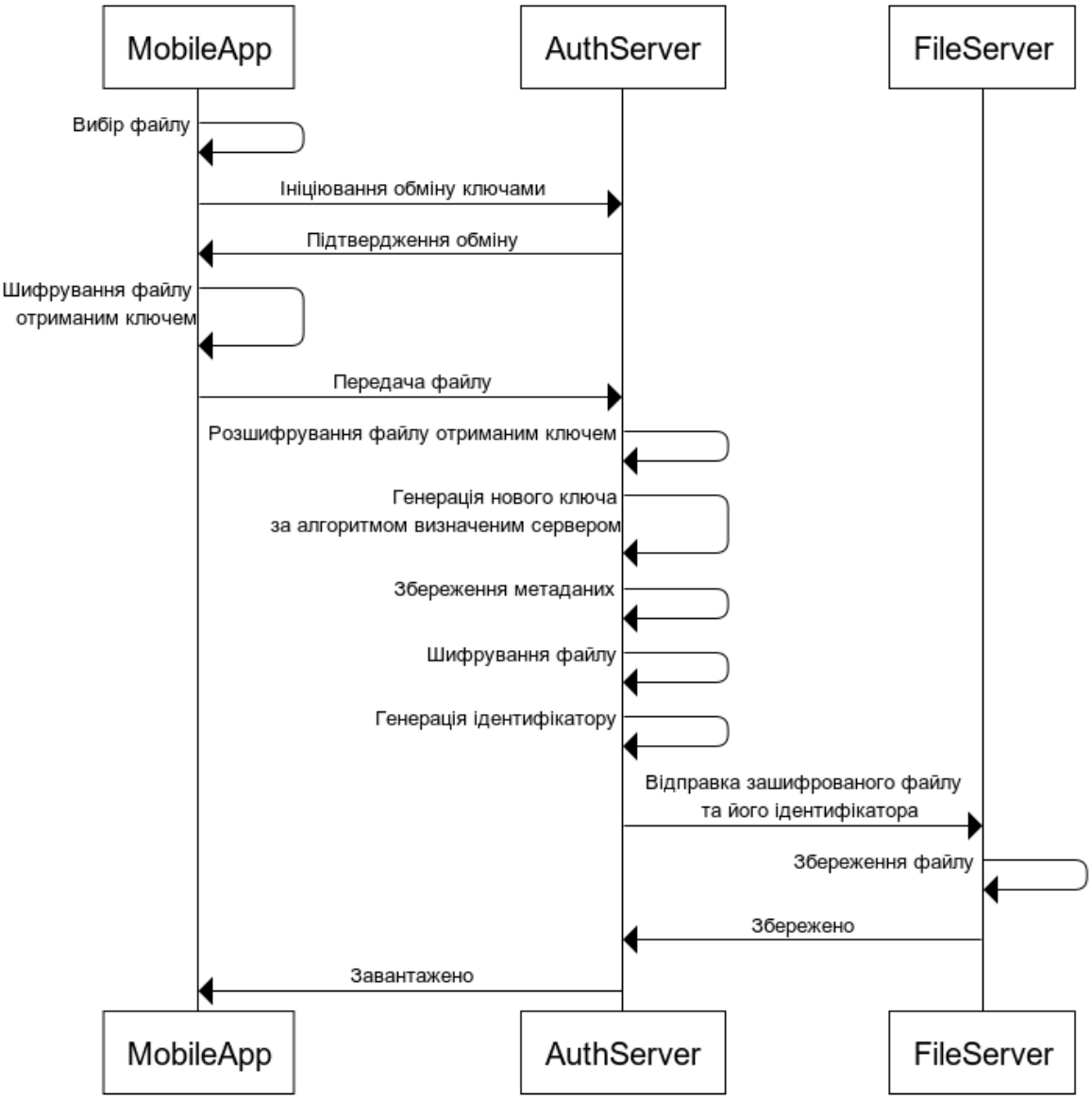


Рисунок 2.5 - Діаграма послідовності завантаження файлу

AuthServer зберігає поточний ключ у відповідності до сесії клієнта, якщо сесія була обірвана - ключ видаляється і необхідно повторити алгоритм з самого початку.

Розшифрування файлу відбувається аналогічно, з використання ідентичних алгоритмів, у зворотньому порядку.

Користувач, що отримав повідомлення про новий доступ до файлу, авторизується у мобільному додатку та вводить ідентифікатор файлу, який має бажання завантажити.

Якщо аутентифікація пройшла успішно, то мобільний додаток розпочинає процес завантаження файлу:

– клієнтський додаток ініціює обмін ключами для файлу. Тільки якщо аутентифікація пройшла успішно і користувач має доступу до файлу, запускається процес обміну;

– AuthServer завантажує запитуваний файл з FileServer та розшифровує його використовуючи ключ з локальної бази даних;

– AuthServer використовую спільний ключ з кроку 1, шифрує файл та відправляє його клієнту;

– клієнтський додаток розшифровує файл та відображає його клієнту.

Лише один користувач є власником даних, всі інші користувачі – тільки переглядають іх. При необхідності редагування необхідний запит має бути схвалений власником. Після цього файл копіюється та надається новому власнику.

2.5 Алгоритм першого доступу до системи

Оскільки AuthServer знаходиться у рамках корпоративної мережі, а мобільний клієнт поза її межами, виникає проблема, як безпечно організувати перший доступ до AuthServer. Пропонований алгоритм має на меті надати користувачу мобільного додатку доступ до відповідного AuthServer, що обслуговує організацію користувача. Для функціонування алгоритму необхідно виконання двох передумов: інсталяція Auth Server та реєстрація користувача адміністратором. Після цього можливий доступ з мобільного додатку.

Інсталяція AuthServer

Передбачається, що адміністратор встановлює сервер інсталятором наданим компанією-розробником.

Адміністратор отримує ліцензійний ключ з DistributionServer та вводить його у процесі інсталяції. AuthServer передає на DistributionServer наступну інформацію: VPN налаштування, URL AuthServer, CSR запит, ліцензійній ключ

та публічний ключ для підпису даних у наступних з’єднаннях. У відповідь AuthServer отримує ідентифікатор, який він додає до наступних запитів.

У завершальному етапі встановлення адміністратор задає логін та пароль для входу до веб-сервісу. Після цих етапів AuthServer є готовим до використання.

Реєстрація користувача адміністратором

Адміністратор реєструє користувача з його персональними даними. Важливо те, що лише адміністратор має права на реєстрацію нових користувачів та адміністраторів.

У процесі реєстрації система генерує одноразовий пароль (OTP), що може бути використаний лише один раз та лише для одного пристрою. Для використання системи на декількох додатках адміністратор додатково генерує пароль для вже існуючого користувача.

Одноразовий пароль – це 14 символьний пароль, що генерується за допомогою криптостійкого генератора випадкових значень, алфавіт якого складається з арабських цифр, великих та маленьких латинських літер.

Кількість можливих варіантів складає (26+26+10) = 62 у 14 ступені. Така кількість комбінацій нівелює можливість атаки перебором, оскільки пароль є актуальним короткий час.

Auth Server відправляє електронну пошту користувача та його одноразовий пароль на DistributionServer. DistributionServer зберігає ці дані для подальшої взаємодії: ідентифікації мобільного клієнта та встановлення відповідності між авторизовним клієнтом та його організацією. Distribution Server генерує випадковий ідентифікатор девайсу та відправляє його AuthServer [21].

Після успішного збереження даних на DistributionServer, AuthServer надсилає на вказану корпоративну пошту користувача інформацію про реєстрацію та одноразовий пароль для авторизації мобільного клієнта.

Доступ з мобільного додатку

При першому доступі з мобільного додатку єдина відома додатку інформація - адреса DistributionServer.

Першим кроком користувач вводить електрону пошту та одноразовий пароль надіслані на корпоративну пошту. Мобільний додаток генерує сертифікат та запит на підпис сертифіката (CSR). Після цього, мобільний додаток відправляє електрону пошту, одноразовий пароль та CSR запит до DistributionServer через TLS тунель. DistributionServer перевіряє отримані дані, якщо вони відповідають запису у базі даних, підписує CSR запит, та відправляє сертифікат з даними для підключення до AuthServer (VPN налаштування та обслуговуючий AuthServer), попередньо видаливши електрону пошту та одноразовий ключ зі своєї бази даних.

Мобільний додаток генерує пароль для авторизації на AuthServer.

Мобільний додаток підключається до серверу віддаленого доступу, використовуючи надану DistributionServer конфігурацію. VPN авторизує клієнта за сертифікатом. Через встановлений тунель мобільний додаток надсилає перший запит до AuthServer з електронною поштою, одноразовим ключем та новим паролем для генерації JWT токену. AuthServer перевіряє надані дані, якщо електрона пошта та одноразовий ключ валідні, одноразовий ключ видаляється, зберігається наданий пароль клієнта. Таким чином, користувач проходить аутентифікацію на VPN та на AuthServer окремо.

Після встановлення з’єднання мобільний клієнт та AuthServer можуть безпечно комунікувати зашифрованим каналом. AuthServer підписує запити авторизованого користувача JWT токеном.

За замовчуванням термін валідності згенерованого пароля - 3 місяці, отже він має бути перегенерований при наближенні дати закінчення дії пароля [22].

Схему алгоритму наведено у додатку А.

2.6 Висновки до розділу

У розділі було загально розглянуто пропоноване архітектурне рішення та його компоненти. Детально описано пропоновані алгоритми (протоколи) першого доступу до системи та завантаження файлів.

Для шифрування даних пропонується використовувати ключи довжиною 256 біт.

Для збереження паролів пропонується використовувати циклічний bcrypt з сіллю довжиною, що дорівнює розміру хеша та з кількістю ітерацій близько 60,000. Наприклад, для bcrypt при вазі у 16 отримаємо 65,536 ітерацій.

Для підпису запитів пропонується використовуватся Ed25519 - що є схемою підпису EdDSA (Edwards-curve Digital Signature Algorithm) та використовує SHA-512 та елептичну криву Curve25519.

Для обміну ключами за Діффі-Хелманом також пропонується використовувати алгоритм з використання Curve25519 еліптичної кривої.

РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

Відповідно до алгоритмів та засобів, описаних в розділі 2 магістерської дисертації, необхідно виконати розробку програмної реалізації відповідної системи захисту.

Розробка виконується за допомогою мови програмування Java. Сервери працюють на вбудованому Tomcat, що поставляється у фреймворком Spring, що також є IoC контейнером та виконує впровадження залежностей. Для описання API та генерації документації використано Swagger 2.0. Swagger UI відображає список наявних методів веб-сервісу та надає можливість їх виконати.

Відповідно до теоретичної частини наведеної в розділі 2, розроблено 3 сервіси, веб-інтерфейс для AuthServer та мобільний додаток.

Загальні сценарії використання надано у додатку А.

3.1 Налаштування VPN

Для взаємодії сервера та клієнта необхідно надати правильну конфігурацію.

У рамках роботи було налаштовано сервер, що реалізує авторизацію за сертифікатами.

Конфігурацію сервера надано у додатку Б.

Опис важливих елементів конфігурації:

- proto udp - використання udp як транспортного протоколу;
- dev tap - створення ethernet тунелю;
- ca ca.crt - сертифікат CA який підписує клієнтські сертифікати;
- key server.key - приватний ключ серверу;
- server 10.8.0.0 255.255.255.0 - опис підмережі для клієнтів з можливими ip;
- keepalive 10 120 - час тримання сесії;

- tls-auth ta.key 0 - ключ для перевірки/генерації HMAC;
- explicit-exit-notify 1 - нотифікація клієнта про завершений рестарт серверу.

Відповідна конфігурація клієнта має той же набір налаштувань, проте додатково містить підписаний сертифікат та приватний ключ, виданий CA.

3.2 FileServer

FileServer є сховищем зашифрованих даних (див. розділ 2). Сховище реалізовано за допомогою MongoDB та GridFS.

MongoDB (від англ. Humongous - величезний) - документоорієнтована система управління базами даних (СУБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць. Класифікована як NoSQL, використовує JSON-подібні документи і схему бази даних.

GridFS - це специфікація MongoDB для зберігання та отримання великих файлів, таких як зображення, аудіофайли, відеофайли тощо. Це своєрідна файлова система для зберігання файлів, але її дані зберігаються в колекціях MongoDB. GridFS має можливість зберігати файли навіть більше, ніж обмеження розміру документа MongoDB в 16 МБ.

GridFS ділить файл на шматки і зберігає кожен фрагмент даних в окремому документі, кожен з максимальним розміром 255k.

GridFS за замовчуванням використовує дві колекції fs.files та fs.chunks для зберігання метаданих файлу та фрагментів. Кожен фрагмент ідентифікується унікальним полем _id ObjectId. Файли fs.files служать батьківським документом. Поле files_id у документі fs.chunks пов'язує фрагмент з його батьківським.

На рисунку 3.1 зображено приклад колекції fs.files заповненої бази даних FileServer. Оберемо для аналізу перший файл з ідентифікатором ObjectId(“5d07af11ec924c1f2aad1ed6”). Не зберігається жодних метаданих, окрім його. Виходячи з розміру файлу у 664705 байт та розміру шматка файлу GridFS

у 261120 байт, вихідний файл розділено на 3 шматки у колекції fs.chunks, що і зображено на рисунку 3.2.

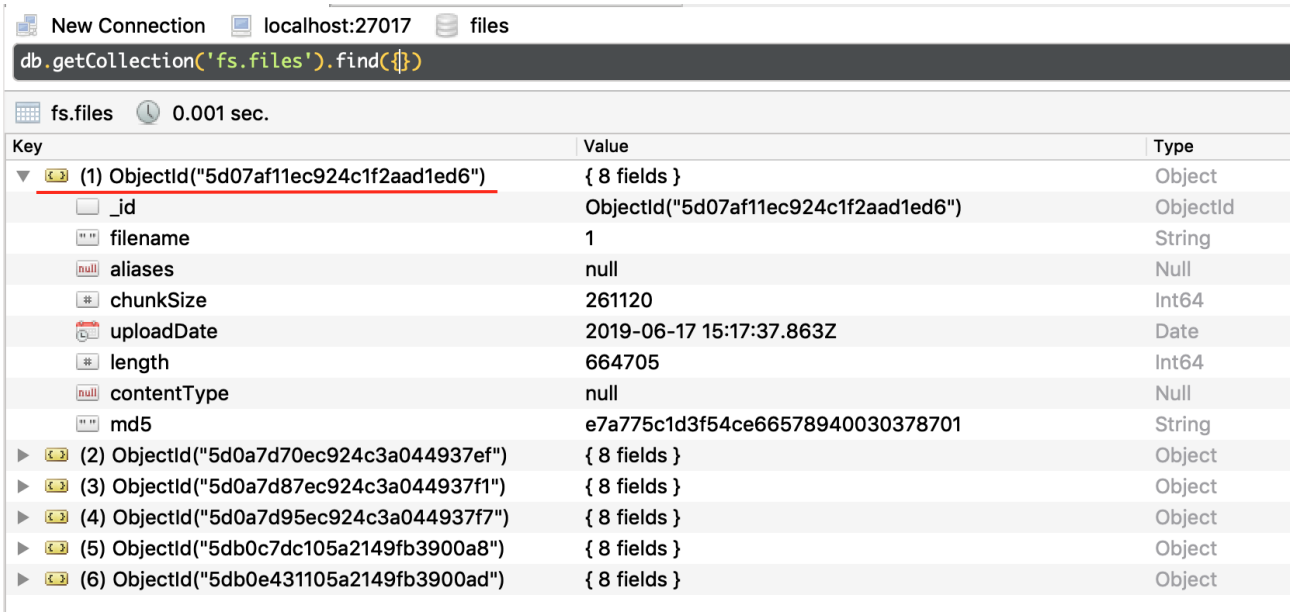


Рисунок 3.1 – Колекція fs.files

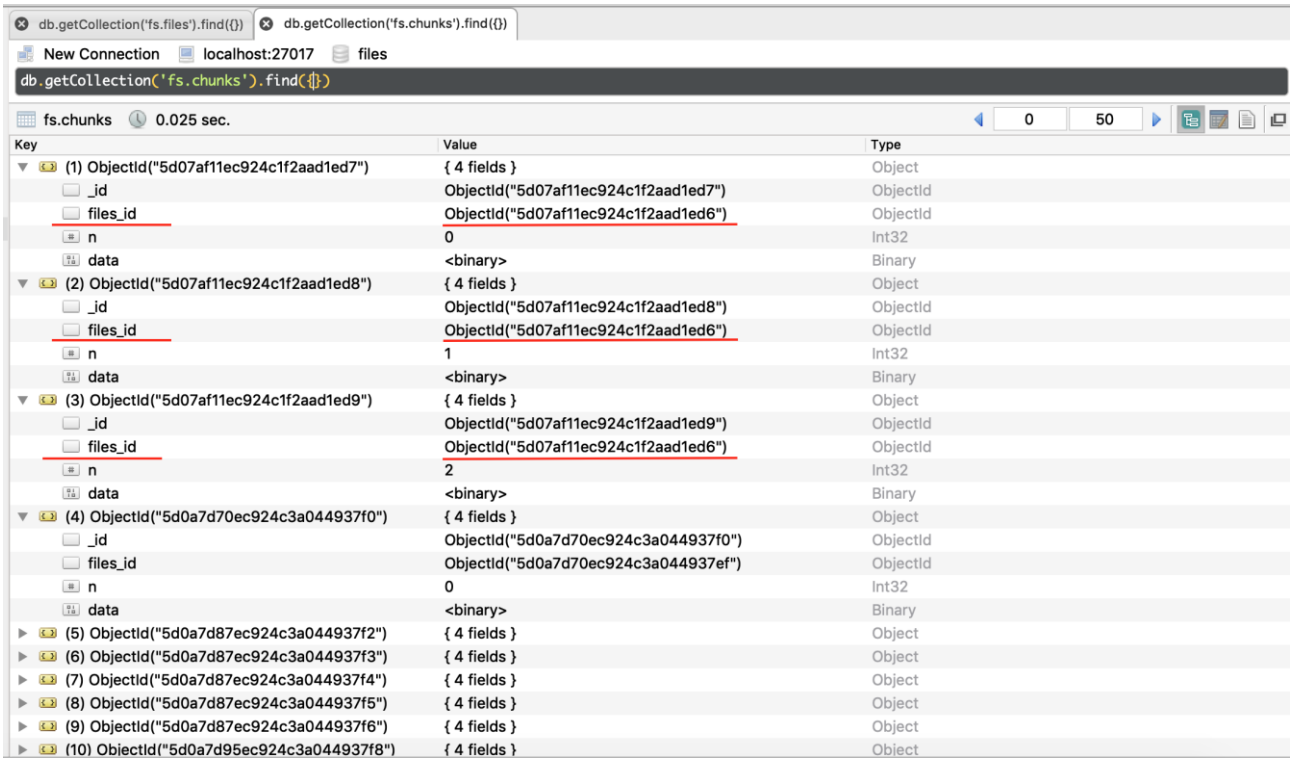


Рисунок 3.2 – Колекція fs.chunks

На рисунку 3.3 зображено список API реалізованого на сервері. Файл можна завантажити, вивантажити, видалити, а також переглянути список усіх файлів.

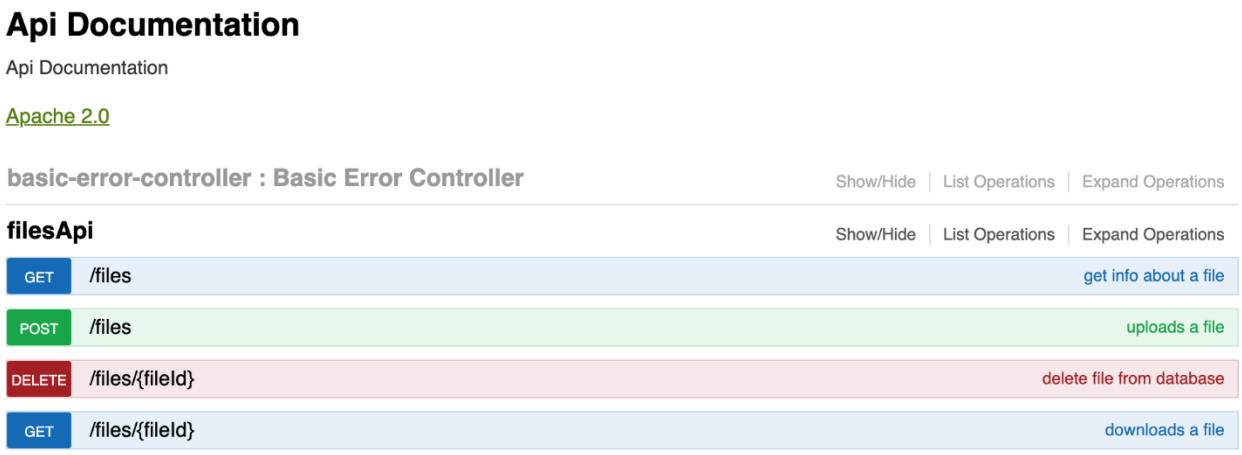


Рисунок 3.3 - Набір API FileServer

Алгоритм шифрування файлів визначається на AuthServer.

3.3 DistributionServer

Як було описано у 2 розділі, DistributionServer є доступним із публічного інтернету та виконує наступні функції: реєстрація організацій, генерація ліцензій, реєстрація серверів та клієнтів, а також розподілення нових клієнтів.

На рисунку 3.4 відображено список API, які надає сервер.

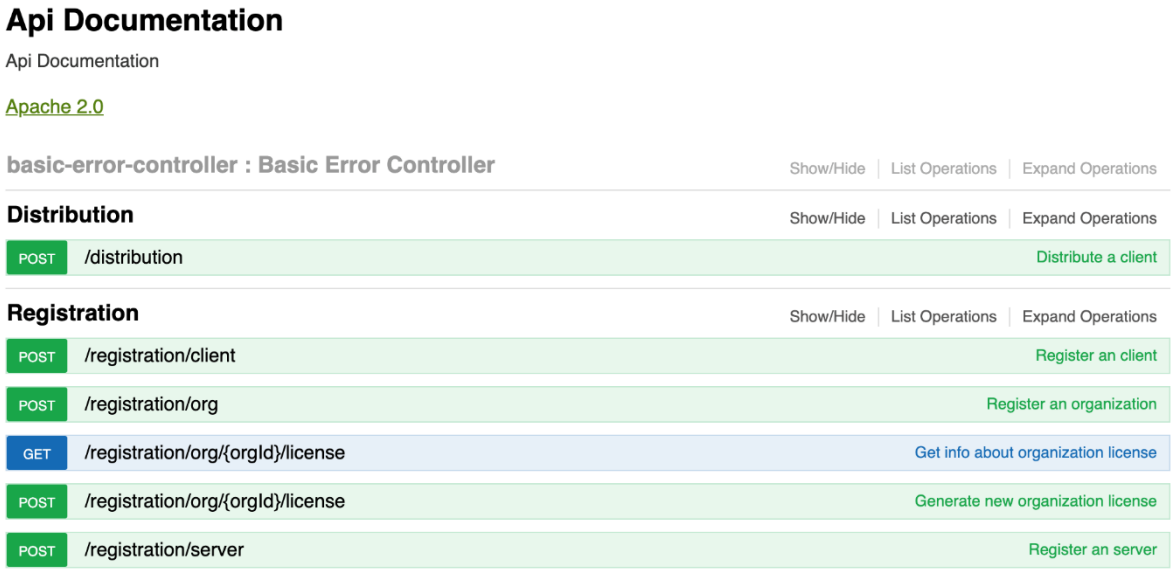


Рисунок 3.4 - Набір API DistributionServer

Важливим елементом працездатності системи є реєстрація AuthServer. Для цього необхідно реалізувати систему ліцензування, яка надать доступ до DistributionServer лише дозволеним серверам зареєстрованих організацій.

Система ліцензування має забезпечити аутентифікований доступ до DistributionServer від AuthServer-ів, а також повинна відповідати деяким основним вимогам, що важливі для розроблюваної системи.

Ліцензійні ключі повинні бути досить простими для введення. А також має бути неможливим розібрати програму (декомпілювати), визначити алгоритм валідації та створити на основі алгоритму робочий «кейген». Це означає, що програма (AuthServer) має перевіряти валідність ключа лише частково.

Для того, щоб задовольнити перелічені вимоги використовується система часткової перевірки ключів (Partial Key Verification System). Оскільки у кінцевій програмі не включено код для тестування ключа, неможливо створити діючий дійсний генератор ключів.

Ця система не є способом повністю запобігти зламам. Зловмисник все ще може редагувати виконуваний файл, щоб змінити код валідації, але тоді він не отримає доступу до DistributionServer.

Отже, ключ складається з 20 символів (для зручності користувача). Дійсний ключ виглядатиме так: A279-1717-7D7A-CA2E-7154. Склад ключа розписано в таблиці 3.1.

Таблиця 3.1 – Формат ліцензійного ключа

Seed value	Key Byte 0	Key Byte 1	Key Byte 2	Key Byte 3	Checksum
A2791717	7D	7A	CA	2E	7154

Схему бази даних зображено на рисунку 3.5

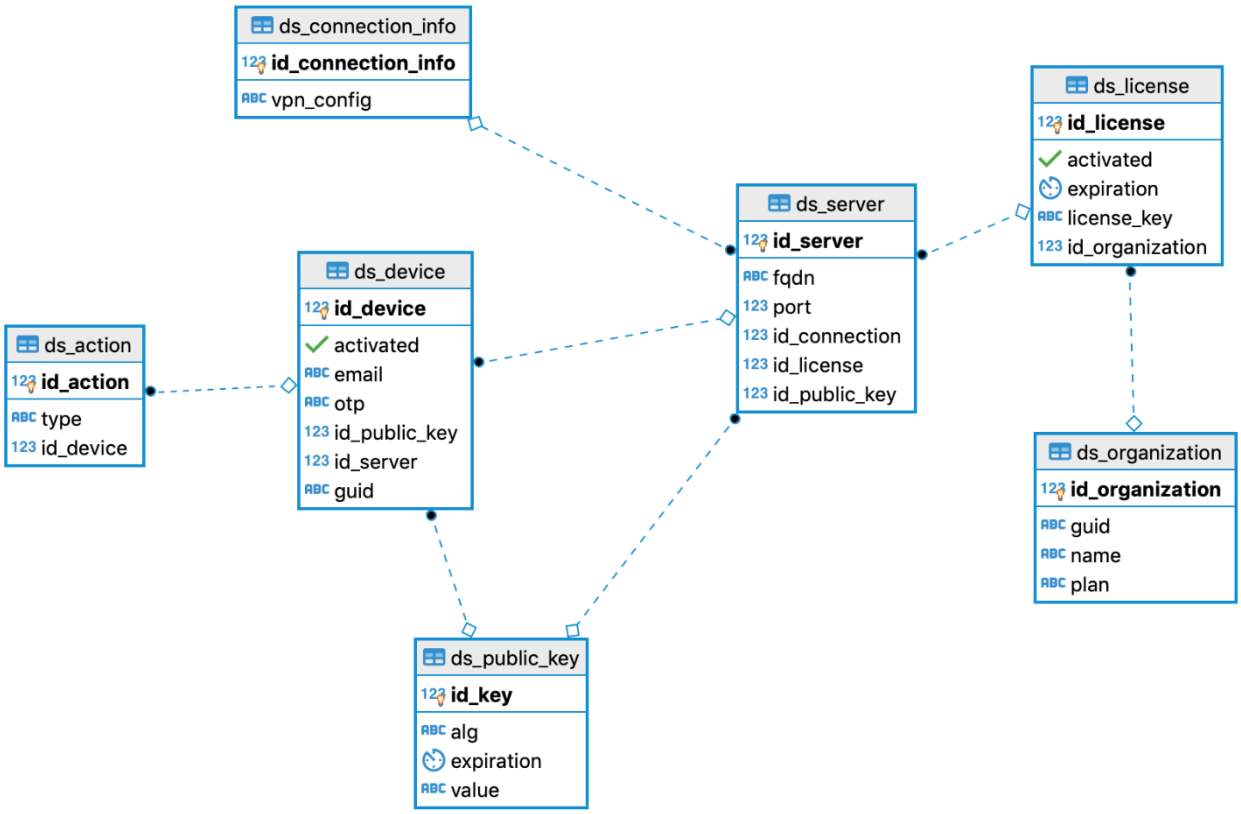


Рисунок 3.5 – ER діаграма сутностей DistributionServer

Усі повідомлення до серверу підписані Ed25519. При обчисленні підпису для запитів від AuthServer додатково додається ім’я хоста та ідентифікатор клієнта. Вигляд запиту наведено нижче.

Заголовки запиту:

- POST /registration/client HTTP/1.1;
- Host: localhost:8181;
- User-Agent: curl/7.54.0;
- Content-Type: application/json;
- Accept: */*;
- X-AUTH: 60454a1118d17bdbf93f845d196715fd1f3c90ebe1a242866d5f2f00cfb3399;
- X-WHO: e541794b-fbbe-4e21-8506-a81a7a9504a7;

- Content-Length: 78;

Тіло запита:

```
{
    "email": “bobkelso@company.com",
    "otp": “M37sAWrUw83QWk”
}
```

Підпис обраховується наступним чином: sign(payload + hostname, private_key). Для мобільних додатків для генерації підпису ім’я хоста не використовується.

У свою чергу DistributionServer верифікує підпис виходячи із збереженої у базі даних інформації про клієнт (ім’я хоста сервера).

3.4 AuthServer

У відповідності до опису архітектури, наданому у другому розділі, AuthServer розташовано у приватній мережі і доступний він лише з під VPN з’єднання. Саме тому при встановлені, адміністратор має надати правильне налаштування VPN, яке надасть змогу віддаленим клієнтам використовувати сервер. ER діаграму сутностей AuthServer наведено на рисунку 3.6.

Надалі AuthServer передає цю інформацію на DistributionServer. Приклад клієнтської документації надано у додатку Б.

Ця конфігурація буде доповнена для кожного клієнта його приватним RSA ключем та сертифікатом, підписаним CA у складі DistributionServer.

Для першого доступу до системи генерується одноразовий пароль, який, як було описано вище, складається з 14ти символів і включає арабські цифри та великі і маленькі латинські літери. Приклад згенерованого одноразового паролю: t2jsJop9CShZRO.

Алгоритм шифрування файлів визначається на AuthServer, проте необхідно зберігати тип алгоритму шифрування для зворотної сумісності різних

версій. Саме тому as_password має такий атрибут, як alg, що визначає криптографічну сутність, для якої було збережено пароль.

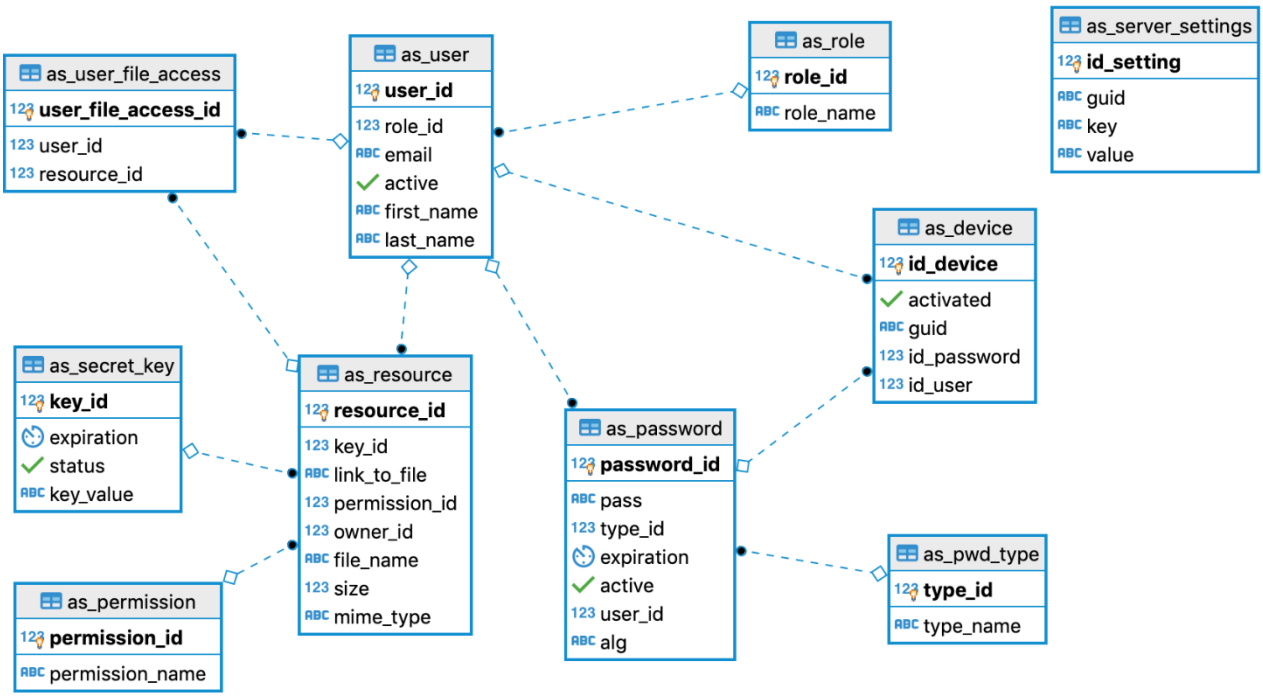


Рисунок 3.6 – ER діаграма сутностей AuthServer

Важливим компонентом розширюваності системи є таблиця з конфігурацією сервера, де можна й майбутньому зберігати такі дані, як кількість допустимих девайсів, кількість спроб введення паролю, термін життя паролей за замовчуванням тощо.

Наразі системою використовується 4 типи паролів:

– DEVICE_PASS - пароль для комунікації з додатком;

– OTP - одноразовий пароль для першої комунікації з додатком;

– SELF_SERVICE - пароль користувача для авторизації на AuthServer через веб додаток;

– DH_SESSION_KEY – сесійний пароль користувача для передачі файлу.

Всі паролі мають час свого життя: OTP - 7 днів, DH_SESSION_KEY – період сесії, інші - 3 місяці. DH_SESSION_KEY шифрується, а інші хешуються.

Для доступу до файлу існує наступний список дозволів:

- ALL_USERS - файл доступний для усіх користувачів системи;
- LIST_OF_USERS – файл доступний для первинного списку користувачів;
- USERS_BY_ROLE - файл доступний для користувачів із вказаною роллю.

Кожен користувач має певну роль:

- ADMIN – адміністратор, створює та видаляє користувачів;
- TECH_SUPP – працівники технічної підтримки;
- DEV – розробники;
- MANAGER – менеджери;
- ACCOUNTANT – бухгалтери;
- USER – усі інші користувачі.

Лише адміністрітор та технічна підтримка мають ширший набір прав для управління системою. Усі інші ролі реалізовано для можливості управління доступами до файлів для працівників на різних рівнях та посадах.

Для збереження паролів, як було описано у попередньому розділі, використовується bcrypt. Паролі зберігаються у базі у наступному форматі (рис. 3.7): $2a$10$igTg2di4vxiJDF08SuUhBumg.CPmgRU6YFJknq/y7HanIxy3MoPgG.

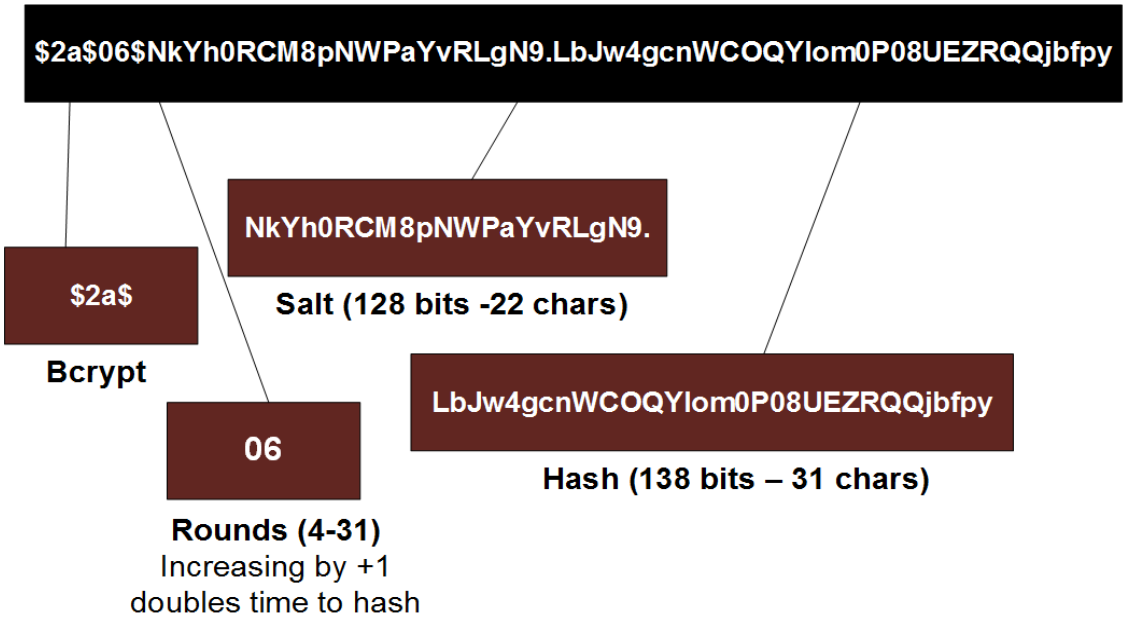


Рисунок 3.7 – Формат зберігання паролей у AuthServer

Хеш розділено на 3 частини. Як проілюстровано на рисунку 3.7, перша частина “$2a$"описує алгоритм хешування, друга частина “$2a$” описує складність - що пропорційне кількості ітерацій. Наступні 128 біт (22 символи) - це сіль, наступні 184 біти (31 символ) - безпосередній хеш паролю.

На рисунку 3.8 продемонстровано приклад збереження паролей для користувача: користувач має активний пароль для входу у веб додаток, девайс користувача активований, тому DEVICE_PASS встановлено, а OTP вже недійсний.

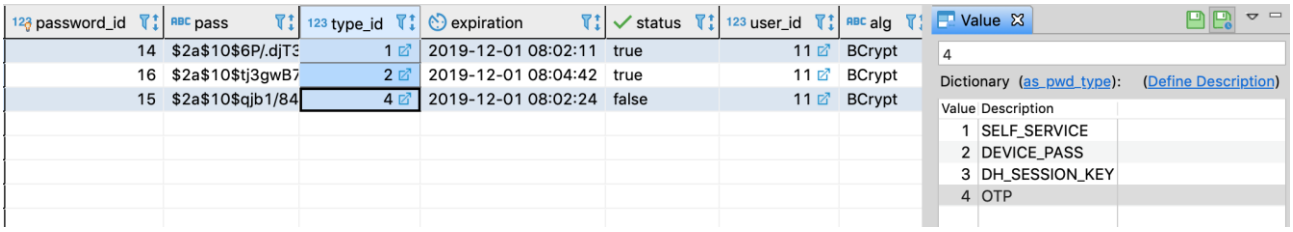


Рисунок 3.8 - Приклад паролей користувача у AuthServer

AuthServer реалізує 3 блоки API: авторизація, робота з користувачами та з файлами (рис. 3.9 - 3.10).

У блоці авторизації знаходяться звичні функції для входу в систему та виходу з неї. При успішній авторизації користувач отримує у відповіді від серверу у заголовку JWT-токен. Приклад токену адміністратора:

eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbkBjb21wYW55LmNvbSIsInJvbGUiOiJBRE1JTiIsImV4cCI6MTU3NDA2NTQxNn0.7VWUqEry8cbrmzJrcdMRL3JJ8QVQsN0yaj_SGODMNxPRKf101AILl_DgK3liP-2BDNiqGonfUxm5idreAAOq9A

У токені зберігаються наступні дані (кому виданий, роль користувача та дата закінчення дії токену):

```
{
  "sub": "admin@company.com",
  "role": "ADMIN",
  "exp": 1574065416
}
```

Функції для управління файлами: завантаження файлів, поширення, обмін ключами для шифрування, повернення списку файлів та їх метаданих, видалення файлів. Функції роботи для роботи з користувачами: додавання, редагування, видалення, виведення списку зареєстрованих.

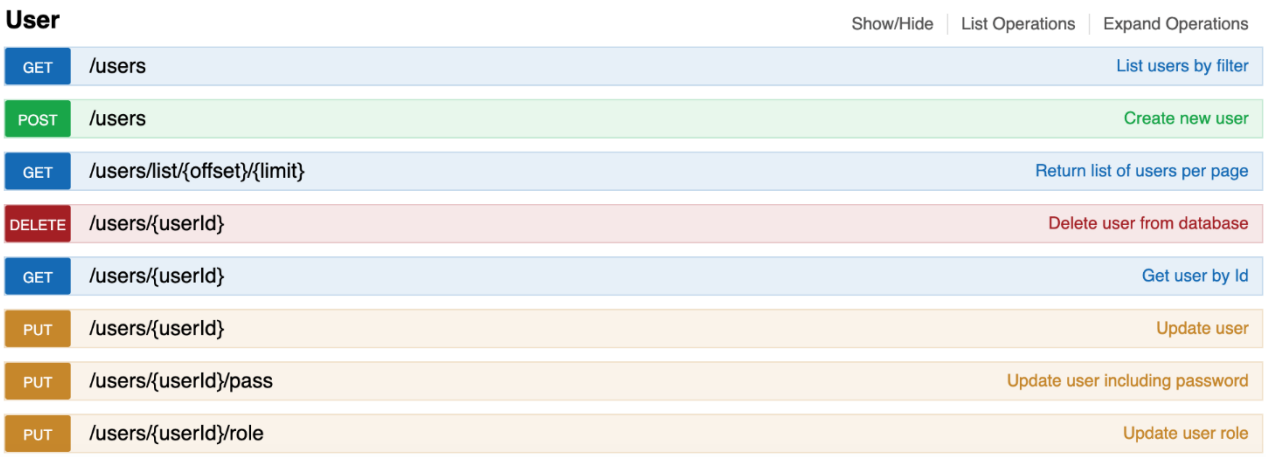


Рисунок 3.9 - Набір API AuthServer для управління користувачами

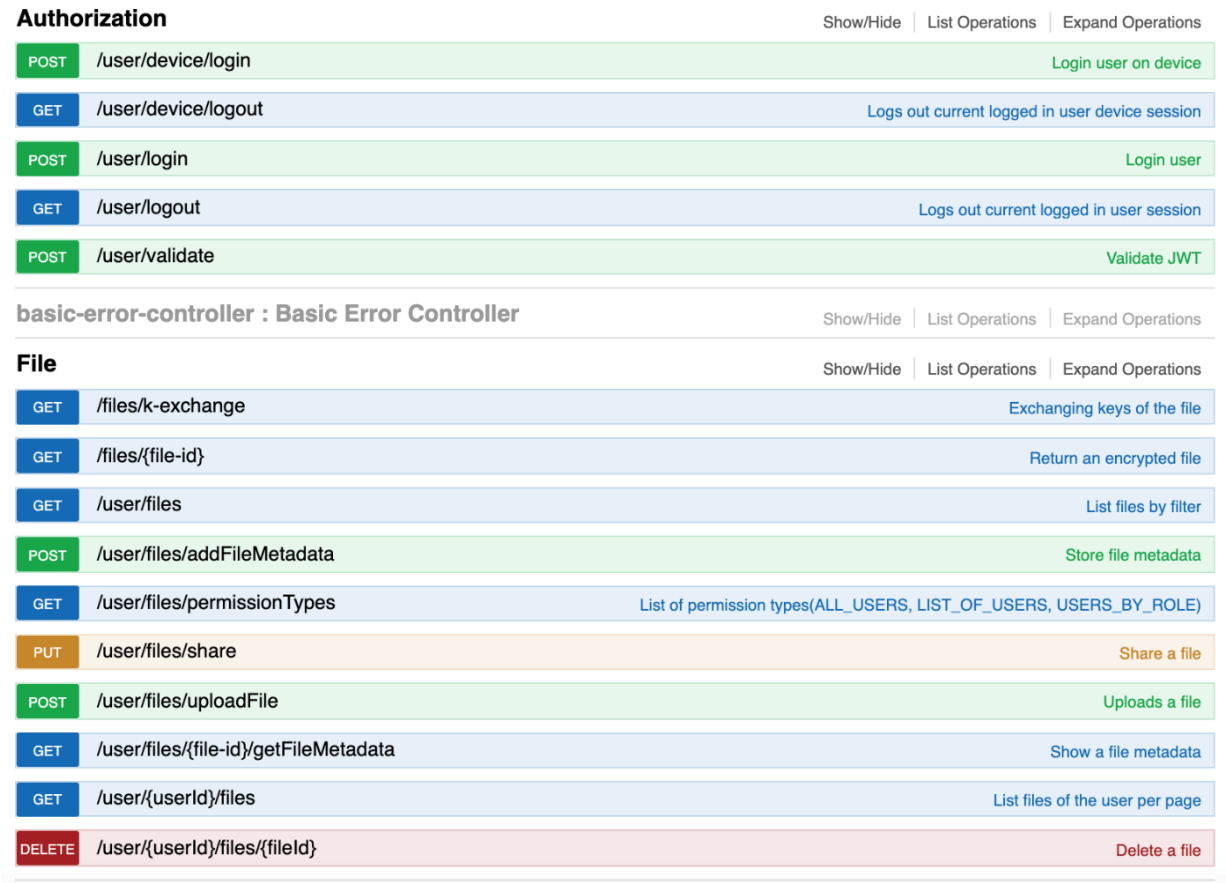


Рисунок 3.10 - Набір API AuthServer для авторизації та роботи з файлами

3.5 Мобільний додаток

Мобільний додаток для підтвердження концепції виконано на платформі Android з використанням VPN.ht, що базується на ics-openvpn, що є відомим OpenVPN клієнтом.

Наголошується на тому, що повноцінний мобільний клієнт у рамках пропонованої системи має реалізовувати функції для відкриття та редагування файлів у додатку, щоб гарантувати надійність зберігання. Проте у додатку, що лише підтверджує працездатність пропонованої архітектури, завантажені файли зберігаються відкрито.

Мобільний додаток передбачає наявність наступних екранів: список файлів, завантаження файлу на сервер, публікація файлу, завантаження файлу з серверу.

3.5 Веб додаток

Веб додаток для підтвердження концепції виконано використовуючи Angular. Реалізовано веб додаток для демонстрації дій адміністратора з AuthServer.

Веб додаток передбачає наявність наступних сторінок: авторизація, додавання користувачів, редагування (паролі, ролі) користувачів, видалення користувачів, перегляд списку користувачів, генерація одноразового пароля, завантаження файлу на сервер, публікація, завантаження файлу з серверу, відображення списку файлів.

3.6 Висновки до розділу

У розділі розглянуто та описано деталі реалізації серверів, надані ER схеми та представлено список наявних функції (API). Описано функції клієнтських додатків.

DistributionServer є доступним із публічного інтернету та виконує наступні функції: реєстрація організацій, генерація ліцензій, реєстрація серверів та клієнтів, а також розподілення нових клієнтів.

FileServer є сховищем тільки зашифрованих файлів. Файлі зберігаються розділеними на шматки розміром у 261120 байт.

AuthServer розташовано у приватній мережі і доступний він лише з під VPN з’єднання. Саме тому при встановлені, адміністратор має надати правильне налаштування VPN, яке надасть змогу віддаленим клієнтам використовувати сервер.

Функції для управління файлами: завантаження файлів, поширення, обмін ключами для шифрування, повернення списку файлів та їх метаданих, видалення файлів.

Функції роботи для роботи з користувачами: додавання, редагування, видалення, виведення списку зареєстрованих.

Мобільний додаток передбачає наявність наступних екранів: список файлів, завантаження файлу на сервер, публікація файлу, завантаження файлу з серверу.

Веб додаток передбачає наявність наступних сторінок: авторизація, додавання користувачів, редагування (паролі, ролі) користувачів, видалення користувачів, перегляд списку користувачів, генерація одноразового пароля, завантаження файлу на сервер, публікація, завантаження файлу з серверу, відображення списку файлів.

РОЗДІЛ 4 ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ, ЩО РЕАЛІЗУЄ РОЗРОБЛЕНУ АРХІТЕКТУРУ

4.1 Опис веб додатку

При запуску додатку вперше, до бази додаються дані адміністратора, з якими можливо авторизуватись у веб застосунку. Користувачам доступна лише сторінка авторизації (рис. 4.1), реєструвати користувачів у системі має право лише адміністратор системи.

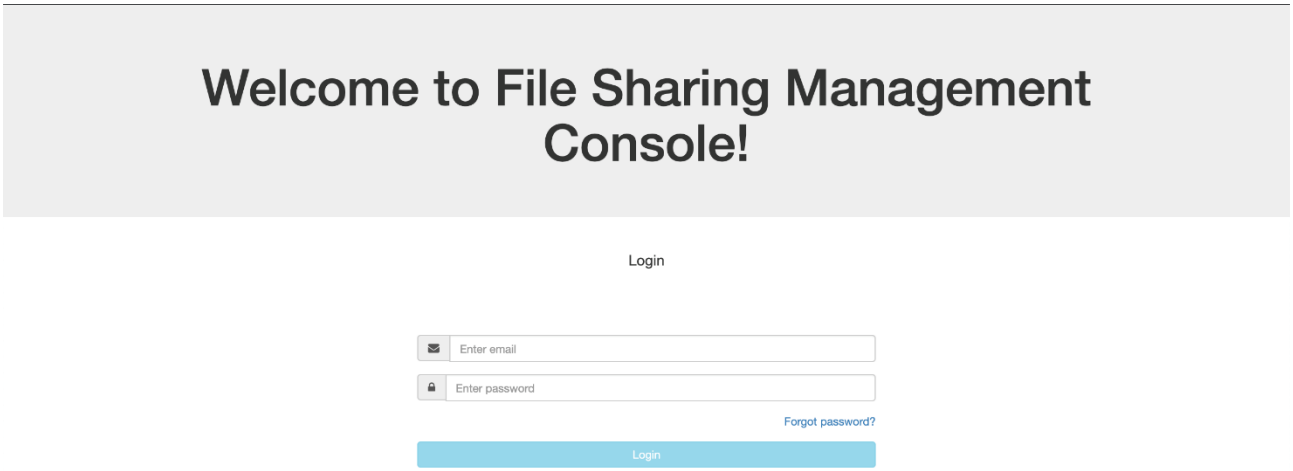


Рисунок 4.1 - Сторінка авторизації

Лише адміністратор має право додавати (рис. 4.2), видаляти користувачів та змінювати їх ролі. При створенні нового користувача вказуються його ім’я, прізвище, електронна пошта, пароль для входу у системи з веб додатку та роль користувача.

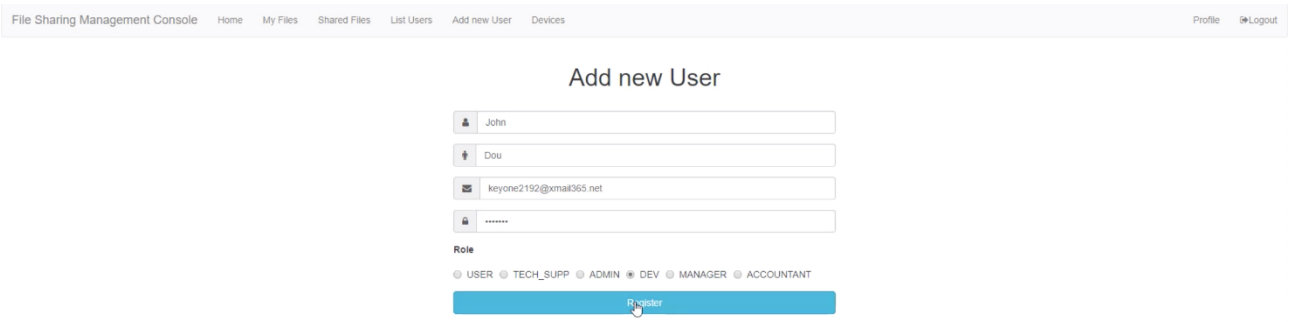


Рисунок 4.2 - Сторінка додавання користувача

При створені нового користувача генерується одноразовий пароль та відправляється на DistributionServer, який, у свою чергу, зберігає ці дані, генерує ідентифікатор девайсу та віддає його у відповіді AuthServer. Після успішної реєстрації користувач отримує відповідне повідомлення на вказану електрону пошту з одноразовим паролем (рис. 4.3).

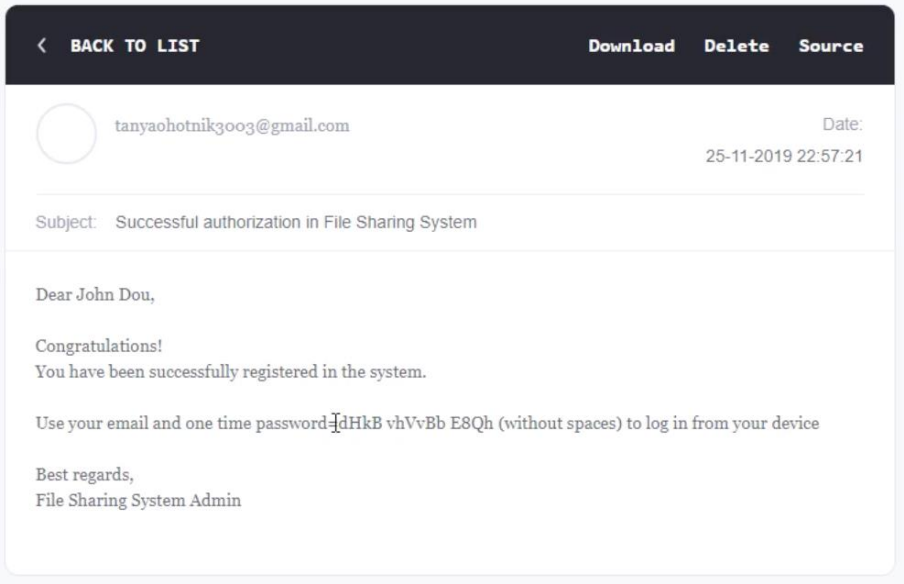


Рисунок 4.3 - Повідомлення про реєстрацію та одноразовий пароль

Користувачі також отримують повідомлення про видалення аккаунту, публікації файлів, закінчення строку дії паролів.

Адміністратор має вкладку для відображення списку усіх активних користувачів (рис. 4.4), з можливими діями: видалення, редагування, генерація одноразового паролю, встановлення нового паролю для авторизації у веб додатку.

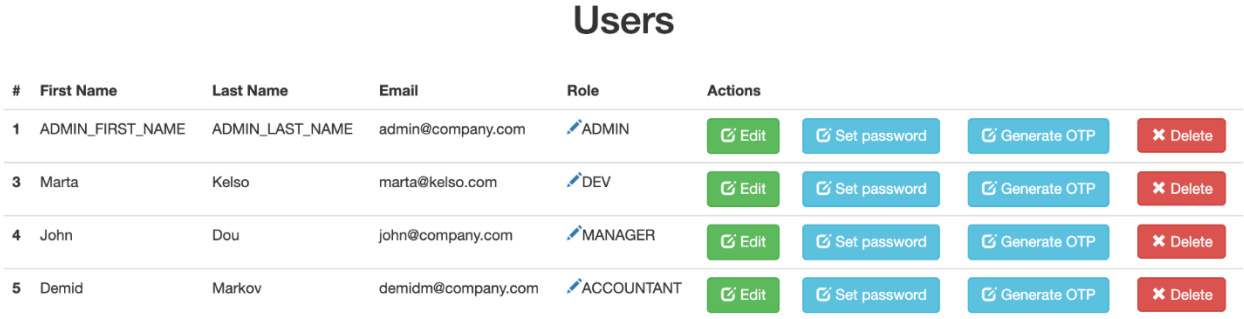


Рисунок 4.4 - Список користувачів

Проте завдяки селективному шифруванню, дані з рисунку 4.4 зберігаються у базі, як показано на рисунку 4.5. Важливі поля зашифровано AES/CTR та переведено у Base64, для того, щоб зберігати дані як текстові з можливістю відключення статичного шифрування у процесі розробки та тестування.

user_id	role_id	email	active	first_name	last_name
1	1	77+9ESzvv73vv70kSe+/vX5N77+9VO+/vU0sOSLvv70=	true	77+9MQzvv73vv707bO+/vUFu7	77+9MQzvv73vv707Zu+/vU
3	4	77+9FDPvv73vv70kQe+/vX9O77+9FO+/vQwiOw==	false	77+9FDPvv73vv70=	77+9EC3vv73vv70=
4	4	77+9Ginvv73vv70ARe+/vT1e77+9V++/vQ==	true	77+9Ginvv70=	77+9GjQ=
5	5	77+9ECzvv73vv70Jau+/vXxQ77+9W++/vRphNSDvv73vv70=	true	77+9ECzvv73vv70=	77+9FDPvv73vv70S

Рисунок 4.5 - Приклад селективного шифрування

З домашньої сторінки сайту можна завантажити файл на сервер або з серверу. Наприклад, на рисунку 4.6 зображено завантаження файлу на сервер.

Рисунок 4.6 - Вибір файлу для завантаження

У результаті файл завантажується на сервер, шифрується, відправляється на FileServer для зберігання. На AuthServer зберігаються метадані про файл (рис. 4.7).

resource_id	key_id	link_to_file	permission_id	owner_id	file_name	size	mime_type
2	2	795b7ec1-c7c3-4559-9176-9ffcf4e47f88	1	1	important.jpg	156,145	image/jpeg
4	4	78861d00-d846-4b7e-952c-f98c40f162d8	2	9	Nature-photo.jpeg	722,890	image/jpeg

Рисунок 4.7 - Метадані файлів на AuthServer

Тепер завантажений файл відображається у списку файлів адміністратора з можливостю завантаження, видалення та публікації (рис. 4.8). Файл є приватним та не опублікованим.

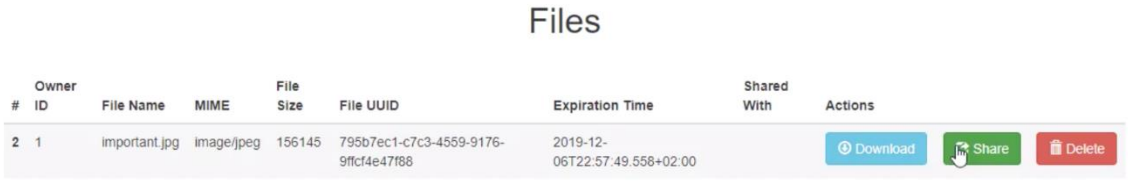


Рисунок 4.8 - Список завантажених файлів

Після завантаження файлу на сервер його можна опублікувати використовуючи наявні дозволи: всі користувачі, вказаний список користувачів, користувачі з певною роллю (Рис. 4.9).

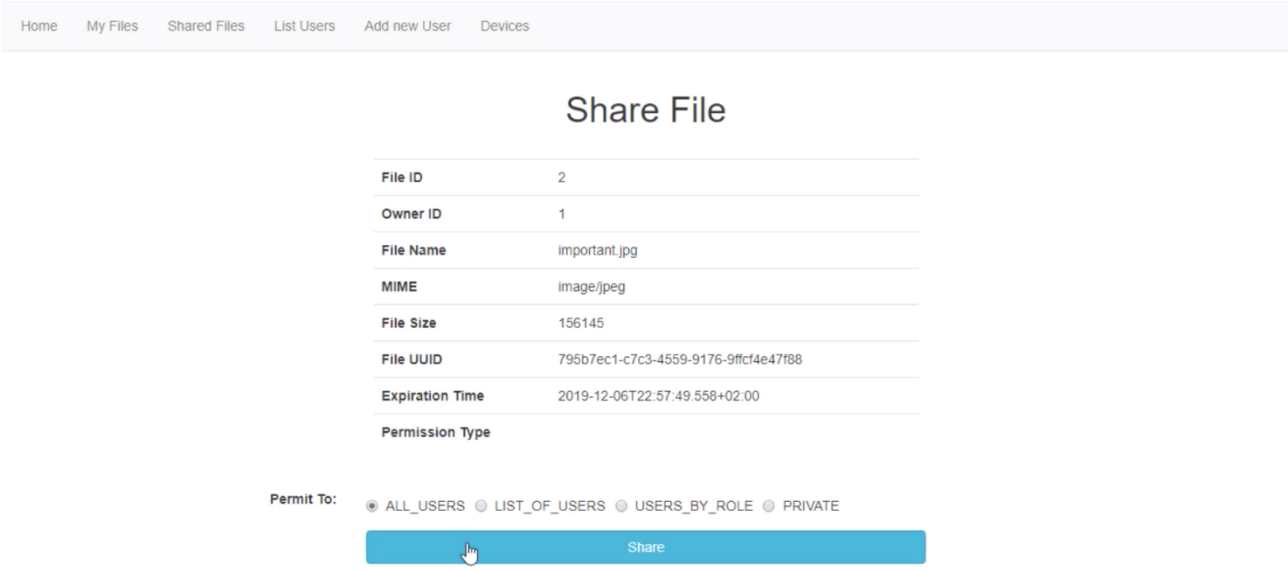


Рисунок 4.9 - Публікація файлу

Коли файл публікується, усі користувачі яким файл є доступним, отримуються повідомлення на пошту про отримання нового доступу до файлу (Рис. 4.10).

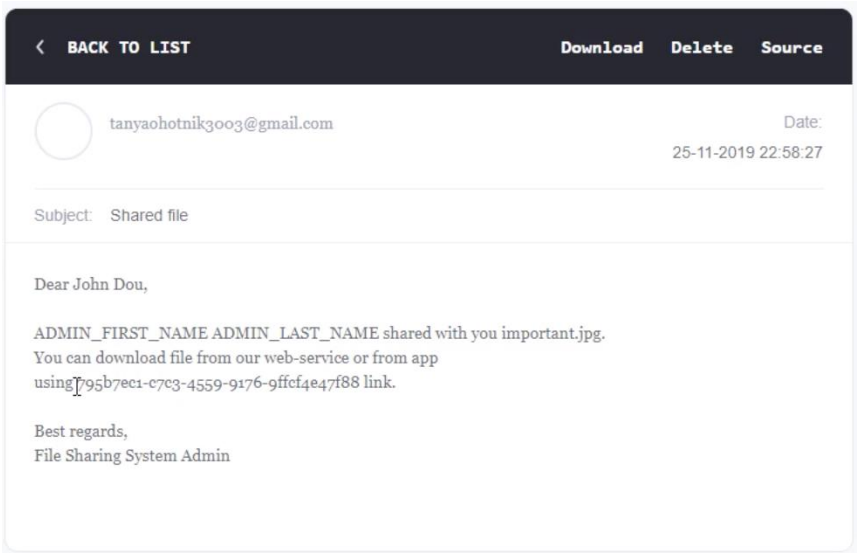


Рисунок 4.10 - Повідомлення про отримання доступів до файлу

4.2 Опис мобільного додатку

Для завантаження опублікованого файлу з мобільного додатку користувач має пройти процес активації (перший доступ до системи), використовуючи логін та одноразовий пароль отримані на електрону пошту у процесі реєстрації (Рис. 4.3). На рисунку 4.11 наведено приклад авторизації.

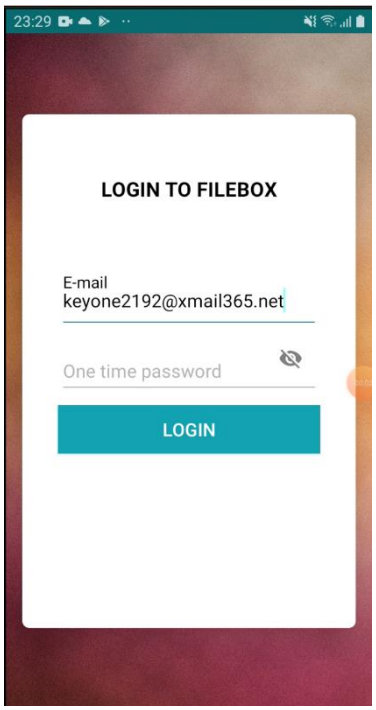


Рисунок 4.11 - Вікно авторизації мобільного додатку

Перший вхід у системи завершується встановленням пароля на додаток (рис. 4.12), який має бути введено кожний раз при відкритті додатку. Паролі мобільного додатку жодним чином не пов’язані з паролем входу у веб додаток.

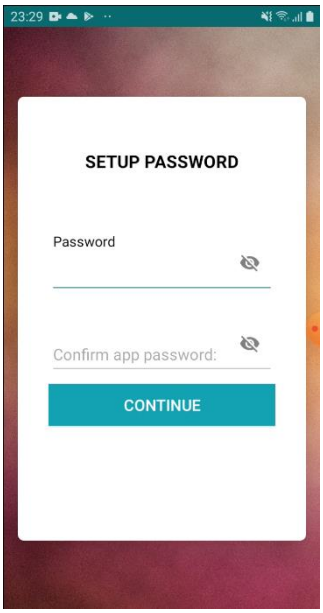


Рисунок 4.12 - Завершення процесу активації - встановлення пароля

Після завершення процесу активації, користувач потрапляє на головний екран додатку, що відображає список завантажених файлів, як показує рисунок 4.13, список файлів для нового користувача є пустим.

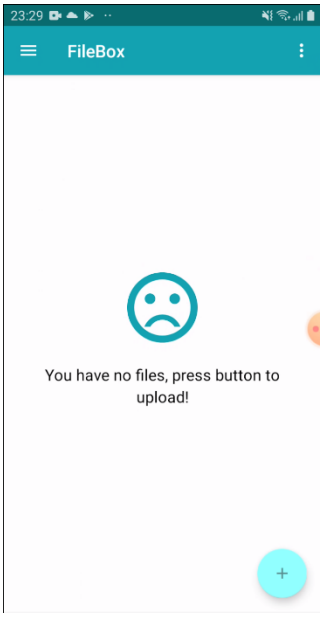


Рисунок 4.13 - Головний екран додатку

Для використання додатку у наступні рази користувач використовує пароль для додатку, я не одноразовий виданий пароль (рис. 4.14).

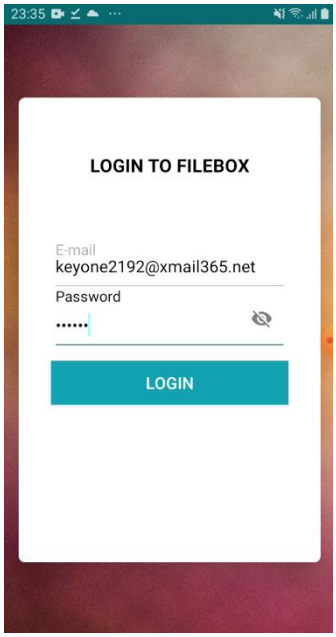


Рисунок 4.14 - Авторизація на активованому додатку

На рисунку 4.15 представлено екран для завантаження файлу на сервер. Користувач, як і при роботі з веб додатком, повинен обрати файл та дату закінчення життя файлу.

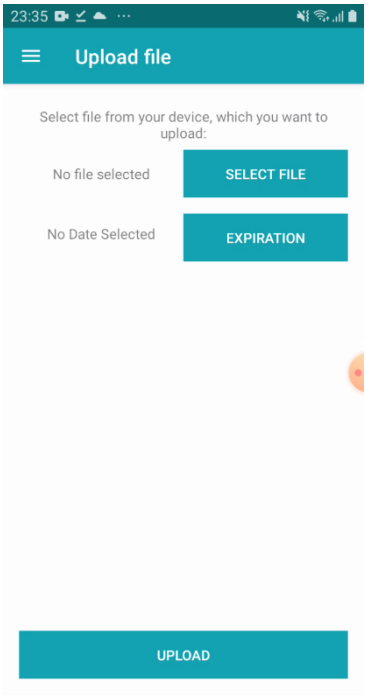


Рисунок 4.15 - Завантаження файлу на сервер

Процес завантаження файлу відбувається у окремому потоці сервісу додатку (Рис. 4.16).

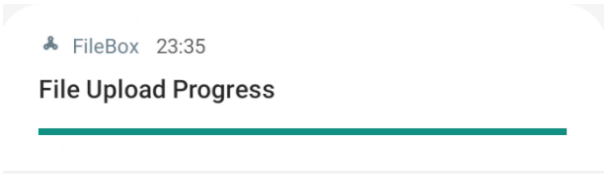


Рисунок 4.16 - Індикатор процесу завантаження файлу

Після успішного завантаження файлу на сервер, він відображається у списку завантажених файлів на головному екрані додатку (рис. 4.17).

Рисунок 4.17 - Список завантажених файлів

Завантажений, але не опублікований файл вважається приватним. На рисунку 4.18 представлено екран публікації файлу з вибором типу дозволів.

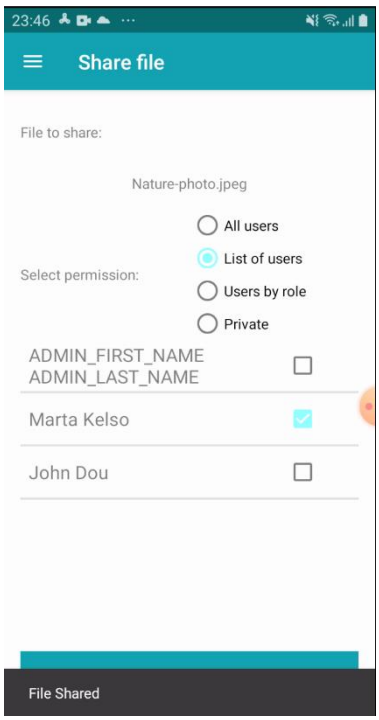


Рисунок 4.18 - Публікація файлу

Користувач з мобільного додатку може завантажити опублікований файл за його посиланням (рис. 4.19), отриманим у листі зі сповіщенням.

Рисунок 4.19 - Завантаження файлу з серверу за посиланням

Коли сервіс завантаження файлу закінчив свою роботу, у списку фалів пристрою (рис. 4.20) можна побачити завантажений адміністратором файл important.jpg.

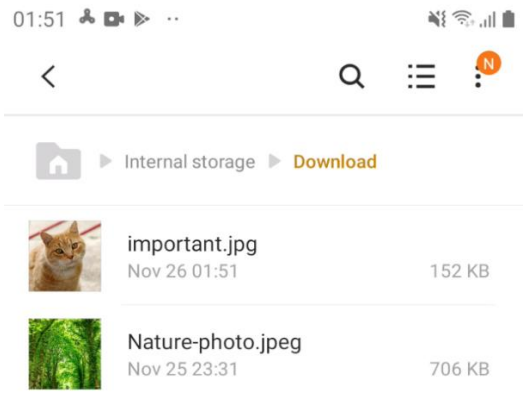


Рисунок 4.20 - Список файлів на пристрої

4.3 Обґрунтування ефективності архітектурного рішення

Система, побудована на основі розробленої архітектури має наступні характеристики:

- селективне шифрування важливих даних;
- огортання ключа;
- наскрізне шифрування;
- авторизація серверів та клієнтів на основі сертифікатів та підписів запитів;
- використання одноразового стійкого пароля для першого доступу до системи;
- відокремлення метаданих від файлів.

Архітектура масштабована - її можливо розширювати для реалізації важливих функцій. Завдяки наявності DistributionServer можливо встановити постійний канал між сервером та мобільним додатком за допомогою веб-сокетів. Цей канал можна використати задля передачі на мобільний додаток даних та подій від AuthServer (наприклад, видалення даних додатку). Також цей канал

можна використовувати для передачі різних політик системи, наприклад, політик паролей.

У розділі 1 магістерської дисертації було розглянуто існуючі системи та виділено пріоритезовані критерії оцінки систем для обміну файлами. У таблиці 4.1 наведено порівняння розробленої системи з існуючими.

Таблиця 4.1 – Порівняння функцій систем

	Box Business	Dropbox Business	Citrix ShareFile	Egnyte Connect	IBM Connections	Розроблена система
Шифрування файлів	AES 256	AES 256	AES 256	AES 256	AES 128	AES 256
Шифрування даних користувачів (не файлів)	-	-	-	+	-	+
Відокремлення сховища даних та ключів	-	+	+	-	+	+
Заборона прямого доступ до сховища даних	+	-	+-	+	-	+
Шиврування метаданих	-	-	-	-	-	+
End-to-end шифрування	+	+	-	+	-	+
Wrapping key	+	-	-	-	-	+
Очищення завантажених файлів	-	-	+	-	-	+

До переваг архітектури можна віднести селективне шифрування та огортання ключів, що забезпечують конфіденційність даних. Наскрізне шифрування, кероване сервером, а також відокремлення файлів від метаданих, забезпечують конфіденційність та цілісність файлів, шо передаються. Алгоритм першого доступу до системи реалізує доступність даних.

До недоліків системи можна віднести її продуктивність, оскільки витрачається більше часу на обробку файлу та на роботу з зашифрованими статичними даними користувачів.

4.4 Висновки до розділу

В даному розділі було описано виконане програмне забезпечення побудоване на представленій архітектурі безпечного обміну файлами між мобільними пристроями за межами корпоративної мережі. За результатами розробки було отримано три серверні компоненти для управління користувачами, зберігання файлів та управлінням доступу до системи, мобільний додаток та веб додаток.

Обґрунтовано ефективність розробленої системи побудованої на спроектованій архітектурі.

РОЗДІЛ 5 РОЗРОБКА СТАРТАП-ПРОЕКТУ

5.1 Опис ідеї проекту

Проаналізуємо зміст ідеї, її можливі напрямки застосування, чим запропонована ідея відрізняється від існуючих аналогів, а також основні вигоди, які може отримати користувач продукту. Результати аналізу представлені у таблиці 5.1.

Таблиця 5.1 - Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Створити платформу для безпечного обміну файлами у рамках корпоративної мережі	Обмін файлами	Безпечний обмін даними через зашифрований тунель та розмежування доступу до цих файлів
	Віддалене управління контейнером на мобільному пристрої користувача	Управління даними на пристрої користувача для їх убезпечення та контролю IT адміністратором
	Контроль доступу до корпоративної мережі	Захищеність корпоративної мережі при використанні BYOD підходу. Неможливо отримати неконтрольований доступ до мережі.
	Збереження файлів на сервері компанії	Повний контроль сервером та доступом до нього

На ринку існують аналоги подібних систем, проти вони не відповідають повному спектру поставлених вимог. У таблиці 5.2 наведено характеристики ідеї проекту.

Таблиця 5.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Cirtix	Box	Egnyte			
1.	Статичне шифруваня	+	-	-	+			+
2.	Розподілене збережння файлів	+	+	-	-			+
3.	Кросплатформеність	+-	+	+-	+	+		
4	Прямий доступ до сховища	-	+	-	-			+
5	Обмеження памяті	+	-	+	+		+	
6	Збереження відкритих метаданих	-	+	+	+			+
7	Автоматичне видалення файлів	+	+	-	-			+
8	Віддалено управління пристроєм	+	+	+	+		+	

Основними відмінностями є спосіб зберігання файлів, метаданих, та віддалене управління пристроєм.

5.2 Технологічний аудит ідеї проекту

У підрозділі проведено аудит технологій, за допомогою яких можна реалізувати проект. Результат наведено у таблиці 5.3

Таблиця 5.3 - Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Завантаження файлів до єдиного сховища	Серверний компонент. Розташовується за фаерволом у корпоративній мережі. Управляє процесом збереження, шифрування файлів, відсилає команди до контейнеру.	Необхідні: Java, Spring, Hibernate, PostgreSQL, MongoDB, OpenVPN	Доступні, безкоштовно
2	Контроль доступу до файлів			
3	Управління контейнером			
4	Відображення файлів на пристрої	Клієнтський компонент. Розробка Android додатку для комунакації з серверним компонентом.	Необіхдні: Java, Android SDK, Retrofit, FCM, SQLite, OpenVPN	Доступні, безкоштовно
5	Збереження даних клієнта у безпечному контейнері			
Обрана технологія реалізації ідеї проекту: Java, Spring, Hibernate, PostgreSQL, Android SDK, Retrofit, FCM, SQLite. Обрані технології є фактично набором інструментів та фреймворків для реалізації мети додатку, не потребують доробки.				

5.3 Аналіз ринкових можливостей запуску стартап-проекту

Для визначення ринкових можливостей було, які можна використати під час ринкового впровадження проекту, проведено аналіз попиту (табл. 5.4).

Таблиця 5.4 - Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	5
2	Загальний обсяг продаж, грн/ум.од	7 800 000$
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Немає
5	Специфічні вимоги до стандартизації та сертифікації	OWASP for Mobile, ISO
6	Середня норма рентабельності в галузі (або по ринку), %	152%

Світовий ринок програмного забезпечення класу EFSS для убезпечення даних розглядається як один з найбільш зростаючих. Тому ринок є привабливим. Крім того, на ньому відсутні будь-яка обмеження щодо стандартизації або сертифікації, що робить вхід на ринок легким. У таблиці 5.5 наведено характеристику потенційних клієнтів стартап проекту.

Таблиця 5.5 - Характеристика потенційних клієнтів стартап-проекту

Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
Безпечний обмін даними та доступ до них	Компанії, що мають потребу у доступному сховищі даних, проте хочуть уникнути використання хмарних рішень та зберігати сервер локально під своїм наглядом	- Наявність фінансового забезпечення - Розподілене збереження зашифрованих даних - Великий документообіг	- Стабільність - Постійна підтримка - Впровадження оновлень
Контроль доступу до внутрішньої мережі компанії та її ресурсів	Компанії, що не можуть собі дозволити повноцінні та повномасштабні MDM рішення, проте прагнуть убезпечити доступ до корпоративної мережі та інформації	- Слідування BYOD концепту - Контроль контейнеру	

Оцінимо фактори загроз, які потенційно можуть завадити реалізації проекту. Оцінку наведено у таблиці 5.6.

Таблиця 5.6 - Фактори загроз

Фактор	Зміст загрози	Можлива реакція компанії
Поява конкурентів	Можлива поява конкурентів, які спроможуться створити більш якісний продукт або рошрити вже популярний та існуючий сервіс. Можлива поява більш дешевих продуктів	Зменшення ціни з підвищенням якості при цьому, розробка удосконалень, розширення асортименту (додавання нових можливостей, нового функціоналу та/або додання можливостей розрахунку нових параметрів)
Значне здороження утримання дата-центрів	Можливе значне збільшення ціни на утримання власних серверів для збереження файлів	Удосконалення продукту з перспективою додавання підтримки хмарного збереження даних
Збільшення переваги хмарних сервісів	Утримання відокремленого та захищеного віртуального простору коштує приблизно однаково з утриманням власного серверу, тому може стати альтернативним підходом до розподіленого збереження даних	Удосконалення продукту з метою задоволення потреб замовників - додавання підтримки хмарного збереження даних
Відсутність репутації компанії	Можлива ситуація, коли потенційні компанії замовники відмовляться від співробітництва та придбання продукту через непопулярність компанії на ринку рішень інформаційної безпеки	Досконалий вибір перших клієнтів та замовників, укладання довгострокових контрактів, реклама продукту

Продовження таблиці 5.6

Впровадження сертифікації сервісів	З розвитком підходів та способів захисту інформації можливе впровадження певного типу перевірки та сертифікації рішень, що відповідають за убезпечення даних	Реалізація інфраструктури за найновішими канонами безпеки. Проходження сертифікації

У таблиці 5.7 наведено фактори можливості системи.

Таблиця 5.7 - Фактори можливостей

Фактор	Зміст можливості	Можлива реакція компанії
Невелика кількість конкурентів	На ринку на сьогоднішній день існує невелика кількість конкурентів, їх програмні продукти в переважній більшості вузько спеціалізовані.	Розповсюджувати створений продукт, розвивати його можливості.
Можливість побудови власної репутації	Новий «гравець» на ринку має всі можливості для побудови власної репутації з «чистого листка»	Пошук замовників, можливих покупців створеного продукту, розширення бази замовників. Зарекомендувати себе, як надійну компанію. Співпраця на вигідних умовах.

Продовження таблиці 5.7

Демонстрація ефективності підходу	Продемонструвати на прикладі мережі клієнта убезпечення даних та контроль доступу до системи з метою залучення нових клієнтів	Продемонструвати на прикладі мережі клієнта убезпечення даних та контроль доступу до системи. Провести тестування на проникнення.
Зростання попиту на системи управління девайсом та доступом до мережі	Збільшення клієнтської бази	Привернення нових клієнтів та реклама

Основними загрозами, у майбутньому, є надання переваги хмарним обчисленням та вихід на ринок серйозних конкурентів, що можуть зменшити кількість компаній-клієнтів. На майбутні релізи необхідно додати підтримку хмарних сховищ.

У таблиці 5.8 наведено проведений аналіз пропозиції: визначаються загальні риси конкуренції на ринку.

Таблиця 5.8 - Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції: монополістична	На ринку присутні багато профільних організацій, що розробляють системи для великих компаній, проте жоден з них не займає відчутну частку ринку	Немає необхідності боротися за користувачів з монополістами, тому цей фактор не є значущим

Продовження таблиці 5.8

2. За рівнем конкурентної боротьби: національний	Локальна конкуренція на вітчизняному ринку	Для ринку кожної країни необхідно буде адаптуватись, тому оптимально буде розпочинати роботу тільки в одній країні, а вже потім розповсюджуватись на інші ринки.
3. За галузевою ознакою: міжгалузева	Продукт націлений на компанії середнього та малого бізнесів	Продукт націлений на компанії, що мають на меті убезпезпечити свої мережі та дані. Не важливо, у якій сфері вони діють.
4. Конкуренція за видами товарів: товарно-видова	Отримати користувача, який не має на меті убезпечення даних внутрішніми сервісам и (не онлайн) дуже важко	Необхідно правильно презентувати свій сервіс, щоб користувачі знали про всі його переваги
5. За характером конкурентних переваг: нецінова	В першу чергу користувачів цікавить якість сервісу та убезпечення даних	Необхідно створити зручний сервіс та правильно подати його компаніям-користувачам
6. За інтенсивністю: марочна	Можуть з’являтись конкуренти	Потрібно рекламувати кращий функціонал створеного продукту, встановлювати конкурентоспроможні ціни, та доводити свою надійність

На основі аналізу конкуренції, проведеного у таблиці 5.8, а також із урахуванням характеристик ідеї проекту (табл. 5.2), вимог споживачів до товару (табл. 5.5) та факторів маркетингового середовища (табл. 5.6-5.7) визначається та обґрунтовується перелік факторів конкурентоспроможності. Аналіз представлено у таблиці 5.9, обґрунтування факторів у таблиці 5.10.

Таблиця 5.9 - Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Клієнти	Товари-замінники
	Розробники аналогічних систем	Кращі продукти, серйозніші продукти, оптимальні ціни	Мають найбільше значення. Важлива постійна співпраця з підприємствами - замовниками	Відсутні. Є лише конкуренти аналогічних розробок
Висновки:	Інтенсивність конкурентної боротьби з боку прямих конкурентів незначна	Наявні усі можливості входу на ринок.	Необхідність клієнтської-бази, тому важливо знаходити можливості приваблення споживачів до власного продукту	Немає обмежень

Таблиця 5.10 - Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
	Комбінація рішення для віддаленого доступу з додатком для обміну файлами, контроль доступу до приватної мережі компанії	Збереження даних контролюється не лише мобільним додатком, що може бути підмінено і скомпрометовано, а й серверним компонентом, що регулює політики безпеки і управляє передачею та збереженням файлів на стороні серверу
	Розподілене збереження інформації про користувачів, їх пристрої, ключі, паролі, та, безпосередньо, файли	Збереження приватної інформації користувачів, її селективне шифрування, відокремлене збереження файлів та закритий доступ до них збільшує убезпеченість даних

На основі проведеного аналізу можемо зробити висновок, що на ринку наявно багато розрізнених сервісів, тому є необхідність у чіткому позиціонуванні та акценті на перевагах сласного сервісу. Оскільки бар'єри для входу на ринок практично відсутні, то є необхідність у отриманні лояльної аудиторії, яка отримає всі необхідні їй функції та не буде шукати інші сервіси, які потенційно можуть з'явитись на ринку.

На основі проведеного аналізу ринку та основних його учасників проведемо обґрунтування факторів конкурентоспроможності.

Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу: матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) (табл. 5.11) на основі виділених ринкових загроз та можливостей, сильних і слабких сторін.

Таблиця 5.11 - SWOT- аналіз стартап-проекту

Сильні сторони: Комбінація рішення для віддаленого доступу з додатком для обміну файлами, контроль доступу до приватної мережі компанії. Розподілене збереження інформації про користувачів, їх пристрої, ключі, паролі, та, безпосередньо, файли.	Слабкі сторони: вихід на ринок інформаційної безпеки невідомою компанією, необхідність додаткових серверів у рамках компанії.
Можливості: покращення сервісу, його функціоналу, впровадження підтримки хмарних сховищ. Розробка інших клієнтських додатків, наприклад для обміну поштою.	Загрози: поява конкурентів, які спроможуться створити більш якісний продукт або розширити вже існуючий популярний сервіс. Поява більш дешевих продуктів. Можливе значне збільшення ціни на утримання власних серверів для збереження файлів.

Сильними сторонами проекту є унікальність функцій, що пропонуються. Слабкими сторонами є високі вимоги до програмного забезпечення: наявність

власних серверів. Вагомим також є вихід на ринок інформаційної безпеки невідомою компанією.

На основі SWOT-аналізу розроблено альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок (див. табл. 5.9).

Визначені альтернативи аналізуються з точки зору строків та ймовірності отримання ресурсів (табл. 5.12).

Таблиця 5.12 - Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Створення мінімального продукту з набором найголовніших функцій та швидкий вихід на ринок	Висока	Від чотирьох до шести місяців
2	Створення потужного серверного та клієнтського компонентів та вихід на ринок тільки після реалізації усіх основних компонентів системи	Середня	Від 12 до 14 місяці

Доступні дві стратегії – створення мінімального продукту для виходу на ринок для оцінки реакції користувачів, та створення продукту, що буде містити всі бажані функції, але з відкладеним виходом на ринок. Обраною альтернативою є перша, адже є важливим отримання зворотного зв’язку від користувачів та компаній перед вкладанням ресурсів у функції які є менш бажаними. Також збільшення термінів розробки збільшує ймовірність появи конкурентів зі схожими функціями.

5.4 Розроблення ринкової стратегії проекту

Для створення ринкової стратегії проаналізуємо цільові групи потенційних користувачів (табл. 5.13).

Таблиця 5.13 - Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Компанії малого бізнесу	Низька	Низька	Низька	Середня
2	Компанії середнього бізнесу	Висока	Високий	Низька	Низька
3	Компанії великого бізнесу	Середня	Низький	Висока	Висока
Які цільові групи обрано: в першу чергу необхідно орієнтуватися на компанії середнього бізнесу, оскільки вони не готові витрачати великі кошти на придбання та підтримку великомасштабних MDM рішень, проте все одно потребують захисту внутрішньої мережі та безпечного документообігу з можливістю віддаленого доступу.					

На основі отриманої інформації та альтернативи ринкового впровадження стартап-проекту розробимо базову стратегію розвитку (табл. 5.14).

Таблиця 5.14 - Визначення базової стратегії розвитку

Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Вихід на профільний ринок з сервісом обміну файлами, контролем доступу до приватної мережі компанії	Масовий маркетинг серед компаній середнього бізнесу	Комбінація рішення для віддаленого доступу з додатком для обміну файлами, контроль доступу до приватної мережі компанії. Розподілене збереження інформації про користувачів, їх пристрої, ключі, паролі, та, безпосередньо, файли.	Стратегія диференціації

Базовою стратегією розвитку є вихід на масовий ринок з потужним масовим маркетингом. Основним акцентом в маркетинговій стратегії є виділення ключових рис сервісу, а саме можливість зручно отримувати тематичні статті.

Наступним кроком є вибір стратегії конкурентної поведінки. На основі базової стратегії розвитку розробимо стратегію конкурентної поведінки (табл. 5.15).

Таблиця 5.15 - Визначення базової стратегії конкурентної поведінки

Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
Ні. Проте онлайн сервіси обміну даними не є безпечними для бізнес цілей, а існуючи EFSS рішення є дуже громіздкими та важкими у підтримці.	Компанія буде пропонувати своє рішення на конференціях, зборах та презентація для привернення компаній, що ще не використовують EFSS рішень, поряд із тими, які вже використовують їх.	Компанія не буде копіювати конкурентів	Зайняття конкурентної ніші

Даний стартап проект не є першопрохідцем на ринку, а тому йому доведеться агітувати компанії які вже використовують схожі рішення для обміну файлами (наприклад онлайн сервіси, або EFSS системи) та пояснювати переваги свого рішення.

Таблиця 5.16 - Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
Безпечний обмін даними між робітниками	Стратегія диференціації	Безпечний обмін даними через зашифрований тунель та розмежування доступу до цих файлів	Цілісність, доступність, конфіденційність даних

Продовження таблиці 5.16

Управлінням додатком користувача	Стратегія диференціації	Управління даними мобільного додатку користувача для їх убезпечення. Контроль IT адміністратором - посилення політик, видалення даних, блокування доступу	Зручність, зрозумілість, актуальність
Захищеність корпоративної мережі	Стратегія диференціації	Захищеність корпоративної мережі при використанні BYOD підходу. Контроль доступу до мережі.	Безпечність, зрозумілість
Управління сервером	Стратегія диференціації	Повний контроль сервером та доступом до нього	Зручність, захищеність

Отже основою ринкової стратегії буде отримання користувачів (компаній), що використовують незахищені онлайн рішення та додатки для обміну даними. Обраною стратегією буде стратегія диференціації з акцентом на унікальності пропонованих функцій та безпечності рішення.

5.5 Розроблення маркетингової програми стартап-проекту

На основі проаналізованих ознак конкурентоспроможності товару розробимо ключові переваги потенційного товару (табл. 5.17).

Таблиця 5.17 - Визначення ключових переваг концепції потенційного товару

Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
Обмін даними за межами приватної мережі компанії	Розподілене збереження даних: файли та приватна інформація користувачів зберігається на різних серверах, метадані зберігаються окремо від файлів.	Зазвичай дані зберігаються або на одному сервері, або у хмарному сховищі (DropBox). Таким чином компанія має довіряти хмарному сховищу, а не своїм ресурсам.
Контроль доступу ззовні	Попередня реєстрація користувачів у системі що унеможливлює отримання доступу порушнику	Схожий підхід використовується у всіх компаніях, проте у випадку нашого додатку, використовуються логін та ключ для активації додатку, надалі додаток генерує пароль, що відповідає вимогам безпеки, а користувач лише встановлює пароль додатку.
Управління мобільним пристроєм	Можливість контролювати стан даних додатку через адмін панель адміністративного серверу	Схожі системи існують (MDM), проте вони зазвичай дуже громіздкі і для сховища даних підключаються сторонні рішення.

Надалі розробляється трирівнева маркетингова модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання (табл. 5.18).

Таблиця 5.18 - Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
I. Товар за задумом	Віддалений контроль мобільного додатку, управління доступом, політиками безпеки та файлами підприємства.
II. Товар у реальному виконанні	Властивості/характеристики
	1. EFSS сервер у мережі компанії, що працює за налаштованим фаерволом. Підключення відбувається через тунель (VPN). Має можливість налаштування політик доступу до файлів, завантаження файлів, управління користувачами. 2. Мобільний клієнт для комунікації з сервером. Мобільний додаток використовує публічний сервер для доступу у підмережу. 3. Проміжний сервер для з`єднання клієнту з сервером 4. Сервер для збереження даних
	Марка: назва організації-розробника + назва товару
III. Товар із підкріпленням	Підтримка хмарного сховища даних
За рахунок чого потенційний товар буде захищено від копіювання: протокол комунікації між клієнтом, сервером та проміжним сервером; алгоритм першого доступу до системи з мобільного додатку.	

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар (остаточне визначення ціни відбувається під час фінансово-економічного аналізу проекту), яке передбачає аналіз ціни на товари-аналоги або товари замінники, а також аналіз рівня доходів цільової групи споживачів (табл. 5.19). Аналіз проводиться експертним методом.

Таблиця 5.19 - Визначення меж встановлення ціни

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
Ціна на онлайн сервіси може бути від 0 до 40$ на місяць в залежності від об’єму даних	Масштабні EFSS платформи в залежності від ліцензії та кількості використаної пам’яті 400 -900 тис $ на рік	Середній бізнес, 10-50 млн $	100-200 тис $ на рік

Останньою складової маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (табл. 5.20).

Таблиця 5.20 - Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
Використання онлайн сервісів та пошти для обміну даними у рамках компанії	Переважно інтернет та внутрішні мережі компаній	Поєднання системи розподіленого файлообміну з віддаленим доступом до мережі компанії	Донести інформацію про можливість віддаленого управління мобільного додатку працівника та розподіленого збереження даних	Розповісти про можливість обміну даними використовуючи сервери всередині компанії

5.6 Розробка фінансового плану

У таблиці 5.21 наведено вихідні дані для визначення доходності проекту.

Таблиця 5.21 - Вихідні дані для визначення доходності проекту

№п/п	Показники	Од. вим.	Значення
1	Витрати на обладнання	грн	270000
2	Монтаж, навчання персоналу	грн	81000
3	Термін роботи власного обладнання після вводу	років	5
4	Прогнозований обсяг продажів продукції в рік	грн	6480729
5	Валютний депозит	%	15,0
6	Фактори ризику	%	3,0
7	Інфляція на валютному ринку	%	13

Для реалізації проекту необхідно мати розробників, маркетологів, та менеджера проекту. Дані заробітної платні кожного спеціаліста за місяць разом з утриманнями наведені у таблиці 5.22.

Таблиця 5.22 - Таблиця заробітної плати робітників в місяць

Посада	Кількість	Заробітня плата, грн	До видачі працівнику, грн	Утримання за рахунок працівника, грн	Утримання за рахунок підприємства, грн
Розробник	2	37800	30429	7371	8316
Спеціаліст з людських ресурсів	1	13500	10867	2632	2970
Маркетолог	4	40000	32200	7800	8800
Менеджер проекту	1	32400	26082	6318	7128

Розрахуємо тепер поточні витрати на здійснення проекту. Результати наведено у таблиці 5.23.

Таблиця 5.23 - Поточні витрати на здійснення проекту

Витрати	1 рік	2 рік	3 рік	4 рік	5 рік	Всього
1. Зарплата	4285200	4285200	4285200	4285200	4285200	21426000
2.Нарахування	743160	743160	743160	743160	743160	3715800
4.Амортизація	48644	39880	32695	26804	21975	169998
5. Інші	11700	13000	13000	13000	13000	63700
6.Умовно-постійні	30000	30000	30000	30000	30000	150000
Всього:	5118704	5111240	5104055	5098164	5093335	25525498

Витрати на придбання обладнання здійснюються в 0-й рік (з моменту придбання обладнання починається відрахунок терміну виконання проекту); витрати на монтаж обладнання та навчання персоналу - в 1-й рік .

0-й рік - 1245000;

1-й рік - 100000.

Визначення обсягу грошових потоків (чистого доходу + амортизації, враховуючи оподаткування прибутку за ставкою 18%), грн.:

1й рік = 48644 (тільки амортизація, оскільки продукція не вироблялась);

2й рік = 6480729-5111240-246508+39880=1162861;

3й рік=(6480729-5104055-247801)+ 32695=1161568;

4й рік=(6480729-5098164-248862)+ 26804= 1160507;

5й рік=(6480729-5093335-249731)+ 21975=1159638;

Визначення норми дисконтування проекту (d):

$$d = 0,15 + 0,03 + 0,13 = 0,31$$

У таблиці 5.24 наведено розрахунок дисконтований дохід проекту.

Таблиця 5.24 - Розрахунок чистого дисконтованого доходу проекту

Роки	D	K	$\frac{1}{(1+d)^t}$	$\frac{D}{(1+d)^t}$	$\frac{K}{(1+d)^t}$	ЧДД	ЧПВ
0	0	270000	1	0	270000	-270000	-270000
1	48644	81000	0.7633	37132.8	61832.1	-32356	-302356
2	1162861	0	0.5827	677618	0	1162861	860505
3	1161568	0	0.4448	516691	0	1161568	2022073
4	1160507	0	0.3395	394060	0	1160507	3182580
5	1159638	0	0.2592	300584	0	1159638	4342218
Всього:	4693218	351000	x	1926086	331832	4342218	x

На підставі попередніх розрахунків ЧДД та ЧПВ визначимо термін окупності:

Ток = 1 + 302356/860505= 1,3513 (року).

Термін окупності проекту дорівнює 1.35 рокам.

Визначимо індекс доходності та середньорічну рентабельність проекту:

ІД = 1926086/331832 = 5.8044

Тоді середньорічна рентабельність проекту (R) буде:

R = ІД / n x 100% = 580.44 / 5 = 116%.

Таким чином, даний проект є високорентабельним. Оскільки в перший рік його реалізації завершувались наукові дослідження, освоювалось обладнання та навчався персонал роботі на обладнанні, то грошові потоки в перший рік забезпечувались у вигляді амортизації обладнання. Термін окупності даного проекту визначається періодом між першим та другим роками (коли ЧПВ більше 0, то проект окупається).

5.7 Висновки до розділу

Основною ідеєю проекту є створення платформи для безпечного обміну файлами між мобільними пристроями за межами корпоративної мережі.

Основними технологіями, що будуть використані для створення проекту є бібліотеки: Java, Spring, Hibernate, PostgreSQL, Android SDK, VPNht, Retrofit,

MongoDB, SQLite. Обрані технології є фактично набором інструментів та фреймворків для реалізації мети додатку та не потребують доробки.

Світовий ринок програмного забезпечення для убезпечення даних розглядається як один з найбільш зростаючих, саме тому ринок є привабливим. Крім того, на ньому відсутні будь-яка обмеження щодо стандартизації або сертифікації, що робить вхід на ринок дуже легким.

Головними факторами конкурентоспроможності є збереження приватної інформації користувачів та безпосередньо файлів відокремлено на серверах компанії, збереження даних контролюється не лише мобільним додатком, що може бути підмінено і скомпрометовано, а й серверним компонентом, що регулює політики безпеки і управляє передачею та збереженням файлів на стороні серверу.

Потенційною аудиторією проекту є компанії середнього бізнесу. Цей сегмент аудиторії має розуміння необхідності збереження корпоративних даних. Після освоєння цього сегменту можна розглянути можливість просування продукту для великих компаній. Для подальшого розширення аудиторії можна розглянути вихід на міжнародний ринок.

Основною стратегією позиціонування є стратегія диференціації з акцентом на основні відмінності від конкурентів. Найбільш оптимальним є прямий контакт з потенційними клієнтами з метою залучення його до нашої платформи.

Монетизація проекту буде відбуватись за рахунок платної ліцензії на програмний додаток.

У підсумку даний проект є перспективним. Ринок є досить великим та постійно зростає, на ньому немає відчутних вхідних бар’єрів. Хоча користувачі на ринку вже поділені між існуючими сервісами, проте за умови надання унікального (для залучення нових користувачів) та зручного (для їх утримання) сервісу можу принести значну вигоду.

ВИСНОВКИ

У рамках магістерської дисертація метою була поставлена побудова архітектурного рішення, що надає безпечний віддалений доступ до корпоративної мережі мобільним клієнтам з убезпеченням передачі та збереження даних, а також реалізація програмного додатку на основі пропонованої архітектури. Мета була досягнена за рахунок виконання поставлених задач. Доведена актуальність і необхідність розробки пропонованої архітектури та системи.

У першому розділі магістерської дисертації було розглянуто і проаналізовано методи віддаленого доступу до мережі, методи обміну файлами та методи збереження файлів. Також було виконано огляд і порівняння існуючих рішень: Box Business, IBM Connections, Dropbox Buxiness, Citrix ShareFile, Egnyte Connect

У другому розділи було описано основні обрані методи криптографічного захисту, наведено загальний огляд архітектури системи, описано її компоненти. FileServer відповідає за зберігання зашифрованих файлів, AuthServer реалізує управління користувачами, метаданими файлів та доступом до них. DistributionServer відповідає за зберігання даних про AuthServer та конфігурації для доступу до нього, зберігає дані про дозволені доступи з мобільних додатків, надає користувачу конфігурацію для доступу до системи. У розділі також представлено алгоритми першого доступу до системи, передачі і шифрування даних та алгоритм комунікації між компонентами. Наведено діаграми послідовностей алгоритмів.

У третьому розділі магістерської дисертації описано реалізовані серверні компоненти та використані технології. Розглянуто публічні інтерфейси серверів, приклади конфігурації, ER-діаграмі бази даних, деталі роботи тощо. Описано вимоги до веб додатку та мобільного додатку.

Четвертий розділ присвячено демонстрації основних функцій готового продукту за скріншотами на основі пропонованої архітектури.

Продемонстровано процеси створення та видалення користувачів, розглянуто приклади збережених даних у базі даних з використання селективного шифрування. Розглянуто нотифікації, вироблені системою. Продемонстровано процеси першого доступу до системи, завантаження файлів з серверу та на сервер, а також процес публікації файлу.

П’ятий розділ описує відповідний стартап-проект. Визначено слабкі, сильні та нейтральні характеристика пропонованої платформи, проведено аналіз конкурентних продуктів, проведено SWOT-аналіз стартап проекту, проаналізовано ринкові можливості для запуску проекту,

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) How many phones are in the world [Електронний ресурс] - 2019 – Режим доступу: https://www.bankmycell.com/blog/how-many-phones-are-in-the-world

2) File Sharing and Group Information Management/ T. Whalen, E. Toms, J. Blustein. // Dalhousie University Halifax, NS, Canada – 2017. – №57. – 9 с.

3) Network Security: Private Communication in a Public World / C. Kaufman, R. Perlman, and M. Speciner. – Prentice Hall PTR, 2015. – pp. 75-90

4) Підвищення ефективності використання корпоративної мережі за концепцією BYOD / О. П. Ткаліч, Р. С. Одарченко, Є. В. Рибальченко, О. В. Марченко, Є. Ю. Шеремет, О. В. Лагодний // Проблеми створення, випробування, застосування та експлуатації складних інформаційних систем. - 2013. - Вип. 7. - С. 77-87.

5) CIS Controls [Електронний ресурс] - 2018 – Режим доступу: https://www.cisecurity.org/controls/

6) The Complete Guide to Internet Security / M. S. Merkow, J. Breithaupt. – AMACOM, 2018. – pp. 34-52

7) Guide to Enterprise Telework, Remote Access, and Bring Your Own Device (BYOD) Security / Murugiah Souppaya, Karen Scarfone//NIST Special Publication 800-46 Revision 2 - 2016р. - 53 с.

8) 10 Secure File Sharing Options and Tips / Panda Security [Електронний ресурс] - 2018 – Режим доступу: https://www.pandasecurity.com/mediacenter/panda-security/secure-file-sharing/

9) Методологія створення сховищ даних: стандарти та моделювання / Г. Асєєв // Вісник Книжкової палати. - 2009. - № 5. - С. 30-32.

10) Загрози інформаційної безпеки для користувачів сучасних мобільних пристроїв та засоби їх захисту / А. В. Платоненко // Сучасний захист інформації. - 2017. - № 1. - С. 128-132.

11) The Transport Layer Security (TLS) Protocol Version 1.2 [Електронний ресурс] - 2008 – Режим доступу: https://tools.ietf.org/html/rfc5246

12) JSON Web Token (JWT) [Електронний ресурс] - 2015 – Режим доступу: https://tools.ietf.org/html/rfc7519

13) Способи захисту каналів корпоративних мереж на базі VPN-рішень /Пархоменко І. І., Галкін В. В.// Сучасний захист інформації - №4, 2016 - С. 48-54.

14) Open VPN [Електронний ресурс] - 2018 – Режим доступу: https://openvpn.net/vpn-server-resources/

15) Guide to Enterprise Password Management (Draft)/ K.Scarfone, M.Souppaya// NIST Special Publication 800-118 (Draft) - 2015 р. - 38 с.

16) Cryptographic Right Answers [Електронний ресурс] / Latacora - 2018 – Режим доступу: https://latacora.micro.blog/2018/04/03/cryptographic-right-answers.html

17) Guide to General Server Security /K. Scarfone, W.Jansen, M.Tracy /NIST Special Publication 800-123 - 2008р - 38 с.

18) Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations/T.Polk, K.McKay, S. Chokhani//NIST Special Publication 800-52 Revision 1 - 2019 р. - 50 с.

19) Secure File Sharing Mechanism and Key Management for Mobile Cloud Computing Environment/I. Indu, P. M. Rubesh Anand and Shaicy P. Shaji // Indian Journal of Science and Technology 9(48) - 2017 - № 48.

20) A Survey on Security for Smartphone Device /Syed Farhan Alam Zaidi, Munam Ali Shah, Muhammad Kamran // International Journal of Advanced Computer Science and Applications - 7(4):206-219, 2016. - С.52-60

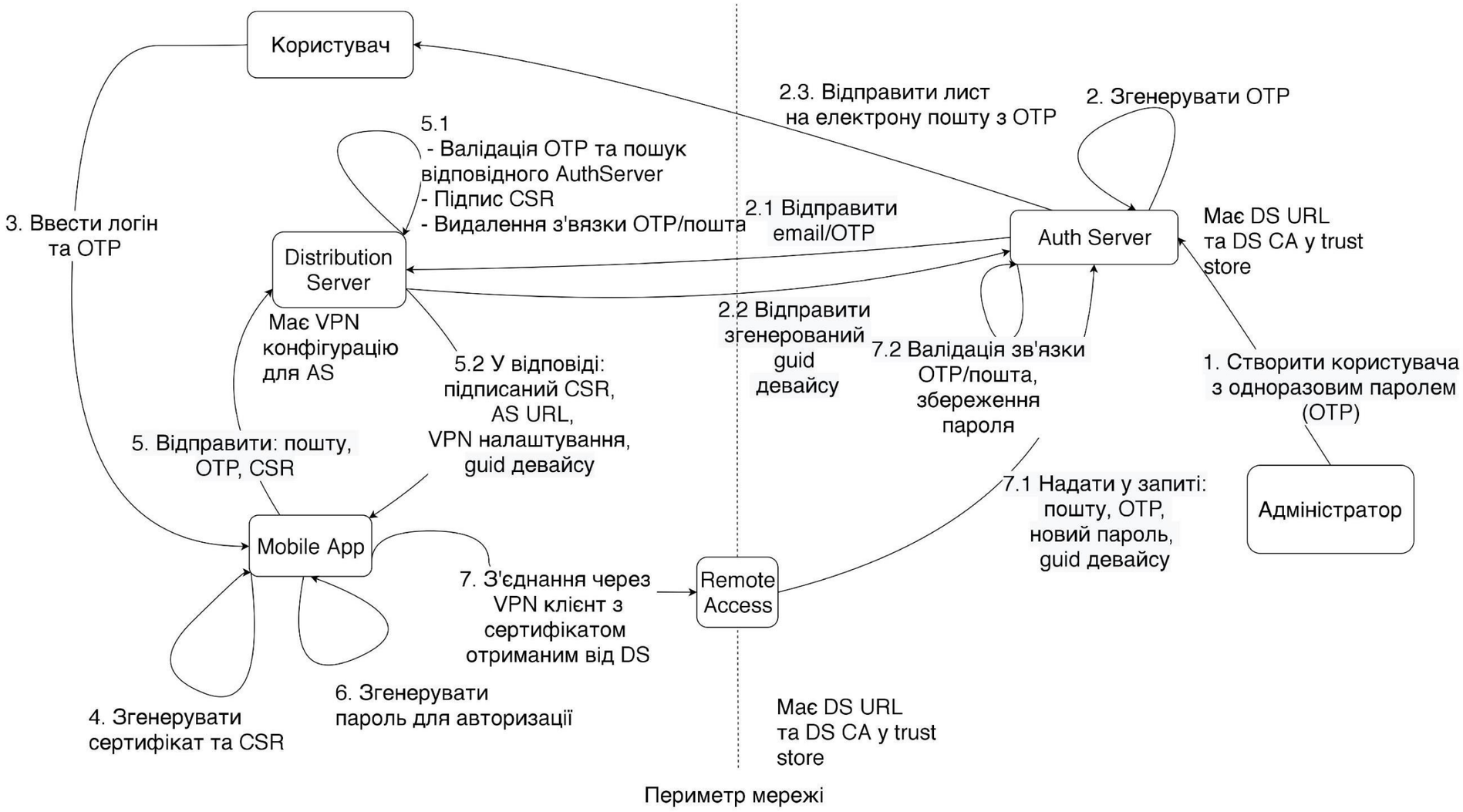
21) Gaps and Future Directions in Mobile Security Research / V. Vylegzhanina, D. C. Schmidt, J. White// Vanderbilt University - Conference: the 3rd International Workshop - 2016 р. - 4 с.

22) Загальні принципи захисту мобільних пристроїв в корпоративній мережі / М. О. Жованик // Молодий вчений. - 2015. - № 5(1). - С. 39-42.

ДОДАТОК А

ГРАФІЧНИЙ МАТЕРІАЛ

Схема роботи алгоритму першого доступу до системи



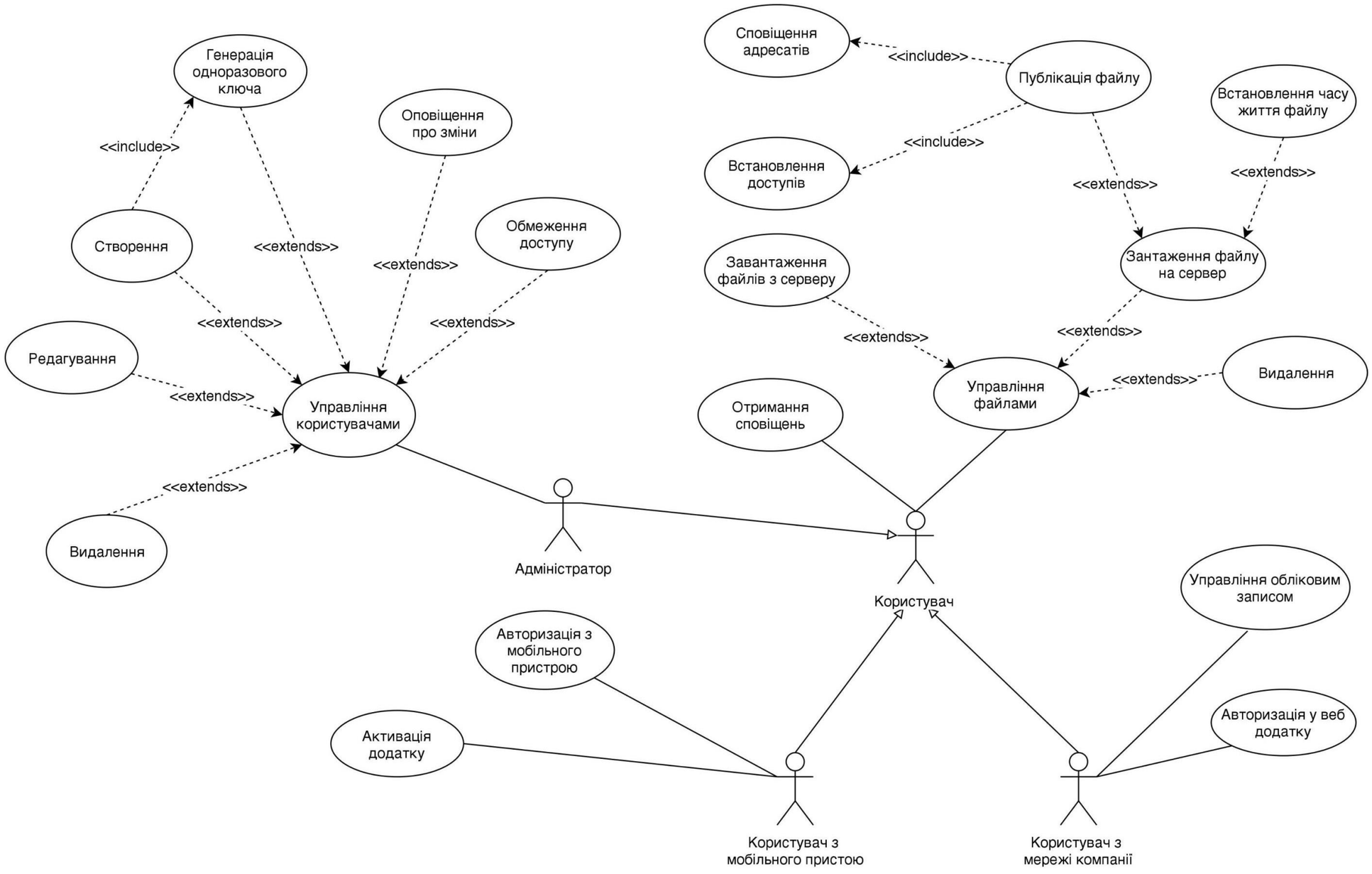
Демонстраційний плакат до магістерської дисертації

Архітектурне рішення для безпечного обміну файлами між мобільними пристроями в корпоративній мережі

Виконала студентка гр. ІП-82мп Глушкова Т.О.

Керівник Фіногенов О.Д.

Схема варіантів використання системи



Демонстраційний плакат до магістерської дисертації

Архітектурне рішення для безпечного обміну файлами
між мобільними пристроями в корпоративній мережі

Виконала студентка гр. ІП-82мп Глушкова Т.О.

Керівник Фіногенов О.Д.

ДОДАТОК Б VPN НАЛАШТУВАННЯ

Конфігурація сервера:

```
Port 1194
proto udp
dev tun
ca ca.crt
cert server.crt
key server.key  # This file should be kept secret
dh dh2048.pem
server 10.8.0.0 255.255.255.0
ifconfig-pool-persist ipp.txt
keepalive 10 120
tls-auth ta.key 0 # This file is secret
cipher AES-256-CBC
max-clients 1000
user nobody
group nogroup
persist-key
persist-tun
status openvpn-status.log
verb 3
explicit-exit-notify 1
```

Конфігурація клієнту:

```
client
dev tun
proto udp
```

```
remote 192.168.0.100 443
resolv-retry infinite
nobind
user nobody
group nogroup
persist-key
persist-tun
remote-cert-tls server
cipher AES-256-CBC
auth SHA256
key-direction 1
<tls-auth>
#
# 2048 bit OpenVPN static key
#
-----BEGIN OpenVPN Static key V1-----
476e58869aeb592a3fd0e1bab1b1d68914a30522b665e8a330a61fe833416578
ba939f41094dc4227c9a762da3e04510aa9410a71fc6c552f2db526fd64ac703
b6530619ce8ab7d5954a960d3fa299fd859cd9e67a67e5031b5f06c4124a7b54
c718408b803394c37e24073855c59d4d8ac03bcf3735acca5581ed42bf2cb10c
7c79e4e3f9f80d9de2229bd7ac86a52cfb6f323279a477b4fb7183caf7c8cfcb
7060687f77c3d75dcf404889c3c131f330ee908e97bf3d9c116ca1af40243dd0
73e7209d603181e21ab2b9325f86dcf77bc88a5bbcfb4a46d8168d82f2dac4ef
a0562c1d2bcda24b9d00b272bdd212d75abed692d340f525912a9cde64b89d9d
-----END OpenVPN Static key V1-----
</tls-auth>
```

ДОДАТОК В ЛІСТИНГ ПРОГРАМИ

FileServer

FilesServiceImpl.java

```
/**
 * this class is implementation of S3FilesTransfer
 * @see FilesService
 */
@Service
public class FilesServiceImpl implements FilesService
{

   private FileStorage fileStorage;

   @Autowired
   public FilesServiceImpl(FileStorage fileStorage){
      this.fileStorage = fileStorage;
   }

   @Override
   public void saveFile(String id, MultipartFile file){
      byte[] fileBytes;
      try {
         fileBytes = file.getBytes();
         fileStorage.save(id, fileBytes);
      } catch (IOException e) {
         throw new FileServiceException();
      }
   }

   @Override
   public byte[] getFile(String id) {
      return fileStorage.load(id);
   }

   @Override
   public void deleteFile(String id) {
      fileStorage.delete(id);
   }

   @Override
   public Map<Object, Long> findAll() {
      return fileStorage.findAll();
   }
}
```

FilesApiController.java

```
/**
 * Rest Controller to manage client requests regarding
files stored on this host.
 *
 * @author user
 * @version 1.0
 * @since 2017-11-19
 */
@RestController
public class FilesApiController implements FilesApi {

   private FilesService filesService;
   private Logger logger;
   private HttpClient httpClient;

   @Autowired
   public FilesApiController(FilesService filesService,
Logger logger, HttpClient httpClient) {
      this.filesService = filesService;
      this.logger = logger;
      this.httpClient = httpClient;
   }

   /**
    * Finds and returns a file specified by ID under
which the file is stored on this server.
    *
    * @param fileId ID under which a file is stored on
this host.
    * @return response entity consisting of resource if
found and of http status
    */


   @Override
   public ResponseEntity<Resource>
downloadFile(@PathVariable("fileId") String fileId,
@RequestHeader(value = "X-AUTH", required =
false) String X_AUTH) {
      if
(httpClient.validateToken(X_AUTH).getStatusCode()
== HttpStatus.OK) {
         HttpServletRequest request =
((ServletRequestAttributes)
RequestContextHolder.getRequestAttributes()).getReq
uest();
         logger.info("Received request for file: {}",
fileId);
         byte[] fileBytes = filesService.getFile(fileId);
         if (fileBytes != null) {
            logger.info("Received bytes from DB: {}",
fileBytes);
            Resource resource = new
InputStreamResource(new
ByteArrayInputStream(fileBytes));
            MultiValueMap<String, String> headers =
new HttpHeaders();
            headers.add("Content-Type",
"application/octet-stream ");
            headers.add("Content-Disposition",
String.format("attachment; filename=%s", fileId));
            return new ResponseEntity<>(resource,
headers, HttpStatus.OK);
         } else {
            return new
ResponseEntity<>(HttpStatus.BAD_REQUEST);
         }
      }
```

```
        return new
ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }

  @Override
  public ResponseEntity<String> getAllFilesInfo() {
    Map<Object, Long> all = filesService.findAll();
    return ResponseEntity.ok(all.toString());
  }

  /**
     * Finds a file stored on this server by file ID.
     *
     * @param fileId ID under which a file must be
stored on this host.
     * @return response entity consisting of http status
     */
    @Override
    public ResponseEntity<Void>
uploadFile(@RequestPart(value = "fileId", required =
false) String fileId, @RequestPart("file") MultipartFile
file, @RequestHeader(value = "X-AUTH", required =
false) String X_AUTH) {
        if
(httpClient.validateToken(X_AUTH).getStatusCode()
== HttpStatus.OK) {
            if (file.isEmpty())
                logger.info("File is empty");
            else logger.info("File Size " + file.getSize());

            filesService.saveFile(fileId, file);
            return new ResponseEntity<>(HttpStatus.OK);
        }
        return new
ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }

    @Override
    public ResponseEntity<Void>
deleteFileById(@PathVariable("fileId") String fileId,
@RequestHeader(value = "X-AUTH", required =
false) String X_AUTH) {
        logger.info("Received request for file deletion:
{}", fileId);
        if
(httpClient.validateToken(X_AUTH).getStatusCode()
== HttpStatus.OK) {
            filesService.deleteFile(fileId);
            return new ResponseEntity<>(HttpStatus.OK);
        }
        return new
ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }
}
```

FileStorageMongoImpl.java

```
/**
 * this class is implementation of FileStorage class
using MongoDB as storage
 * @see FileStorage
 */
public class FileStorageMongoImpl implements
FileStorage {

    private GridFsTemplate gridFsTemplate;
    private Logger logger;

    @Autowired
    public FileStorageMongoImpl(GridFsTemplate
gridFsTemplate, Logger logger) {
        this.gridFsTemplate = gridFsTemplate;
        this.logger = logger;
    }

    @Override
    public void save(String id, byte[] bytes) {
        logger.debug("Storing file into MongoDB");

        if
(!gridFsTemplate.find(createFindByIdQuery(id)).isEm
pty()) {
            MongoException mongoException = new
MongoException("File with specified id already
exist");
            logger.error(mongoException.getMessage(),
mongoException);
            throw mongoException;
        }
        gridFsTemplate.store(new
ByteArrayInputStream(bytes), id);
    }

    @Override
    public byte[] load(String id) {
        logger.debug("Loading file from MongoDB");

        GridFSDBFile file =
gridFsTemplate.findOne(createFindByIdQuery(id));
        if (file == null ) {
            MongoException mongoException = new
MongoException(String.format("File with id %s does
not exist in MongoDB", id));
            logger.error(mongoException.getMessage(),
mongoException);
            throw mongoException;
        }

        ByteArrayOutputStream byteArrayOutputStream
= new ByteArrayOutputStream();
        try {
            file.writeTo(byteArrayOutputStream);
        } catch (IOException e) {
            logger.error(e.getMessage(), e);
            throw new MongoException("Can't load file");
        }

        return byteArrayOutputStream.toByteArray();
    }

    @Override
    public Map<Object, Long> findAll() {
        logger.debug("Loading file from MongoDB");
```

```
        List<GridFSDBFile> gridFSDBFiles =
gridFsTemplate.find(new Query());

        if (gridFSDBFiles == null ||
gridFSDBFiles.isEmpty()) {
            MongoException mongoException = new
MongoException(String.format("File with id %s does
not exist in MongoDB"));
            logger.error(mongoException.getMessage(),
mongoException);
            throw mongoException;
        }

        return
gridFSDBFiles.stream().collect(Collectors.toMap(Grid
FSDBFile::getId, GridFSDBFile::getLength));
```

Distribution Server

Action.java

```
@Entity
@Table(name = "ds_action")
@JsonIgnoreProperties(ignoreUnknown = true)
public class Action {
  @Id
  @GeneratedValue
  @Column(name = "id_action")
  private long id;
  @ManyToOne
  @JoinColumn(name = "id_device")
  private Device device;
  @Column
  private String type;

  public long getId() {
    return id;
  }

  public void setId(long id) {
    this.id = id;
  }

  public Device getDevice() {
    return device;
  }

  public void setDevice(Device device) {
    this.device = device;
  }

  public String getType() {
    return type;
  }

  public void setType(String type) {
    this.type = type;
  }
}
```

ConnectionInfo.java

```
@Entity
@Table(name = "ds_connection_info")
@JsonIgnoreProperties(ignoreUnknown = true)
public class ConnectionInfo {
  @Id
  @GeneratedValue
  @Column(name = "id_connection_info")
  private long id;

  @Column(name = "vpn_config")
  private String vpnConfig;

  public ConnectionInfo() {
  }

  public long getId() {
    return id;
  }

  public void setId(long id) {
    this.id = id;
  }


  public String getVpnConfig() {
    return vpnConfig;
  }

  public void setVpnConfig(String vpnConfig) {
    this.vpnConfig = vpnConfig;
  }
}
```

Device.java

```
@Entity
@Table(name = "ds_device")
@JsonIgnoreProperties(ignoreUnknown = true)
public class Device  {
  @Id
  @GeneratedValue
  @Column(name = "id_device")
  private long id;
  @Column
  private String guid;
```

```
  @Column
  private String email;
  @Column
  private String otp;
  @Column
  private boolean activated;

  @OneToOne//(mappedBy = "id_public_key")
  @JoinColumn(name = "id_public_key")
  private PublicKey publicKey;

  @OneToOne(cascade = CascadeType.ALL)//, fetch =
FetchType.LAZY)//fetch default eager
  @JoinColumn(name = "id_server")
  private Server server;

  public Device() {
  }

  public String getGuid() {
    return guid;
  }

  public void setGuid(String guid) {
    this.guid = guid;
  }

  public long getId() {
    return id;
  }

  public void setId(long id) {
    this.id = id;
  }

  public String getEmail() {
    return email;
  }

  public void setEmail(String email) {
    this.email = email;
  }

  public String getOtp() {
    return otp;
  }

  public void setOtp(String otp) {
    this.otp = otp;
  }

  public boolean isActivated() {
    return activated;
  }

  public void setActivated(boolean activated) {
    this.activated = activated;
  }

  public PublicKey getPublicKey() {
    return publicKey;
  }

  public void setPublicKey(PublicKey publicKey) {
    this.publicKey = publicKey;
  }

  public Server getServer() {
    return server;
  }

  public void setServer(Server server) {
    this.server = server;
  }
}
```

License.java

```
@Entity
@Table(name = "ds_license")
@JsonIgnoreProperties(ignoreUnknown = true)
public class License {

  @Id
  @Column(name = "id_license")
  @GeneratedValue
  private long id;
  @Column(name = "license_key")
  private String licenseKey;

  @ManyToOne
  @JoinColumn(name = "id_organization")
  private Organization organization;

  @Column(name = "activated")
  private boolean activated;

  @Column(name = "expiration")
  private Date expiration;

  public long getId() {
    return id;
  }

  public void setId(long id) {
    this.id = id;
  }

  public String getLicenseKey() {
    return licenseKey;
  }

  public void setLicenseKey(String licenseKey) {
    this.licenseKey = licenseKey;
  }


  public Organization getOrganization() {
    return organization;
  }

  public void setOrganization(Organization
organization) {
    this.organization = organization;
  }

  public boolean isActivated() {
```

```
    return activated;
  }

  public void setActivated(boolean activated) {
    this.activated = activated;
  }

  public Date getExpiration() {
    return expiration;
  }

  public void setExpiration(Date expiration) {
    this.expiration = expiration;
  }
}
```

Organization.java

```
@Entity
@Table(name = "ds_organization")
@JsonIgnoreProperties(ignoreUnknown = true)
public class Organization {

  @Id
  @GeneratedValue
  @Column(name = "id_organization")
  private long id;
  @Column
  private String guid;
  @Column
  private String name;
  @Column
  private String plan;

  public long getId() {
    return id;
  }

  public void setId(long id) {
    this.id = id;
  }

  public String getGuid() {
    return guid;
  }

  public void setGuid(String guid) {
    this.guid = guid;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getPlan() {
    return plan;
  }

  public void setPlan(String plan) {
    this.plan = plan;
  }
}
```

PublicKey.java

```
@Entity
@Table(name = "ds_public_key")
@JsonIgnoreProperties(ignoreUnknown = true)
public class PublicKey {

  @Id
  @GeneratedValue
  @Column(name = "id_key")
  private long id;

  @Column(name = "value")
  private String value;

  @Column(name = "alg")
  private String algorithm;

  @Column(name = "expiration")
  private Date expiration;

  public long getId() {
    return id;
  }

  public void setId(long id) {
    this.id = id;
  }

  public String getValue() {
    return value;
  }

  public void setValue(String value) {
    this.value = value;
  }

  public String getAlgorithm() {
    return algorithm;
  }

  public void setAlgorithm(String algorithm) {
    this.algorithm = algorithm;
  }

  public Date getExpiration() {
    return expiration;
  }

  public void setExpiration(Date expiration) {
    this.expiration = expiration;
  }
}
```

Server.java

```
@Entity
@Table(name = "ds_server")
@JsonIgnoreProperties(ignoreUnknown = true)
public class Server{
```

```
@Id
@GeneratedValue
@Column(name = "id_server")
private long id;
@Column
private String fqdn;
@Column
private int port;
@OneToOne
@JoinColumn(name = "id_public_key")
private PublicKey publicKey;
@OneToOne//(mappedBy = "id_license")
@JoinColumn(name = "id_license")
private License license;
@OneToOne//(mappedBy = "id_connection")
@JoinColumn(name = "id_connection")
private ConnectionInfo connectionInfo;

public long getId() {
  return id;
}

public void setId(long id) {
  this.id = id;
}

public String getFqdn() {
  return fqdn;
}

public void setFqdn(String fqdn) {
  this.fqdn = fqdn;
}

public int getPort() {
  return port;
}

public void setPort(int port) {
  this.port = port;
}

public PublicKey getPublicKey() {
  return publicKey;
}

public void setPublicKey(PublicKey publicKey) {
  this.publicKey = publicKey;
}

public License getLicense() {
  return license;
}

public void setLicense(License license) {
  this.license = license;
}

public ConnectionInfo getConnectionInfo() {
  return connectionInfo;
}

  public void setConnectionInfo(ConnectionInfo
connectionInfo) {
    this.connectionInfo = connectionInfo;
  }
}
DistributionService.java
;

@Service
public class DistributionService {

  private static Logger logger =
LoggerFactory.getLogger(DistributionController.class)
;
  long DEFAULT_EXPIRATION =
TimeUnit.DAYS.toMillis(30);

  @Autowired
  private DeviceRepository deviceRepository;
  @Autowired
  private DistributionService distributionService;

  @Autowired
  private ConnectionInfoRepository
connectionInfoRepository;
  @Autowired
  private PublicKeyRepository publicKeyRepository;
  @Autowired
  private ServerRepository serverRepository;

  @Transactional
  public ResponseEntity<String>
distributeClient(@RequestBody DistributionBodyDTO
distributionBody,
                                  @RequestHeader String
X_AUTH) {

    String csr = new String(new
byte[]{0,23,34,56,56,67});
    try {
//      byte[] bytes = processCsr(distributionBody);

    } catch (Exception e) {
      return
ResponseEntity.status(HttpStatus.UNPROCESSABLE
_ENTITY).build();
    }

    Server server = getServer(X_AUTH);

    if (server == null) {
      return
ResponseEntity.status(HttpStatus.BAD_REQUEST).b
uild();
    }


    try {
      ObjectMapper mapper = new ObjectMapper();
      Device foundDevice =
deviceRepository.findByEmailAndOtp(distributionBod
y.getEmail(), distributionBody.getOtp());
```

```
    if (foundDevice == null ||
foundDevice.isActivated()) {
      return
ResponseEntity.status(HttpStatus.BAD_REQUEST).b
uild();
    }

    foundDevice.setActivated(true);
    PublicKey publicKey = new PublicKey();
    publicKey.setAlgorithm("Base64");

publicKey.setExpiration(Date.from(Instant.now().plus
Millis(DEFAULT_EXPIRATION)));

publicKey.setValue(distributionBody.getPublicKey());

    foundDevice.setPublicKey(publicKey);

   } catch (Exception e) {
    logger.info("asssss");
    logger.info(e.getCause().toString());
    return
ResponseEntity.status(HttpStatus.BAD_REQUEST).b
uild();
   }

   return
ResponseEntity.status(HttpStatus.OK).body(csr);
  }
  private Server getServer(String X_AUTH) {
   Server server = null;
   PublicKey publicKey =
publicKeyRepository.findByValue(X_AUTH);
   if (publicKey == null) {
    logger.info("Public key is null");
    return null;
   }


   server =
serverRepository.findByPublicKey(publicKey);
   if (server == null) {
    logger.info("Server is null");
    return null;
   }
   return server;
  }

  private byte[] processCsr(@RequestBody
DistributionBodyDTO distributionBody) throws
Exception {
   try {
    String csr = distributionBody.getCsr();
    KeyStore p12 = KeyStore.getInstance("pkcs12");
    p12.load(new FileInputStream("pkcs.p12"),
"password".toCharArray());
   } catch (Exception e) {

   }
   return new byte[0];
  }
}
RegistrationService

@Service
public class RegistrationService {

  private static Logger logger =
LoggerFactory.getLogger(RegistrationController.class)
;
  @Autowired
  DeviceRepository deviceRepository;

  @Autowired
  ConnectionInfoRepository connectionInfoRepository;
  @Autowired
  PublicKeyRepository publicKeyRepository;
  @Autowired
  ServerRepository serverRepository;
  @Autowired
  OrganizationRepository organizationRepository;
  @Autowired
  LicenseRepository licenseRepository;
  private long DEFAULT_EXPIRATION =
TimeUnit.DAYS.toMillis(365);


  @Transactional
  public ResponseEntity<String>
createNewOrganizationLicense(String orgId) {
   Organization organization =
organizationRepository.findOne(Long.valueOf(orgId));

   String licenseKey =
UUID.randomUUID().toString().toUpperCase();
   License license = new License();
   license.setActivated(false);
   license.setLicenseKey(licenseKey);
   license.setOrganization(organization);

license.setExpiration(Date.from(Instant.now().plusMill
is(DEFAULT_EXPIRATION)));

   return
ResponseEntity.status(HttpStatus.OK).body(new
JSONObject().put("licenseKey",
licenseKey).toString());
  }


  public ResponseEntity<String>
getOrganizationLicenses(String orgId) {
   return null;
  }


  @Transactional
  public ResponseEntity<String>
registerClient(ClientRegistrationDTO
registrationBody, String X_AUTH) {
   Server server = getServer(X_AUTH);

   if (server == null) {
```

```
      return
ResponseEntity.status(HttpStatus.BAD_REQUEST).b
uild();
    }

    Device device = new Device();
    device.setActivated(false);
    String guid = UUID.randomUUID().toString();
    device.setGuid(guid);
    device.setPublicKey(null);
    device.setServer(null);
    device.setEmail(registrationBody.getEmail());
    device.setOtp(registrationBody.getOtp());


    return ResponseEntity.ok(new
JSONObject().put("guid", guid).toString());
  }
  private Server getServer(String X_AUTH) {
    Server server = null;
    PublicKey publicKey =
publicKeyRepository.findByValue(X_AUTH);
    if (publicKey == null) {
      logger.info("Public key is null");
      return null;
    }


    server =
serverRepository.findByPublicKey(publicKey);
    if (server == null) {
      logger.info("Server is null");
      return null;
    }
    return server;
  }


  @Transactional
  public ResponseEntity<String>
registerOrganization(OrganizationBodyDTO
organizationBody) {
    Organization organization = new Organization();

    license.setActivated(true);
    return ResponseEntity.ok().build();
  }
}
```

AuthServer

FileApiControllerImpl

```
@RestController
public class FileApiControllerImpl implements FileApi
{

    private Logger logger;
    private ResourceService resourceService;
    private AuthService authService;
    private RabbitMQProducer rabbitMQProducer;
organization.setName(organizationBody.getName());
    organization.setPlan(organizationBody.getPlan());

    Organization newOrg =
organizationRepository.save(organization);

    return ResponseEntity.ok(newOrg.getId()+"");
  }


  @Transactional
  public ResponseEntity<Void>
registerServer(RegistrationBodyDTO
registrationBody) {

    String licenseKey = registrationBody.getLicense();
    License license =
licenseRepository.findByLicenseKeyAndActivated(lic
enseKey, false);

    ConnectionInfo connectionInfo = new
ConnectionInfo();

connectionInfo.setVpnConfig(registrationBody.getCon
fig().toString());

    Server server = new Server();
    server.setFqdn(registrationBody.getFqdn());
    server.setConnectionInfo(connectionInfo);
    server.setLicense(license);
//    server.setOrganization(null);
    server.setPort(registrationBody.getPort());

    PublicKey publicKey = new PublicKey();
    publicKey.setAlgorithm("Base64");

publicKey.setExpiration(Date.from(Instant.now().plus
Millis(DEFAULT_EXPIRATION)));

publicKey.setValue(registrationBody.getPublicKey());

    server.setPublicKey(publicKey);
```

```
    @Value("${fileservice.int.cipher-file-size}")
    private int maxFileSize;

    @Autowired
    public FileApiControllerImpl(ResourceService
resourceService, Logger logger,
                                 AuthService authService,
RabbitMQProducer rabbitMQProducer) {
        this.rabbitMQProducer = null;
        this.resourceService = resourceService;
```

```
        this.logger = logger;
        this.authService = authService;
    }


@PreAuthorize(AuthorizationApiControllerImpl.HAS
_ROLE)
    @Override
    public ResponseEntity<FileMetadataDTO>
uploadFile(FileMetadataDTO fileMetadata,
@RequestPart("file") MultipartFile file,
@RequestHeader(value="web", required=false)
Optional<String> web) {
        try {
            logger.info("Share file" + fileMetadata);
            ResponseEntity<FileMetadataDTO> response
=
resourceService.checkFileSizeAndUpload(fileMetadat
a, file, web);

            if (rabbitMQProducer != null)
                rabbitMQProducer.send(new
Statistic(fileMetadata.getFileUUID(), new Date(),
fileMetadata.getOwnerId(), "upload",
fileMetadata.getFileSize()));

            if (response != null)
                return response;

        } catch (IOException |
NoSuchAlgorithmException e) {
            logger.error(e.getMessage());
        }

        return new
ResponseEntity<>(HttpStatus.BAD_REQUEST);

    }


@PreAuthorize(AuthorizationApiControllerImpl.HAS
_ROLE)
    @Override
    public ResponseEntity<Void>
shareFile(@RequestBody ShareDataDTO shareData) {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication
();
        if (!isUserOwnerOfResource(authentication,

shareData.getFileMetadata().getResourceId())) {
            throw new PermissionException("not
authorized to share current resource");
        }
        if (shareData != null) {
            FileMetadataDTO fileMetadataDTO =
shareData.getFileMetadata();
            List<Long> userIdList =
shareData.getPermittedUserIds();
            Set<Long> userIdSet = null;
            if (userIdList != null && !userIdList.isEmpty())
{
                userIdSet = new
HashSet<>(shareData.getPermittedUserIds());
            }
            boolean isSharedSuccessfully =
resourceService.shareExistingResource(fileMetadataD
TO, userIdSet);
            if (isSharedSuccessfully) {

                if (rabbitMQProducer != null)
                    rabbitMQProducer.send(new
Statistic(fileMetadataDTO.getFileUUID(), new Date(),
fileMetadataDTO.getOwnerId(), "share",
fileMetadataDTO.getFileSize()));

                return new
ResponseEntity<>(HttpStatus.OK);
            } else {
                return new
ResponseEntity<>(HttpStatus.BAD_REQUEST);
            }
        }
        return new
ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }


@PreAuthorize(AuthorizationApiControllerImpl.HAS
_ROLE)
    @Override
    public ResponseEntity<FileMetadataDTO>
addFileMetadata(@RequestBody FileMetadataDTO
fileMetadata) {
        FileMetadataDTO fileMetadataDTO =
resourceService.addResourceMetadata(fileMetadata);
        HttpHeaders httpHeaders = new HttpHeaders();

httpHeaders.setContentType(MediaType.APPLICATI
ON_JSON_UTF8);

        if (fileMetadataDTO != null) {
            return new ResponseEntity<>(httpHeaders,
HttpStatus.OK);
        }

        return new ResponseEntity<>(httpHeaders,
HttpStatus.BAD_REQUEST);
    }


@PreAuthorize(AuthorizationApiControllerImpl.HAS
_ROLE)
    @Override
    public
ResponseEntity<org.springframework.core.io.Resource
> downloadFile(@PathVariable("file-id") String uuid,
@RequestHeader Optional<String> desktop) {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication
();
        if
(!resourceService.isUserCanReadFile(authentication.ge
tPrincipal().toString(), uuid)) {
```

```
            throw new PermissionException("not permitted
to download current resource");
        }

        byte[] fileBytes =
resourceService.downloadFile(uuid, desktop);

        if (fileBytes == null)
            return new
ResponseEntity<>(HttpStatus.BAD_REQUEST);

        logger.info("Successfully loaded bytes from file
service", fileBytes);

        ua.authservice.entity.Resource res =
resourceService.getFileByUUID(uuid);

        ByteArrayResource resource = new
ByteArrayResource(fileBytes);
        HttpHeaders headers = new HttpHeaders();
        headers.add("Content-Type",
res.getMimeType());
        headers.add("Content-Disposition",
String.format("attachment; filename=%s",
resourceService.getFileNameByUUID(uuid)));

            headers.add("Content-Key",
res.getSecretKey().getKey());

//            rabbitMQProducer.send(new Statistic(uuid,
new Date(), res.getOwner().getUserId(), "download",
res.getSize()));

        return ResponseEntity.ok()
                .headers(headers)
                .contentLength(res.getSize())

.contentType(MediaType.parseMediaType("applicatio
n/octet-stream"))
                .body(resource);
    }


@PreAuthorize(AuthorizationApiControllerImpl.HAS
_ROLE)
    @Override
    public ResponseEntity<List<FileMetadataDTO>>
listFilesByOffsetAndPageLimit(@PathVariable("userI
d") Long userId, Optional<Integer> offset,
Optional<Integer> limit) {
        List<FileMetadataDTO> response =
resourceService.getUserFilesList(userId);
        if (response.isEmpty()) {
            logger.info("Returning No Content");
            return new
ResponseEntity<>(HttpStatus.NO_CONTENT);
        }
        logger.info("Response ready: {}", response);
        return new ResponseEntity<>(response,
HttpStatus.OK);
    }


@PreAuthorize(AuthorizationApiControllerImpl.HAS
_ROLE)
    @Override
    public ResponseEntity<Void>
deleteFile(@PathVariable("userId") Long userId,
@PathVariable("fileId") Long fileId) {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication
();
        if (!isUserOwnerOfResource(authentication,
fileId)) {
            throw new PermissionException("not
authorized to delete current resource");
        }
        if (resourceService.deleteResource(userId, fileId))
{
            return new ResponseEntity<>(HttpStatus.OK);
        } else {
            return new
ResponseEntity<>(HttpStatus.BAD_REQUEST);
        }
    }


@PreAuthorize(AuthorizationApiControllerImpl.HAS
_ROLE)
    @Override
    public ResponseEntity<FileMetadataDTO>
getFileMetadata(@PathVariable("file-id") Long
resourceId) {
        FileMetadataDTO fileMetadataDTO =
resourceService.getFileMetadata(resourceId);
        return new ResponseEntity<>(fileMetadataDTO,
HttpStatus.OK);
    }


@PreAuthorize(AuthorizationApiControllerImpl.HAS
_ROLE)
    @Override
    public ResponseEntity<List<FileMetadataDTO>>
listFilesByFilter(@NotNull @RequestParam(value =
"filter", required = true) String filter) {

        List<FileMetadataDTO> fileMetadataDTOList;
        FilesFilterDTO filterDTO;
        ObjectMapper mapper = new ObjectMapper();
        try {
            filterDTO = mapper.readValue(filter,
FilesFilterDTO.class);
        } catch (IOException e) {
            logger.error(String.format("Can not read value
from json string: %s.", filter));
            throw new
DataValidationException(String.format("Can not read
value from json string: %s.", filter));
        }
        fileMetadataDTOList =
resourceService.listFilesByFilter(filterDTO);
        if (fileMetadataDTOList.size() > 0) {
```

```
            return new
ResponseEntity<>(fileMetadataDTOList,
HttpStatus.OK);
        } else {
            return new
ResponseEntity<>(HttpStatus.NO_CONTENT);
        }
    }

    private boolean
isUserOwnerOfResource(Authentication
authentication, Long fileId) {
        UserDTO userDTO =
authService.findUserByEmail(authentication.getPrinci
pal().toString());
        if
(resourceService.isUserOwner(userDTO.getId(),
fileId)) {
            logger.debug("user is owner of resource");
            return true;
        } else return false;
    }
}
UserAPIController

@RestController
public class UserApiControllerImpl implements
UserApi {

    private AuthService authService;
    @Autowired
    private UserDAO userDAO;
    private Logger logger;
    private EmailValidator emailValidator;
    private JwtUtil jwtUtil;

    @Autowired
    public UserApiControllerImpl(AuthService
authService, Logger logger, EmailValidator
emailValidator, JwtUtil jwtUtil) {
        this.authService = authService;
        this.logger = logger;
        this.emailValidator = emailValidator;
        this.jwtUtil = jwtUtil;
    }


    @PreAuthorize("hasRole('ADMIN')")
    @Override
    public ResponseEntity<UserDTO> addUser(@Valid
@RequestBody UserCredentialsDTO user) {
        emailValidator.validateEmail(user.getEmail());
        UserDTO userDTO = authService.addUser(user);

        HttpHeaders httpHeaders = new HttpHeaders();

httpHeaders.setContentType(MediaType.APPLICATI
ON_JSON);

        httpHeaders.add(Constants.TOKEN_HEADER,
jwtUtil.generateToken(userDTO.getEmail(),
userDTO.getRole().toString()));

        return new ResponseEntity<>(userDTO,
httpHeaders, HttpStatus.OK);
    }

    @PreAuthorize("hasRole('ADMIN')")
    @Override
    public ResponseEntity<Void>
deleteUserById(@PathVariable("userId") Long userId)
{
        authService.deleteUser(userId);
        return new ResponseEntity<>(HttpStatus.OK);
    }


@PreAuthorize(AuthorizationApiControllerImpl.HAS
_ROLE)
    @Override
    public ResponseEntity<UserDTO>
getUserById(@PathVariable("userId") Long userId) {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication
();
        if(isRoleUser(authentication)) {
            UserDTO userDTO =
authService.findUserByEmail(authentication.getPrinci
pal().toString());
            if( !userId.equals(userDTO.getId())) {
                logger.error(String.format("User with role
USER and ID %d not authorized to getById user with
ID %d", userDTO.getId(), userId));
                throw new PermissionException("Not
authorized to get by id the other user");
            }
        }
        UserDTO userDTO =
authService.findUserById(userId);
        return new ResponseEntity<>(userDTO,
HttpStatus.OK);
    }

    @PreAuthorize("hasRole('TECH_SUPP') or
hasRole('ADMIN')")
    @Override
    public ResponseEntity<UserDTO>
updateUserById(@PathVariable("userId") Long
userId, @Valid @RequestBody UserDTO user) {
        if (user.getEmail() != null) {
            emailValidator.validateEmail(user.getEmail());
        }
        UserDTO updatedUser =
authService.updateUser(userId, user);
        return new ResponseEntity<>(updatedUser,
HttpStatus.OK);
    }


@PreAuthorize(AuthorizationApiControllerImpl.HAS
_ROLE)
    @Override
    public ResponseEntity<UserDTO>
updateUserCredentials(@Min(1)
@PathVariable("userId") Long userId, @Valid
```

```
@RequestBody UserCredentialsDTO userCredentials)
{
      Authentication authentication =
SecurityContextHolder.getContext().getAuthentication
();
      if(isRoleUser(authentication)) {
         UserDTO userDTO =
authService.findUserByEmail(authentication.getPrinci
pal().toString());
         if( !userId.equals(userDTO.getId())) {
            logger.error(String.format("User with role
USER and ID %d tries to update user with ID %d",
userDTO.getId(), userId));
            throw new PermissionException("Not
authorized to update the other user");
         }
      }
      if (userCredentials.getEmail() != null) {

emailValidator.validateEmail(userCredentials.getEmail
());
      }
      UserDTO updatedUser =
authService.updateUserCredentials(userId,
userCredentials);
      return new ResponseEntity<>(updatedUser,
HttpStatus.OK);
   }


@PreAuthorize(AuthorizationApiControllerImpl.HAS
_ROLE)
   @Override
   public ResponseEntity<List<UserDTO>>
listUsersByOffsetAndPageLimit(@Min(0)
@PathVariable("offset") Integer offset, @Min(5)
@Max(50) @PathVariable("limit") Integer limit) {
      List<UserDTO> userDTOList =
authService.listUsersPerPage(offset, limit,
SortingOrderEnum.ASC);
      if (userDTOList.size() > 0) {
         return new ResponseEntity<>(userDTOList,
HttpStatus.OK);
      } else {
         return new ResponseEntity<>(userDTOList,
HttpStatus.NO_CONTENT);
      }
   }

   @PreAuthorize("hasRole('ADMIN')")
   @Override
   public ResponseEntity<UserDTO>
updateUserRole(@Min(1) @PathVariable("userId")
Long userId, @Valid @RequestBody UserDTO user) {
      if (user.getRole() == null) {
         logger.warn("Role is null.");
         return new
ResponseEntity<>(HttpStatus.BAD_REQUEST);
      }
      UserDTO updatedUser =
authService.updateUser(userId, user);
      return new ResponseEntity<>(updatedUser,
HttpStatus.OK);
   }


@PreAuthorize(AuthorizationApiControllerImpl.HAS
_ROLE)
   @Override
   public ResponseEntity<List<UserDTO>>
listUsersByFilter(@RequestParam(value = "filter",
required = false) Optional<String> filter) {
      List<UserDTO> userDTOList;
      UsersFilterDTO filterDTO;
      if (filter.isPresent()) {
         String filterJSONString = filter.get();
         ObjectMapper mapper = new ObjectMapper();
         try {
            filterDTO =
mapper.readValue(filterJSONString,
UsersFilterDTO.class);
         } catch (IOException e) {
            logger.error(String.format("Can not read
value from json string: %s.", filterJSONString));
            throw new
DataValidationException(String.format("Can not read
value from json string: %s.", filterJSONString));
         }
      } else {
         filterDTO = new UsersFilterDTO();
      }
      userDTOList =
authService.listUsersByFilter(filterDTO);
      if (userDTOList.size() > 0) {
         return new ResponseEntity<>(userDTOList,
HttpStatus.OK);
      } else {
         return new ResponseEntity<>(userDTOList,
HttpStatus.NO_CONTENT);
      }
   }

   private boolean isRoleUser(Authentication
authentication) {
      List<SimpleGrantedAuthority> authorities =
(List<SimpleGrantedAuthority>)
authentication.getAuthorities();
      SimpleGrantedAuthority authority =
authorities.get(0);
      return
authority.getAuthority().equals("ROLE_USER");
   }

   @Override
   public ResponseEntity<String>
generateNewOtp(@PathVariable("userId") Long
userId) {
      Optional<User> user =
userDAO.findElementById(userId);
      try {
         String s =
authService.generateOtpWithEmail(user.get());
         return new ResponseEntity<>(new
JSONObject().put("otp",s).toString(), HttpStatus.OK);
      } catch (IOException e) {
         e.printStackTrace();
```

```
            return new
ResponseEntity<>(HttpStatus.BAD_REQUEST);
        }
    }
}
```

CipherServiceImpl

```
@Service
public class CipherServiceImpl implements
CipherService {


    private Cipher cipher;
    private IvParameterSpec ivSpec;
    private Logger logger;
    private String algorithm;

    private SecretKey secretKey;

    @Autowired
    public
CipherServiceImpl(@Value("${cipher.algorithm.name
}")String algorithm, Cipher cipher, IvParameterSpec
ivSpec, Logger logger) {
        this.algorithm = algorithm;
        this.cipher = cipher;
        this.ivSpec = ivSpec;
        this.logger = logger;
    }

    @Override
    public byte[] encrypt(byte[] fileBytes, String key)
throws CipherException {
        if (key.length() != Constants.KEY_SIZE) {
            logger.error("Illegal key size");
            throw new CipherException("Illegal key size");
        } else {
            try {
                byte[] keyBytes =
Base64.getDecoder().decode(key);
                secretKey = new SecretKeySpec(keyBytes,
algorithm);
                cipher.init(Cipher.ENCRYPT_MODE,
secretKey, ivSpec);
                logger.info("Bytes encryption");
                System.out.println(secretKey.toString());
                return cipher.doFinal(fileBytes);
            } catch (InvalidAlgorithmParameterException |
                     InvalidKeyException |
BadPaddingException | IllegalBlockSizeException e) {
                logger.error(e.getMessage(), e);
                throw new
CipherException("ENCRYPTION: "+e.getMessage());
            }
        }

    }
    //TODO: extract key size into properties
    @Override
    public byte[] decrypt(byte[] fileBytes, String key)
throws CipherException {
        if (key.length() != Constants.KEY_SIZE) {
            logger.error("Illegal key size");
            throw new CipherException("Illegal key size");
        } else {
            try {
                secretKey = new
SecretKeySpec(key.getBytes(), algorithm);
                cipher.init(DECRYPT_MODE, secretKey,
ivSpec);
                logger.info("Bytes decryption");
                System.out.println(secretKey.toString());
                return cipher.doFinal(fileBytes);
            } catch (InvalidAlgorithmParameterException |
                     InvalidKeyException |
BadPaddingException | IllegalBlockSizeException e) {
                logger.error(e.getMessage(), e);
                throw new
CipherException("DECRYPTION: "+ e.getMessage());
            }
        }
    }

    @Override
    public String generateKey() throws
NoSuchAlgorithmException {
        KeyGenerator keyGen =
KeyGenerator.getInstance("AES");
        keyGen.init(128);
        SecretKey secretKey = keyGen.generateKey();
        String encodedKey =
Base64.getEncoder().encodeToString(secretKey.getEn
coded());
        return encodedKey;
    }

}
```

PasswordExpirationService

```
/**
 * Created by user on 08.12.17.
 */
@Service
@EnableScheduling
public class PasswordExpirationServiceImpl
implements PasswordExpirationService {

    private Logger logger;
    private PasswordDAO passwordDAO;
    private EmailService emailService;

    @Autowired
    public PasswordExpirationServiceImpl(Logger
logger, PasswordDAO passwordDAO, EmailService
emailService) {
        this.logger = logger;
        this.passwordDAO = passwordDAO;
        this.emailService = emailService;
    }

    @Scheduled(cron = "${cron.expression.password-
notify}")
    @Transactional(readOnly = true)
    @Override
```

```
    public void notifyUserIfPasswordExpiresSoon() {
        logger.info(String.format("Running scheduled
task thread [expiring password notification]: %s",
Thread.currentThread().getName()));

        final long currentTimeMillis =
System.currentTimeMillis();
        Timestamp startTime = new
Timestamp(currentTimeMillis);
        Timestamp endTime = new
Timestamp(currentTimeMillis +
Constants.PASSWORD_EXPIRY_NOTICE_DAYS *
24 * 60 *60 * 1000);
        List<Password> passwords =
passwordDAO.findAllActivePasswordsExpiredWithin
TimeRange(startTime, endTime);
        logger.info(String.format("There are %d
passwords that expire within %tF-%tF.",
passwords.size(), startTime, endTime));
        passwords.forEach(password -> {
            User user = password.getUser();
            if( !user.isStatus()) {
                logger.error(String.format("Status of User
with ID %s is expected to be true, but false instead.",
user.getUserId()));
            } else {
                try{
                    logger.info(String.format("Sending email
to user (ID %d) about expiring password (ID %d)",
user.getUserId(), password.getPasswordId()));
                    emailService.sendMail(user,
EmailType.PASSWORD_EXPIRATION);
                } catch (Exception e) {
                    logger.error(String.format("Exception
occurred when sending email to notify user (ID %d)
about expiring password (ID %d): %s",
                            user.getUserId(),
password.getPasswordId(), e.getMessage()));
                }
            }
        });

        logger.info(String.format("Ending scheduled task
thread [expiring password notification]: %s",
Thread.currentThread().getName()));
    }

    @Scheduled(cron = "${cron.expression.password-
clean}")
    @Transactional
    @Override
    public void disableExpiredPasswords() {
        logger.info(String.format("Running scheduled
task thread [disable expired passwords]: %s",
Thread.currentThread().getName()));

        List<Password> passwords =
passwordDAO.findAllActivePasswordsThatExpired();
        logger.info(String.format("There are %d
passwords already expired.", passwords.size()));
        passwords.forEach(password -> {
            User user = password.getUser();
            logger.info(String.format("Disabling expired
password (ID %1$d, expired on %2$tF %2$tT) for
user (ID %3$d)",
                    password.getPasswordId(),
password.getExpirationTime(), user.getUserId()));
            try{
                password.setStatus(false);
                passwordDAO.update(password);
                emailService.sendMail(user,
EmailType.PASSWORD_EXPIRATION);
            } catch (Exception e) {
                logger.error(String.format("Exception
occurred when disabling expired password (ID %d) for
user (ID %d): %s",
                        password.getPasswordId(),
user.getUserId(), e.getMessage()));
            }

        });

        logger.info(String.format("Ending scheduled task
thread [disable expired passwords]: %s",
Thread.currentThread().getName()));
    }

}
```

UserCredentialsConverter

```
/**
 * Utility class for converting objects between Entity
and DTO
 */
public final class UserCredentialsConverter {

    private UserCredentialsConverter() {
    }

    /**
     * Converts user entity object to corresponding user
with credentials DTO
     *
     * @param user User that needed to be converted to
DTO
     * @return user with credentials DTO object
     * @throws AuthServiceException if User entity is
null
     */
    public static UserCredentialsDTO
convertToDto(User user) {
        if(user == null) {
            throw new AuthServiceException("User entity
to be converted to UserCredentialsDTO is null.");
        }
        UserCredentialsDTO userCredentialsDTO = new
UserCredentialsDTO();
        userCredentialsDTO.setId(user.getUserId());

userCredentialsDTO.setFirstname(user.getFirstName()
);

userCredentialsDTO.setLastname(user.getLastName())
;
```

```
        userCredentialsDTO.setEmail(user.getEmail());
        Role role = user.getRole();
        if(role!=null && role.getRoleName()!=null) {

userCredentialsDTO.setRole(Enum.valueOf(RoleDTO.
class, role.getRoleName()));
        }
        userCredentialsDTO.setPassword("");
        return userCredentialsDTO;
    }

    /**
     * Converts user with credentials DTO object to
corresponding Entity
     *
     * @param userCredentialsDTO User with
credentials data transfer object that needed to be
converted to Entity
     * @return User object
     * @throws AuthServiceException if
UserCredentials DTO is null
     */
    public static User
convertToEntity(UserCredentialsDTO
userCredentialsDTO) {
        if(userCredentialsDTO == null) {
            throw new
AuthServiceException("UserCredentialsDTO to be
converted to User entity is null.");
        }

        User user = new User();
        user.setEmail(userCredentialsDTO.getEmail());

user.setFirstName(userCredentialsDTO.getFirstname()
);

user.setLastName(userCredentialsDTO.getLastname())
;
        List<Password> passwords = new ArrayList<>();
        Password password = new Password();
        BCryptPasswordEncoder passwordEncoder = new
BCryptPasswordEncoder();
        String hashedPassword =
passwordEncoder.encode(userCredentialsDTO.getPass
word());
        password.setPassword(hashedPassword);
        password.setUser(user);
        password.setAlg(BCrypt.class.getSimpleName());
        passwords.add(password);
        user.setPasswords(passwords);
        Role role = new Role();
        if(userCredentialsDTO.getRole() != null) {

role.setRoleName(userCredentialsDTO.getRole().name
());
        }
        user.setRole(role);
        return user;
    }

    public static Device
convertToEntity(UserDeviceCredentialsDTO
userCredentialsDTO, Device device, PasswordType
type) {
        if(userCredentialsDTO == null) {
            throw new
AuthServiceException("UserDeviceCredentialsDTO to
be converted to User entity is null.");
        }

        Password password = new Password();
        BCryptPasswordEncoder passwordEncoder = new
BCryptPasswordEncoder();
        String hashedPassword =
passwordEncoder.encode(userCredentialsDTO.getPass
word());
        password.setPassword(hashedPassword);
        password.setUser(device.getUser());
        password.setType(type);
        password.setStatus(true);
        password.setAlg(BCrypt.class.getSimpleName());

        device.setPassword(password);

        return device;
    }

    /**
     * Converts collection of user entity objects to list of
DTO objects
     *
     * @param users Collection of user objects that will
be converted to DTO
     * @return List of user with credentials DTOs
     */
    public static List<UserCredentialsDTO>
convertToDtoList(Collection<User> users) {
        return users.
                stream().

map(UserCredentialsConverter::convertToDto).
                collect(Collectors.toList());

    }

    /**
     * Converts collection of user with credentials DTOs
to list of entity objects
     *
     * @param userCredentialsDTOS Collection of user
with credentials DTOs that will be converted to entity
     * @return List of User entity objects
     */
    public static List<User>
convertToEntityList(Collection<UserCredentialsDTO>
userCredentialsDTOS) {
        return userCredentialsDTOS.
                stream().

map(UserCredentialsConverter::convertToEntity).
                collect(Collectors.toList());
    }
}
```

JwtAuthFilter

```
@Component
public class JwtAuthenticationFilter extends
GenericFilterBean {

    @Autowired
    private JwtUtil jwtUtil;

    @Override
    public void doFilter(ServletRequest request,
ServletResponse response, FilterChain filterChain)
throws IOException, ServletException {

        try {
            String token = ((HttpServletRequest)
request).getHeader(Constants.TOKEN_HEADER);

            if (token != null) {
                Authentication auth =
jwtUtil.getAuthentication(token);

SecurityContextHolder.getContext().setAuthentication(
auth);

                ((HttpServletResponse)
response).setHeader(Constants.TOKEN_HEADER,
jwtUtil.refresh(token));
            }
        } catch (InvalidTokenException e) {
            ((HttpServletResponse)
response).setStatus(HttpServletResponse.SC_FORBID
DEN);
        }

        filterChain.doFilter(request,response);
    }
}
```