

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Інститут (факультет) Комп'ютерних та інформаційних технологій
Кафедра Обчислювальної техніки та програмування
Спеціальність 123 Комп'ютерна інженерія
Освітня програма Сучасне програмування, мобільні пристрої та комп'ютерні ігри

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи

другого (магістерського) рівня вищої освіти

на тему Дослідження методів захисту веб-сервісу для хмарного
зберігання та обміну файлів

Виконав студент 6 курсу, групи КІТ-М1196

Щербініна Є.М.

(підпис, прізвище та ініціали)

Керівник Філоненко А.М.

(підпис, прізвище та ініціали)

Рецензент Ноздрачова К.Л.

(підпис, прізвище та ініціали)

Нормоконтроль Філоненко А.М.

(підпис, прізвище та ініціали)

Харків 2020

ЗМІСТ

Перелік позначень та скорочень.....	5
Вступ.....	6
1 Дослідження вразливостей веб-застосунків та методів захисту.....	8
1.1 Аналіз загроз безпеки веб-застосунків.....	8
1.2 Найпоширеніші загрози веб-застосунків за версією OWASP.....	9
1.2.1 Ін'єкції (Injections).....	11
1.2.2 Недоліки системи аутентифікації і зберігання сесій (Broken Authentication and Session Management)	11
1.2.3 Незахищеність критичних даних (Sensitive Data Exposure).....	12
1.2.4 Міжсайтовий скриптинг - XSS (Cross Site Scripting).....	13
1.2.5 Небезпечні прямі посилання на об'єкти (Insecure Direct Object References)..	14
1.2.6 Похибки в конфігуруванні (Security Misconfiguration)	15
1.3 Аналіз методів хешування паролів.....	16
1.3.1 Алгоритм хешування MD5	17
1.3.2 Алгоритм хешування SHA1	21
1.3.3 Scrypt.....	24
1.3.4 Аналіз розглянутих методів.....	26
1.4 Аналіз методів шифрування бази даних	29
2 Аналіз використаних технологій та методів розробки	31
2.1 ASP.NET Core	31
2.2 Entity Framework.....	32
2.3 Аналіз і вибір технологій для проектування бази даних.....	34
2.4 Способи зберігання файлів	39

3	Розробка програмного забезпечення	41
3.1	Розробка веб-інтерфейсу	41
3.2	Розробка бази даних	46
3.3	Проектування серверної частини	48
4	Охорона праці та навколишнього середовища	52
4.1	Загальні питання охорони праці	52
4.2	Перелік шкідливих факторів.....	53
4.3	Виробнича санітарія	53
4.3.1	Освітлення	55
4.3.2	Шум і вібрації	55
4.3.3	Електромагнітне випромінювання	56
4.4	Електробезпека	57
4.4.1	Індивідуальне завдання.....	58
4.5	Пожежна безпека	62
4.6	Охорона навколишнього природного середовища	63
5	Цивільний захист	65
5.1	Оперативні дії та основні завдання органів управління та підрозділів під час проведення пожежно-рятувальних робіт.....	65
5.2	Рятування людей при пожежі	67
5.3	Гасіння пожеж	69
6	Техніко-економічна частина.....	72
6.1	Резюме.....	72
6.2	Опис програмного продукту.....	72
6.3	Дослідження й аналіз ринку збуту	73
6.4	Оцінка конкурентоздатності.....	75

6.5 Стратегія маркетингу	77
Висновки.....	82
Список джерел інформації.....	84
Додаток А – Лістинг програмного коду	87

ПЕРЕЛІК ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

XSS	– Міжсайтовий скриптинг
SQL	– Structured query language
OWASP	– Open Web Application Security Project
XML	– Extensible Markup Language
XXE	– Впровадження зовнішніх сутностей
CSRF/XSRF	– Межсайтова підробка запитів
ORM	– Object-relational mapping
API	– Application Programming Interface
HTTP	– HyperText Transfer Protocol
HTTPS	– HyperText Transfer Protocol Secure
SSL	– Secure Sockets Layer
OC	– Операційна система
URL	– Uniform Resource Locator
MD	– Message Digest
SHA	– Scriptores Historiae Augustae
TDE	– Transparent Database Encryption
EFS	– Encrypting File System
MVC	– Model-View-Controller
HTML	– HyperText Markup Language
IIS	– Internet Information Services
UI	– User interface
ПДВ	– Податок на додану вартість
ДСТУ	– Державні стандарти України
ПП	– Програмний продукт

ВСТУП

З поширенням комп'ютерної мережі Інтернет в усі сфери людської діяльності особливого значення набуває інформаційна безпека даних, що виставляються користувачами мережі у різного ступеня відкритості доступ. Кіберпростір не має державних кордонів, людина, що здійснює несанкціонований доступ до комп'ютерної інформації, може перебувати на будь-якої території, автор комп'ютерного вірусу може розмістити його на популярному сайті і тим самим забезпечить його проникнення в різні країни.

Так, згідно з даними Cisco в 2015 році кіберзлочинці все активніше впроваджуються в веб-ресурси шляхом крадіжки даних і доступу до серверів АСУ. це створило умови для зростання вразливостей підробки міжсайтових запитів, незважаючи на те, що з 2014 по 2015 рік категорія міжсайтового скриптинга скоротилася на 47% [1]. Аналіз також показує, що організації досягли певних успіхів в шифруванні інформації при передачі між вузлами, однак збережені дані нерідко залишаються незахищеними. У більшості значущих порушень безпеки за останні кілька років зловмисники скористалися незашифрованими даними, що зберігаються в центрі обробки даних та інших внутрішніх системах, що дозволило хакерам добиратися до цінних відомостей без особливих проблем.

Актуальність проведення досліджень, пов'язаних з розробкою нових методів і способів захисту даних інформаційних систем управління, обумовлена стрімким зростанням впровадження процесів автоматизації в діяльність підприємств і організацій, як державного сектора економіки, так і приватних фірм і компаній.

Серед критеріїв інформаційної безпеки основними є: конфіденційність, цілісність і доступність інформації. Конфіденційність - це збереження в секреті інформації, доступ до якої обмежується вузьким колом користувачів. Цілісність - властивість, згідно з якою інформація зберігає свої початкові або узгоджені з кінцевим користувачем вид і якість, порушенням цілісності є санкціонована

зміна змісту інформації. Доступність – це використання інформації за можливостями користувача, що має відповідні повноваження в необхідному для нього вигляді, час і місце, порушення доступності тягне за собою неможливість отримання або обробки інформації.

Метою дипломної роботи є формування методів захисту інформації та впровадження їх у веб-сервіс хмарного зберігання та обміну файлів. На основі поставленої цілі були поставлені задачі:

- проведення дослідження та аналізу існуючих методів захисту інформації та вибір використовуваних методів;
- проведення аналізу і вибір використовуваних технологій;
- розробка веб-сервісу хмарного зберігання файлів;
- впровадження обраних методів захисту в розроблений продукт;
- оцінка ефективності виконаної розробки.

1 ДОСЛІДЖЕННЯ ВРАЗЛИВОСТЕЙ ВЕБ-ЗАСТОСУНКІВ ТА МЕТОДІВ ЗАХИСТУ

1.1 Аналіз загроз безпеки веб-застосунків

Безпека веб-застосунків це розділ інформаційної безпеки який відповідає за вирішення питань безпеки веб-сайтів, веб-застосунків та веб-сервісів. На високому рівні безпека веб-застосунків базується на принципах безпеки прикладних програм які мають доступ до інтернету.

Більшість атак на веб-застосунки реалізуються шляхом міжсайтового скриптингу (XSS) і SQL-ін'єкцій які зазвичай можливі через недостатню професійність розробника і помилками зв'язаними з обробкою застосунком вхідних і вихідних даних. Зазвичай ці дві загрози стоять на початку усіх чартів які надають інформацію про найчастіші загрози які є наслідком погано написаного коду [2].

Відповідно до постачальника послуг безпеки Cenzic, топ загроз наведено у табл. 1.1.

Таблиця 1.1. – Топ загроз веб-застосунків від Cenzic

Відносна частота загрози	Назва загрози
1	2
37%	Міжсайтовий скриптинг
16%	SQL ін'єкції
5%	Розкриття повного шляху у get запитах
5%	Data breach (information disclosure)
4%	Виконання довільного коду
4%	Пошкодження пам'яті
4%	Cross-site request forgery
3%	Розкриття конфіденційної інформації
3%	Довільне виконання файлів

Закінчення таблиці 1.1 - Топ загроз веб-застосунків від Cenzic

1	2
2%	Виконання локальних файлів серверу
1%	Віддалене виконання файлів
1%	Переповнення буферу
15%	Інші, включаючи ін'єкції JS-коду

1.2 Найпоширеніші загрози веб-застосунків за версією OWASP

Open Web Application Security Project (OWASP) – це відкритий проект забезпечення безпеки веб-застосунків. Товариство OWASP включає в себе корпорації і наукові заклади багатьох країн. OWASP працює над створенням статей, навчальних посібників, рекомендацій, документацій, інструментів і технологій, які зберігаються у відкритому доступі.

OWASP рекомендації щодо безпечної розробки коду описує деякі прийоми і методи які розробники можуть використовувати для створення захищених застосунків які забезпечують конфіденційність, цілісність і доступність інформації[3]:

- валідація вхідних даних;
- кодування вихідних даних відносно належного контексту;
- автентифікація і менеджмент паролів;
- менеджмент сеансів;
- контроль доступу;
- криптографічний захист;
- обробка помилок і логування;
- захист чутливих даних;
- безпека зв'язку;
- системна конфігурація;
- безпека бази даних;
- менеджмент файлів;

- керування пам'яттю;
- загальні прийоми кодування.

Один з найбільших вкладів OWASP вніс за рахунок проекту Top Ten Vulnerabilities – список найбільш частих 10 загроз веб-застосунків, зазвичай список оновлюється з періодом 3 роки [4].

Перелік вразливостей постійно оновлюється і на 2017 виглядає наступним чином:

- ін'єкції (Injections);
- недоліки системи аутентифікації і зберігання сесій (Broken Authentication and Session Management);
- незахищеність критичних даних (Sensitive Data Exposure);
- впровадження зовнішніх XML-сутностей (XXE);
- порушення контролю доступу (Broken Access Control);
- похибки в конфігуруванні (Security Misconfiguration);
- міжсайтовий скриптинг - XSS (Cross Site Scripting);
- небезпечна десериалізація (Insecure Deserialization);
- використання компонентів з відомими вразливостями (Using Components with Known Vulnerabilities);
- недостатнє логування та моніторинг.

Окрім цих 10 найбільш популярних вразливостей веб-застосунків також дуже поширені:

- небезпечні прямі посилання на об'єкти (Insecure Direct Object References);
- відсутність функції контролю доступу (Missing Function Level Access Control);
- межсайтова підробка запитів (Cross-Site Request Forgery, CSRF/XSRF);
- неперевірені переадресації та пересилання (Unvalidated Redirects and Forwards).

1.2.1 Ін'єкції (Injections)

Зазвичай в веб-додатках для зберігання даних використовуються бази даних, звернення до яких найчастіше відбувається за допомогою спеціальної мови SQL. Веб-додатки використовують SQL-запити для того, щоб отримувати, додавати, видаляти або змінювати дані в базі даних, наприклад, при редагуванні своїх даних в особистому кабінеті. При недостатній захищеності цих даних, зловмисник може впровадити код що містить SQL-запит.

Даний вид атаки називається SQL-ін'єкція і є одним з найпоширеніших у наш час. Це дуже небезпечна уразливість, яка може дозволити зловмиснику отримати доступ до бази даних і додавати змінювати та видаляти інформацію. Наприклад, змінити баланс свого рахунку в банку, переглянути конфіденційну інформацію інших користувачів.

Приклад класичної SQL ін'єкції швидше характерний саме для додатків Web Forms. Від атак допомагає захиститися використання параметрів в якості значень запиту. Entity Framework прикриває деякі уразливості. Для того, щоб в MVC або EF додатку спрацювала SQL ін'єкція потрібно дуже постаратись. Однак це можливо якщо ви виконуєте SQL код за допомогою ExecuteQuery методу або викликаєте погано написані збережені процедури.

Незважаючи на те, що Object-relational mapping (ORM) дозволяє уникнути SQL Injection (за винятком наведених вище прикладів), рекомендується обмежувати значення, які можуть приймати поля моделі, а значить і форми. Обмежити можливі значення можна за допомогою валідаторів [5].

1.2.2 Недоліки системи аутентифікації і зберігання сесій (Broken Authentication and Session Management)

Для того, щоб відрізнити одного користувача від іншого, у веб додатку використовуються сесійні куки. Після того, як користувач виконав вхід на сайт, наприклад, за допомогою логіна і пароля, при встановленій опції в браузері зберігається спеціальний ідентифікатор, який потім надається веб-додатком при кожному зверненні. Саме таким чином веб-додаток розуміє, що Ви це Ви.

Суть даної атаки полягає в тому, що зломисник намагається отримати цей унікальний ідентифікатор і скористатися ним для входу на сайт від вашого імені і, якщо на сайті немає додаткових перевірок на цей унікальний ідентифікатор, наприклад, ір-адреса первинного звернення або опція, яка забороняє вести одночасно дві сесії з одного облікового запису, то отримавши цей унікальний ідентифікатор, користувач зможе скористатися всіма можливостями вашого профілю в веб-застосунку.

Функції додатків, які відповідають за аутентифікацію і управління сесіями, найчастіше застосовуються неправильно, наслідком чого стане компрометація паролів, ключів, сеансових токенів, і навіть можливість повністю перехопити сеанс користувача.

1.2.3 Незахищеність критичних даних (Sensitive Data Exposure)

Багато веб-додатків та API можуть некоректно зберігати і обробляти важливу інформацію персональних даних. Зломисники можуть вкрати або змінити таку інформацію, що може стати основою для серйозних фінансових або репутаційних втрат. Інформація повинна зберігатися належним чином, а також повинна бути захищена при передачі по каналах зв'язку.

Деякі веб-додатки не вважають за потрібне захищати конфіденційні та персональні дані. Тому зломисник може отримати до них доступ і впливати на них. Одним з таких прикладів є використання протоколу HTTP для передачі конфіденційних даних. Вся проблема полягає в тому, що при передачі даних через протокол HTTP не відбувається їх шифрування. І використовуючи звичайний сниффер, зломисник може перехопити ці дані.

Щоб уникнути такої ситуації необхідно використовувати протокол HTTPS.

Так само використання HTTPS протоколу для передачі конфіденційних даних надає підтвердження вашого права на інформацію за допомогою SSL сертифікату. Так як його можна перевірити на валідність або термін дії.

Так само наявність шифрування захищає конфіденційні дані від їх крадіжки, навіть якщо зломисник отримав доступ до бази даних. Так, для

паролів, зазвичай, використовується хеш-функція, і відновити його з хешу дуже складно, а його перевірка відбувається шляхом порівняння хешу, введеного пароля і хешу який зберігається в базі даних.

1.2.4 Міжсайтовий скриптинг - XSS (Cross Site Scripting)

Міжсайтовий скриптинг - ще одна помилка валідації призначених для користувача даних, яка дозволяє передати JavaScript код на виконання в браузер користувача. Атаки такого роду часто також називають HTML-ін'єкціями, адже механізм їх впровадження дуже схожий з SQL-ін'єкціями, але на відміну від останніх, впроваджуваний код виконується в браузері користувача.

Не небезпечно. По-перше, зловмисник може викрасти унікальний ідентифікатор (сесійний cookie). І скористатися ним, як описано в розділі 1.2.2 Broken Authentication and Session Management.

По-друге, зловмисник може викрати дані, які вводяться у форми.

По-третє, зловмисник може змінити дані, які виводяться на сторінці. Додати вірусні посилання або замінити, наприклад, банківські реквізити.

XSS вразливість виникає тоді, коли дані, прийняті від користувача, виводяться в браузер без належної фільтрації. Наприклад, програмний код рекламного банера може містити скрипт для перехоплення призначених для користувача даних або навіть прозоре перенаправлення на інші сайти.

Щоб запобігти XSS атаці потрібно валідувати та кодувати вхідні данні та санітизувати на виході.

В першу чергу, необхідно виявити всі потоки даних, цілісність або автентичність яких не контролюється всередині розглянутого компонента веб-додатку. Під компонентом веб-додатку, як правило (хоча і не завжди), маються на увазі елементи його серверної або клієнтської частини, що виконуються в рамках одного процесу ОС.

Як можна ближче до місця появи таких даних в компоненті, необхідно забезпечити їх приведення до очікуваних типів. Вся подальша робота з даними всередині компонента повинна здійснюватися тільки через створені на основі

вихідних даних об'єкти. Важливо пам'ятати, що основним принципом етапу типізації є якомога менша кількість об'єктів строкового типу на його виході URL, адреси електронної пошти, дата, часта інше по стандартизації повинні представляти із себе об'єкти конкретних типів, відмінних від строкового. У вигляді рядків повинні бути представлені тільки ті дані, які насправді є рядком, тобто які дійсно можуть містити довільний повноалфавітний текст. Одразу після типізації по стандартизації, семантику отриманих об'єктів необхідно перевірити на відповідність функціоналу компонента. Наприклад, для цілочисельних типів або дати та часу - це буде перевірка діапазону, для строкових, в більшості випадків, буде достатньо перевірки на відповідність регулярними виразами, а для об'єктів більш комплексних типів необхідно реалізовувати перевірку семантики кожного з його полів і властивостей. Будь-які перевірки, пов'язані з валідацією завжди повинні здійснюватися на основі принципу білого списку, тобто семантика даних повинна відповідати дозволеним критеріям, а не навпаки. Метою даного етапу є отримання гарантії того, що всі дані всередині компонента будуть відповідати реалізованому в ньому функціоналу і не зможуть його порушити.

У всіх місцях, де валідовані і типізовані нами об'єкти залишають межі компонента (або там, де на їх основі формуються вихідні дані), необхідно забезпечити їх приведення до виду, безпечного для приймаючої сторони. Як правило, це досягається шляхом видалення з них небезпечних елементів (фільтрації) або ж їх перетворення до безпечних еквівалентів (екранування). Санітизацію необхідно реалізовувати адекватно до того місця, в яке потраплять дані в кінцевому підсумку [6].

1.2.5 Небезпечні прямі посилання на об'єкти (Insecure Direct Object References)

Ця вразливість є наслідком того, що доступ до конфіденційних даних користувачів здійснюється не з прав доступу до даного об'єкта, а за будь-яким значенням, наприклад ідентифікатором або унікальним ім'ям. В такому випадку

маючи сторінку виду: «testsite.ru/read_user_message?userName=userName» зловмиснику досить перебирати різні комбінації і підставляючи їх в параметр userName, він зможе читати повідомлення, які належать користувачам.

Пропозиція полягає у використанні хеша для заміни прямого ідентифікатора. Цей хеш солиться зі значенням, визначеним на рівні програми, щоб підтримувати топологію, в якій додаток розгортається в режимі декількох екземплярів. Використання хешу дозволяє наступні властивості:

- не потрібно підтримувати таблицю зіставлення (реальний ідентифікатор проти ідентифікатора інтерфейсу) у сеансі користувача або кеші рівня програми;
- ускладнює створення колекції значень перерахування, оскільки навіть якщо зловмисник може вгадати хеш-алгоритм за розміром ідентифікатора, він не може відтворити значення через сіль, яка не прив'язана до прихованого значення.

1.2.6 Похибки в конфігуруванні (Security Misconfiguration)

Безпечність будь-якої системи завжди залежить від правильного налаштування безпеки, веб-додатки не є виключенням.

Неправильна конфігурація безпеки є найбільш часто проблемою. Зазвичай це результат небезпечних конфігурацій за замовчуванням, неповних або спеціальних конфігурацій, відкритого хмарного сховища, неправильно налаштованих заголовків HTTP та детальних повідомлень про помилки, що містять конфіденційну інформацію.

Налаштування компонентів сервера за замовчуванням найчастіше небезпечні і відкривають можливості до атак. Наприклад, крадіжка сесійного cookie через JavaScript при XSS-атаці стає можливою завдяки виключеною за замовчуванням налаштуванням cookie_http only.

Наступна помилка в конфігурації – це використання даних, для доступу до будь-яких елементів веб-додатків за замовчуванням. Одним з найяскравіших є злом корейського облікового запису facebook за допомогою логін / пароля адміністратора admin / admin.

Висновком цієї атаки є те, що необхідно підтримувати актуальність програмного забезпечення, так як нові вразливості знаходяться кожен день.

1.3 Аналіз методів хешування паролів

Хешування паролів – метод, що дозволяє користувачам запам'ятовувати не дуже довгий ключ, а деякий осмислений вираз, слово чи послідовність символів, що називається паролем. Дійсно, при розробці будь-якого криптоалгоритмами слід враховувати, що в половині випадків кінцевим користувачем системи є людина, а не автоматична система. Це ставить питання про те, зручно, і взагалі чи реально людині запам'ятати 128-бітний ключ. Насправді межа запам'ятовуваності лежить на кордоні 8-12 подібних символів, а, отже, якщо ми будемо примушувати користувача оперувати саме ключем, тим самим ми практично змусимо його до запису ключа на будь-якому листку паперу або електронному носії, наприклад, в текстовому файлі. Це, звичайно, різко знижує захищеність системи. Для вирішення цієї проблеми були розроблені методи, які перетворюють осмислений рядок довільної довжини – пароль, у вказаний ключ заздалегідь заданої довжини. У переважній більшості випадків для цієї операції використовуються так звані хеш-функції. Хеш-функцією в даному випадку називається таке математичне або алгоритмічне перетворення заданого блоку даних, яке володіє наступними властивостями:

- хеш-функція має нескінченну область визначення;
- хеш-функція має кінцеву область значень;
- вона незворотня;
- зміна вхідного потоку інформації на один біт змінює близько половини всіх біт вихідного потоку, тобто результату хеш-функції.

Ці властивості дозволяють подавати на вхід хеш-функції паролі, тобто текстові рядки довільної довжини на будь-якою національною мовою і, обмеживши область значень функції діапазоном $0 \dots 2^N - 1$, де N - довжина ключа в бітах, отримувати на виході досить рівномірно розподілені по області значення блоки інформації – ключі.

Серед методів хешування особливий клас складають криптографічно стійкі, що призначені для застосування в системах захисту інформації: при генерації цифрового підпису, для збереження паролів та ін. Найбільш відомими методами криптографічного хешування є MD (5,6), SHA (1,2,3).

1.3.1 Алгоритм хешування MD5

MD5 (Message Digest 5) - алгоритм хешування, який був розроблений професором Л. Ривестом в 1991 році. Призначений він для створення відбитків, або контрольних сум. MD5 є одностороннім алгоритмом хешування, тобто зворотної розшифровки не має. Вихідний рядок завжди має постійну довжину в 32 символи. Відновлення даних, зашифрованих цим алгоритмом, можливо лише шляхом грубої сили, тобто брутфорсом (береться хеш від передбачуваного тексту, хеш-суми порівнюються, якщо вони не рівні, значить, текст інший).

Зберігати паролі у відкритому тексті в базі даних дуже небезпечно. З метою підвищення безпеки паролів алгоритми MD5 можуть використовуватися для хешування вихідних паролів та хеш-значень, а не простого тексту, які зберігаються в базі даних. Під час автентифікації вхідний пароль також хешується MD5 аналогічним чином, а результат хеш-значення порівнюється зі значенням хешу в базі даних для цього конкретного користувача.

Вибірки повідомлень MD5 широко використовувались у світі програмного забезпечення для гарантування правильної передачі файла. Наприклад, файлові сервери часто надають попередньо підраховану контрольну суму MD5 (відому як md5sum) для файлів, так що користувач може порівняти контрольну суму завантаженого файла з нею. Більшість юніксових операційних систем включають утиліти MD5 у їх дистрибутиви. Користувачі Windows можуть використовувати інтегровану функцію PowerShell "Get-FileHash", інсталиувати утиліту Microsoft або використовувати сторонні програми.

В алгоритмі MD5, як і в його попередника, можлива поява колізій (повторів), тобто складний пароль довжиною в 32 символи, що містить спеціальні символи, цифри, букви різних регістрів, може дати таку ж хеш-суму,

як і, наприклад, п'яти-, шести-символьний простий пароль. Однак ймовірність появи колізій цифрового дайджесту MD5 критично мала.

Щоб уникнути цього, а точніше, щоб звести шанс появи колізії практично до нуля, розробники програмного забезпечення придумали спосіб штучного ускладнення пароля - накладання «солі».

Отже, «сіль» являє собою якийсь набір символів; зазвичай це символи обох регістрів, цифри і спецсимволи, які накладаються або склеюються з самим паролем або з хеш-сумою пароля. На даний момент відомі наступні способи накладення солі: $\text{md5}(\text{md5}(\text{salt}).\text{md5}(\text{pass}))$, $\text{md5}(\text{md5}(\text{pass}).\text{salt})$.

Алгоритм складається з п'яти кроків:

- Append Padding Bits

У вихідний рядок дописують одиничний байт 0x80, а потім дописують нульові біти, до тих пір, поки довжина повідомлення не буде порівнянна з 448 по модулю 512. Тобто дописуємо нулі до тих пір, поки довжина нового повідомлення не буде дорівнює $[довжина] = (512 * N + 448)$,

де N - будь-яке натуральне число, таке, що цей вислів буде найближче до довжини блоку.

- Append Length

Далі в повідомлення дописується 64-бітове представлення довжини вихідного повідомлення.

- Initialize MD Buffer

На цьому кроці ініціюється буфер

word A 01 23 45 67

word B: 89 ab cd ef

word C: fe dc ba 98

word D: 76 54 32 10

Як можна помітити буфер складається з чотирьох констант, призначених для збору хеша.

- Process Message in 16-Word Blocks

На четвертому кроці в першу чергу визначаються 4 допоміжні логічні функції, які перетворюють вхідні 32-бітові слова, в 32-бітові вихідні.

$$F(X, Y, Z) = XY \vee \text{not}(X) Z$$

$$G(X, Y, Z) = XZ \vee Y \text{ not}(Z)$$

$$H(X, Y, Z) = X \text{ xor } Y \text{ xor } Z$$

$$I(X, Y, Z) = Y \text{ xor } (X \vee \text{not}(Z))$$

Також на цьому етапі реалізується так званий «білий шум» - посилення алгоритму, що складається з 64 елементного масиву, що містить псевдовипадкові числа, залежні від синуса числа i :

$$T[i] = 4,294,967,296 * \text{abs}(\sin(i))$$

Кожен 16-бітний блок копіюється в масив $X[16]$:

$$AA = A$$

$$BB = B$$

$$CC = C$$

$$DD = D$$

Потім відбуваються перетворення-раунди, яких всього буде 4. Кожен раунд складається з 16 елементарних перетворень, які в загальному вигляді можна представити у вигляді $[abcd \text{ ksi}]$, яке, в свою чергу, можна уявити як $A = B + ((A + F(B, C, D) + X[k] + T[i]) \lll s)$, де

A, B, C, D - регістри

$F(B, C, D)$ - одна з логічних функцій

$X[k]$ - k -тий елемент 16-бітного блоку.

$T[i]$ - i -тий елемент таблиці «білого шуму»

$\lll s$ - операція циклічного зсуву на s позицій вліво.

В кінці підсумовуються результати обчислень:

$$A = A + AA$$

$$B = B + BB$$

$$C = C + CC$$

$$D = D + DD$$

- Output

Виводячи побайтово буфер ABCD починаючи з А і закінчуючи D отримаємо наш хеш.

Алгоритм MD5 схильний до двох основних типів атак: словникових атак та радужних таблиць. У словникових атаках зловмисник намагається застосувати всі можливі паролі у вичерпному списку, який називається словником.

Зловмисник хешує кожен пароль зі словника та виконує двійковий пошук по скомпрометованих хешованих паролях. Цей метод можна зробити набагато швидше, попередньо обчисливши хеш-значення цих можливих паролів та зберігаючи їх у хеш-таблиці.

Радужні таблиці складаються з хеш-ланцюгів і ефективніші за хеш-таблиці, оскільки вони оптимізують вимоги до зберігання, хоча пошук виконується дещо повільніше. Радужні таблиці відрізняються від хеш-таблиць тим, що вони створюються за допомогою функцій скорочення та хеш-функцій. Функції скорочення перетворюють хеш-значення у відкритий текст. Відкритий текст - це не оригінальний відкритий текст, з якого було створено хеш-значення, а інший. Чергуючи хеш-функцію з функцією скорочення, формуються ланцюжки чергування паролів і хеш-значень. У таблиці зберігаються лише перший (початкова точка ланцюжка) та останній відкритий текст (кінцева точка ланцюжка). Щоб розшифрувати хешований пароль, ми спочатку обробляємо хешований пароль за допомогою функцій зменшення, поки не знайдемо відповідність кінцевій точці ланцюжка. Потім ми беремо відповідну початкову точку цього ланцюжка та регенеруємо хеш-ланцюжок і знаходимо вихідний відкритий текст до хешованого пароля. Звичайно, використання радужних таблиць не гарантує 100% успіху злому систем паролів. Однак, чим більший набір символів, використовуваний для створення таблиці веселки, і чим довша довжина хеш-ланцюжка, тим більшою буде таблиця веселки [7].

1.3.2 Алгоритм хешування SHA1

SHA був розроблений Національним інститутом стандартів і технологій і опублікований як Федеральний стандарт обробки інформації у 1993 р. На вхід SHA-1 може приймати максимум 64 біт і на вихід 160 біт. Це обмеження стало однією із сильних сторін алгоритму, оскільки кількість обмежень введення зменшить колізії, які можуть виникнути при використанні цих алгоритмів.

Алгоритм методу SHA1:

Ініціалізація змінних:

$h_0 = 0x6A09E667h_0 = 0x67452301$

$h_1 = 0xEFCDAB89$

$h_2 = 0x98BADCFE$

$h_3 = 0x10325476$

$h_4 = 0xC3D2E1F0$

Попередня обробка:

Приєднуємо біт "1" до повідомлення

Приєднуємо k бітів "0", де k найменше число ≥ 0 таке, що довжина отриманого повідомлення (в бітах) порівнянна по модулю

$512 \text{ з } 448 \text{ (length mod } 512 == 448)$

Додаємо довжину вихідного повідомлення (до попередньої обробки) як ціле 64-бітове Big-endian число, в бітах.

В процесі повідомлення розбивається послідовно по 512 біт:

for перебираємо всі такі частини

розбиваємо цей шматок на 16 частин, слів по 32-біта $w[i]$, $0 \leq i \leq 15$

16 слів по 32-біта доповнюються до 80 32-бітових слів:

for i from 16 to 79

$w[i] = (w[i-3] \text{ xor } w[i-8] \text{ xor } w[i-14] \text{ xor } w[i-16])$ циклічний зсув 1

Ініціалізація хеш-значень цієї частини:

$a = h_0$

$b = h_1$

$$c = h_2$$

$$d = h_3$$

$$e = h_4$$

Основний цикл:

for i from 0 to 79

if $0 \leq i \leq 19$ then

$$f = (b \text{ and } c) \text{ or } ((\text{not } b) \text{ and } d)$$

$$k = 0x5A827999$$

else if $20 \leq i \leq 39$

$$f = b \text{ xor } c \text{ xor } d$$

$$k = 0x6ED9EBA1$$

else if $40 \leq i \leq 59$

$$f = (b \text{ and } c) \text{ or } (b \text{ and } d) \text{ or } (c \text{ and } d)$$

$$k = 0x8F1BBCDC$$

else if $60 \leq i \leq 79$

$$f = b \text{ xor } c \text{ xor } d$$

$$k = 0xCA62C1D6$$

$$\text{temp} = (a \text{ leftrotate } 5) + f + e + k + w[i]$$

$$e = d$$

$$d = c$$

$$c = b \text{ leftrotate } 30$$

$$b = a$$

$$a = \text{temp}$$

Додаємо хеш-значення цієї частини до результату:

$$h_0 = h_0 + a$$

$$h_1 = h_1 + b$$

$$h_2 = h_2 + c$$

$$h_3 = h_3 + d$$

$$h_4 = h_4 + e$$

Підсумкове хеш-значення:

digest = hash = h0 append h1 append h2 append h3 append h4

Замість оригінального формулювання FIPS PUB 180-1 наведені такі еквівалентні вирази і можуть бути використані на комп'ютері f в головному циклі:

$(0 \leq i \leq 19): f = d \text{ xor } (b \text{ and } (c \text{ xor } d))$ (альтернатива 1)

$(0 \leq i \leq 19): f = (b \text{ and } c) \text{ xor } ((\text{not } b) \text{ and } d)$ (альтернатива 2)

$(0 \leq i \leq 19): f = (b \text{ and } c) + ((\text{not } b) \text{ and } d)$ (альтернатива 3)

$(40 \leq i \leq 59): f = (b \text{ and } c) \text{ or } (d \text{ and } (b \text{ or } c))$ (альтернатива 1)

$(40 \leq i \leq 59): f = (b \text{ and } c) \text{ or } (d \text{ and } (b \text{ xor } c))$ (альтернатива 2)

$(40 \leq i \leq 59): f = (b \text{ and } c) + (d \text{ and } (b \text{ xor } c))$ (альтернатива 3)

Порівняння з MD5

MD5 і SHA-1 є, по суті, поліпшеними версіями MD4.

Схожість:

- чотири етапи;
- кожна дія додається до раніше отриманого результату;
- розмір блоку обробки становить 512 біт;
- обидва алгоритми виконують складання по модулю 2^{32} , вони розраховані на 32-х бітну архітектуру.

Відмінності:

- у SHA-1 на четвертому етапі використовується та ж функція f, що і на другому етапі;
- в MD5 у кожній дії використовується унікальна адитивна константа. У SHA-1 константи використовуються повторно для кожної із чотирьох груп;
- у SHA-1 додана п'ята змінна;
- SHA-1 використовує циклічний код виправлення помилок;
- в MD5 чотири різних елементарних логічних функції, в SHA-1 – три;
- в MD5 довжина дайджесту становить 128 біт, в SHA-1 - 160 біт;
- SHA-1 містить більше раундів (80 замість 64) і виконується на 160-бітному буфері у порівнянні із 128-бітовим буфером MD5. Таким чином, SHA-1

повинен виконуватися приблизно на 25% повільніше, ніж MD5 на тій же апаратурі.

Брюс Шнайер наводить наступний висновок: «SHA-1 - це MD4 із додаванням розширюючого перетворення, додаткового етапу і поліпшеним лавинним ефектом. MD5 - це MD4 із поліпшеним двійковим хешуванням, додатковим етапом і поліпшеним лавинним ефектом.»

Хеш-функції використовуються в системах контролю версій, системах електронного цифрового підпису, а також для побудови кодів аутентифікації.

SHA-1 є найбільш поширеним із усього сімейства SHA і застосовується у різних широко поширених криптографічних додатках і алгоритмах.

Вважається, що SHA-1 не гарантує достатнього захисту проти атак. Вже в 2005 дослідниками були відкриті методи атаки на SHA-1, які поставили під сумнів тривалість використання цього алгоритму. Тому вже з 2010 року низка організацій та компаній стали рекомендувати використання SHA-2 або SHA-3 замість нього Microsoft, Google, Apple та Mozilla оголосили, що їхні веб-браузери припинять приймати SSL сертифікати з SHA-1 починаючи з 2017 року[8].

1.3.3 Scrypt

Scrypt - адаптивна криптографічна функція формування ключа на основі пароля. Функція створена таким чином, щоб ускладнити атаку перебором за допомогою ПЛІС. Для її обчислення потрібний значний обсяг пам'яті з довільним доступом.

Однак перші функції PBKDF (наприклад PBKDF2, розроблена RSA Laboratories) обчислюються порівняно швидко, і їх перебір може бути ефективно реалізований на спеціалізованому обладнанні. Така реалізація дозволяє запускати масштабні паралельні атаки перебору грубою силою, наприклад, з обчисленням сотень значень функції в кожній мікросхемі FPGA.

Функція scrypt розроблялася з метою ускладнення апаратних реалізацій шляхом збільшення кількості ресурсів, необхідних для обчислення. Даний алгоритм використовує значну кількість оперативної пам'яті (пам'яті з довільним

доступом) порівняно з іншими PBKDF. Пам'ять у `scrypt` використовується для зберігання великого вектора псевдовипадкових бітових послідовностей, що генеруються, на початку алгоритму. Після створення вектора його елементи запитуються у псевдовипадковому порядку і комбінуються один з одним для отримання ключа. Так як алгоритм генерації вектора відомий, можлива реалізація `scrypt`, не вимагає пам'яті, і вираховує кожен елемент в момент звернення. Однак, обчислити елемент відносно складно і в процесі роботи функції `scrypt` кожен елемент прочитується багато разів. У `scrypt` закладений такий баланс між пам'яттю і часом, і реалізації, що не використовують пам'ять, надто повільні.

$\text{scrypt}(P, S, N, r, p, \text{dkLen}) = \text{MFCrypt}(P, S, N, p, \text{dkLen})$

де N, r, p — параметри, які визначають складність обчислення функції.

`MFCrypt` визначена так: $\text{DK} = \text{MFCrypt PRF, MF}(P, S, N, p, \text{dkLen})$

Де:

- PRF — псевдовипадкова функція
- $hLen$ — довжина виходу PRF в байтах
- MF (Mixing Function) — послідовна функція, що потребує пам'ять із

довільним доступом

- $MFLen$ — довжина блока, що перемішується у MF. $MFLen = 128 * r$.

Вхідні параметри `scrypt` і `MFCrypt`:

- P - пароль (passphrase) — байтовий рядок.
- S - сіль (salt) — байтовий рядок.
- N - параметр, що задає складність (кількість ітерацій для MF).
- r - параметр, що задає розмір блоку.
- p - ступінь паралельності, ціле число, менше ніж $(232 - 1) * hLen / MFLen$
- $dkLen$ - необхідна довжина вихідного ключа, не більше ніж $(232 - 1) * hLen$

- DK - вихідний ключ

Рекомендовані параметри `scrypt`: $N = 16384, r = 8, p = 1$ (споживання пам'яті — близько 16 МБ)

Швидкість обчислення однієї операції `script` на процесорі загального призначення становить близько 100 мілісекунд при налаштуванні на використання 32 МБ пам'яті. При налаштуванні на тривалість операції в 1 мілісекунду використовується дуже мало пам'яті і алгоритм стає слабшим алгоритму `bscript`, налаштованого на порівнянну швидкість.

Криптовалюта Litecoin використовує такі параметри `script`: $N = 1024$, $r = 1$, $p = 1$, розмір вхідного параметра і солі — 80 байт, розмір ДК — 256 біт (32 байти). Споживання оперативної пам'яті — близько 128 КБ. Обчислення такого `script` на відеокартах приблизно в 10 разів швидше, ніж на процесорах загального призначення, що є ознакою вибору недостатньо сильних параметрів.

1.3.4 Аналіз розглянутих методів

Одна базова вимога будь-якої криптографічної хеш функції та, що має бути неможливо знайти два конкретні повідомлення, які хешують одне значення. MD5 не задовольняє цю вимогу. Такі колізії можуть виявлятися за секунди на звичайному настільному комп'ютері.

На 2020 рік, MD5 все ще широко використовується, незважаючи на свої вразливості.

Вважається, що SHA-1 не гарантує достатнього захисту проти атак. Вже в 2005 дослідниками були відкриті методи атаки на SHA-1, які поставили під сумнів тривалість використання цього алгоритму. Тому вже з 2010 року низка організацій та компаній стали рекомендувати використання SHA-2 або SHA-3 замість нього. Microsoft, Google, Apple та Mozilla оголосили, що їхні веб-браузери припинять приймати SSL сертифікати з SHA-1 починаючи з 2017 року.

Для криптографічних хешей є три додаткових умови, які відрізняють їх від всіх інших:

- незворотність: для заданого значення хеш-функції m повинно бути обчислювально-нездійсненним знайти блок даних X , для якого $H(X) = m$;

- стійкість до колізій першого роду: для заданого повідомлення M має бути обчислювально нездійсненним підібрати інше повідомлення N , для якого $H(N) = H(M)$;

- стійкість до колізій другого роду: має бути обчислювально-нездійсненним підібрати пару повідомлень $\sim (M, M')$, що мають однаковий хеш.

Нижче перераховані вимоги, яким хеш в базі повинен задовольняти:

- стійкість до атак перебору (прямий перебір і перебір по словнику);

- неможливість пошуку однакових паролів різних користувачів по хешам.

Для виконання першої вимоги потрібно використовувати стійкі в даний час хеш-функції.

Для виконання другої - до паролю перед хешуванням додається випадковий рядок (сіль). Таким чином, у двох користувачів з паролем «123456» будуть різні солі «сіль1» і «сіль2», а відповідно і хеш-функції від «123456сіль1» і «123456сіль2» в базі теж будуть різні.

Тепер трохи про систему зберігання - і сіль і сам хеш зберігаються в базі даних. Тобто отримавши доступ до СУБД, зловмисник отримує і значення хешів і солі. Щоб ускладнити життя при атаці перебору слід дописати сіль до паролю, а не навпаки.

Так як хеш-функція, як правило, обчислюється послідовно по рядку (вимоги поточності алгоритму), то зловмиснику при переборі «солоних» хешів, буде простіше, коли підхешовий вираз починається з солі.

Простіше тому, що він (зловмисник) може перелічити заздалегідь хеш (сіль) і далі обчислювати хеш (сіль) + хеш (пароль) вже куди швидше (практично з тією ж швидкістю, що і просто хеш (пароль)). Для всіх паролів, що він буде перебирати.

Для того щоб ще ускладнити життя атакуючому, запропоновується ввести локальний параметр.

Це по суті «друга сіль» дописується до всіх (паролів + сіль) конструкцій, і є однаковою для всіх хешів в базі. Перевага у тому, що локального параметра в

базі немає. Це константа системи, яка зберігається в пам'яті програми, куди вона потрапляє з налаштування (будь-яким способом, тільки не з бази).

Дуже проста і дієва міра, яка дозволяє практично повністю виключити атаку перебору за даними тільки одного сховища хешів (без знання локального параметра).

Компанія ONsec у 2019 році провела дослід, прорахувавши швидкість перебору хешів (одиниці виміру - мегахеші в секунду, тобто кількість):

MD5: 16000 M/s

SHA-1: 5900 M/s

SHA256: 2050 M/s

SHA512: 220 M/s

NTLM: 28400 M/s

bcrypt: 8,5 k/s

Локальний параметр (його ще називають перцем). Перець - такий глобальний випадковий рядок, який дописується до всіх паролів (крім солі). Він секретний (на відміну від солі). Таким чином, отримавши базу, дізнатися паролі стає неможливо. Перець глобальний, по своїй суті. Якщо ви захочете його змінити, наприклад, у разі атаки, то вам доведеться змінити всі паролі.

Не існує жодної реалізації відомих перевірених алгоритмів, які б брали локальний параметр як аргумент.

Засновані на паролі функції формування ключа (password-based key derivation function, PBKDF) зазвичай розробляються таким чином, щоб вимагати більшого часу обчислення (по порядку величини - сотні мілісекунд). При використанні легальним користувачем потрібно обчислити подібну функцію один раз (наприклад при автентифікації) і такий час допустимий. Але при проведенні атаки повного перебору атакувачу потрібно зробити мільярди обчислень функції і її обчислювальна складність робить атаку повільнішою і більш дорогою. Швидкість обчислення однієї операції методу scrypt на процесорі загального призначення становить близько 100 мілісекунд при

налаштуванні на використання 32 МБ пам'яті. SCrypt - кращий вибір сьогодні, і використовується в цій області вже 10 років.

1.4 Аналіз методів шифрування бази даних

Шифрування бази даних — використання технології шифрування для перетворення інформації, що зберігається в базі даних (БД), в шифротекст, що робить її прочитання неможливим для осіб, що не володіють ключами шифрування.

Прозоре шифрування бази даних — технологія, для шифрування і дешифрування вводу-виводу файлів БД. Дані шифруються перед записом на диск і дешифруються під час читання в пам'ять, що вирішує проблему захисту «неактивних» даних, але не забезпечує збереження інформації при передачі по каналах зв'язку або під час використання. Перевагою TDE є те, що шифрування і дешифрування виконуються прозоро для додатків, тобто їх модифікація не потрібна.

TDE застосовується для файлів БД і журналу транзакцій на рівні сторінок. Сторінки шифруються за допомогою спеціального симетричного ключа шифрування бази даних, захищеного сертифікатом, який зберігається в БД master і шифрується її головним ключем, або асиметричним ключем, захищеним модулем розширеного керування ключами.

Шифрування на рівні стовпців на відміну від TDE, що шифрує БД повністю в реалізації Microsoft і окремі стовпці, але з однаковим ключем для всієї таблиці в реалізації Oracle, цей метод дозволяє шифрувати окремі стовпці з різними ключами, що забезпечує додаткову гнучкість при захисті даних. Ключі можуть бути призначені користувачам і захищені паролем для запобігання автоматичної розшифровки, однак це ускладнює адміністрування БД.

Важливо зазначити, що традиційні методи шифрування баз даних зазвичай шифрують і дешифрують вміст БД, адміністрування якої забезпечується системою керування базами даних, що працює поверх операційної системи[6]. Це зменшує захищеність інформації, так як зашифрована БД може бути

запущена на відкритій або потенційно вразливій операційній системі. Наприклад, Microsoft використовує технологію шифрування файлової системи, яка забезпечує шифрування на рівні файлів. Кожен об'єкт шифрується за допомогою унікального ключа шифрування файлів, захищеного сертифікатом користувача. Цей сертифікат може бути складовим, що дає можливість отримати доступ до файлу більше ніж одному користувачеві. Через розширення сфери шифрування, використання EFS може знизити продуктивність і ускладнити адміністрування, так як системному адміністратору потрібно мати доступ до операційної системи для використання EFS.

Існує два основних способи шифрування інформації: симетричний та асиметричний. Головним принципом у них є те, що передавач і приймач заздалегідь знають алгоритм шифрування і ключ до повідомлення, без знання яких інформація являє собою безглуздий набір символів.

- Симетричне шифрування. Є найстарішим і найвідомішим методом. У контексті баз даних він включає в себе закритий ключ, який застосовується для шифрування та дешифрування інформації, що зберігається в БД і викликається з неї. Цей параметр змінює дані таким чином, що їх прочитання без розшифровки стає неможливим. Явним недоліком цього методу є те, що може статися витік конфіденційної інформації, якщо ключ виявиться у осіб, які не повинні мати доступ до даних. Однак використання лише одного ключа в процесі шифрування дає перевагу у вигляді швидкості і простоти застосування даної технології.

- Асиметричне шифрування. Проблема потрапляння секретного ключа в чужі руки при передачі по каналах зв'язку, яку має шифрування з закритим ключем, вирішена в методі асиметричного шифрування (шифруванні з відкритим ключем), в якому є два пов'язаних між собою ключа — це пара ключів. Відкритий ключ відомий всім і може передаватися по незахищеному каналу зв'язку. У той час як другий, закритий ключ зберігається в таємниці і є унікальним для кожного користувача. Асиметричне шифрування є більш безпечним, у порівнянні з симетричним, але в той же час воно істотно повільніше.

2 АНАЛІЗ ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ТА МЕТОДІВ РОЗРОБКИ

2.1 ASP.NET Core

ASP.NET Core є багатоплатформним, високопродуктивним середовищем з відкритим вихідним кодом для створення сучасних хмарних додатків, підключених до Інтернету. ASP.NET Core дозволяє виконувати наступні завдання:

- створювати веб-додатки та служби, додатки Інтернету речей (IoT) і серверні частини для мобільних додатків;
- використовувати вибрані засоби розробки в Windows, macOS і Linux;
- виконувати розгортання в хмарі або локальному середовищі;
- запускати в .NET Core.

Завдяки модульності фреймворка всі необхідні компоненти веб-додатку можуть завантажуватися як окремі модулі через пакетний менеджер NuGet. Крім того, на відміну від попередніх версій платформи немає необхідності використовувати бібліотеку System.Web.dll.

ASP.NET Core включає в себе фреймворк MVC, який об'єднує функціональність MVC, Web API і Web Pages. У попередніх версіях платформи дані технології реалізувалися окремо і тому містили багато дублюючої функціональності. Зараз же вони об'єднані в одну програмну модель ASP.NET Core MVC. А Web Forms повністю пішли в минуле [9].

Крім об'єднання вищезазначених технологій в одну модель в MVC був доданий ряд додаткових функцій.

Однією з таких функцій є тег-хелпери (tag helper), які дозволяють більш органічно поєднувати синтаксис HTML з кодом C #.

ASP.NET Core характеризується розширюваністю. Фреймворк побудований з набору незалежних компонентів. І допустимо або використовувати вбудовану реалізацію цих компонентів, або розширити їх за допомогою механізму спадкування, або зовсім створити і застосовувати свої компоненти зі своїм функціоналом.

Також було спрощено управління залежностями і конфігурацією проекту. Фреймворк тепер має свій легкий контейнер для впровадження залежностей, і більше немає необхідності застосовувати сторонні контейнери, такі як Autofac, Ninject. Хоча при бажанні їх також можна продовжувати використовувати.

В якості інструментарію розробки можна використовувати останні випуски Visual Studio, починаючи з версії Visual Studio 2015. Крім того, можна створювати додатки в середовищі Visual Studio Code, яка є крос-платформною та може працювати як на Windows, так і на Mac OS X і Linux.

Для обробки запитів тепер використовується новий конвеєр HTTP, який заснований на компонентах Katana і специфікації OWIN. А його модульність дозволяє легко додати свої власні компоненти.

Якщо підсумувати, то можна виділити наступні ключові відмінності ASP.NET Core від попередніх версій ASP.NET:

- новий легкий і модульний конвеєр HTTP-запитів;
- можливість розгортати додаток як на IIS, так і в рамках свого власного процесу;
- використання платформи .NET Core і її функціональності;
- поширення пакетів платформи через NuGet;
- інтегрована підтримка для створення та використання пакетів NuGet;
- єдиний стек веб-розробки, що поєднує Web UI і Web API;
- конфігурація для спрощеного використання в хмарі;
- вбудована підтримка для впровадження залежностей;
- можливість розширення;
- розвиток як open source, відкритість до змін.

2.2 Entity Framework

Entity Framework являє собою спеціальну об'єктно-орієнтовану технологію на базі фреймворка .NET для роботи з даними. Якщо традиційні засоби ADO.NET дозволяють створювати підключення, команди та інші об'єкти для взаємодії з базами даних, то Entity Framework являє собою більш високий рівень

абстракції, який дозволяє абстрагуватися від самої бази даних і працювати з даними незалежно від типу сховища. Якщо на фізичному рівні ми оперуємо таблицями, індексами, первинними і зовнішніми ключами, але на концептуальному рівні, який нам пропонує Entity Framework, ми вже працюємо з об'єктами.

Перша версія Entity Framework - 1.0 вийшла ще в 2008 році і представляла дуже обмежену функціональність, базову підтримку ORM (object-relational mapping - відображення даних на реальні об'єкти) і один єдиний підхід до взаємодії з БД - Database First. З виходом версії 4.0 у 2010 році багато чого змінилося - з цього часу Entity Framework став рекомендованою технологією для доступу до даних, а в сам фреймворк були введені нові можливості взаємодії з БД - підходи Model First і Code First.

Додаткові поліпшення функціоналу пішли з виходом версії 5.0 в 2012 році. І нарешті, в 2013 році був випущений Entity Framework 6.0, що володіє можливістю асинхронного доступу до даних.

Центральною концепцією Entity Framework є поняття сутності або entity. Сутність представляє набір даних, асоційованих з певним об'єктом. Тому дана технологія передбачає роботу не з таблицями, а з об'єктами і їх наборами.

Будь-яка сутність, як і будь-який об'єкт з реального світу, має низку властивостей. Наприклад, якщо сутність описує людину, то ми можемо виділити такі властивості, як ім'я, прізвище, зріст, вік, вага. Властивості необов'язково представляють прості дані типу int, а й можуть представляти більш комплексні структури даних. І у кожної сутності може бути одна або кілька властивостей, які будуть відрізняти цю сутність від інших і будуть унікально визначати цю сутність. Подібні властивості називають ключами.

При цьому сутності можуть бути пов'язані асоціативною зв'язком один-к-багатьом, один-к-одному і багато-до-багатьох, подібно до того, як в реальній базі даних відбувається зв'язок через зовнішні ключі.

Відмінною рисою Entity Framework є використання запитів LINQ для вибірки даних з БД. За допомогою LINQ є можливість не тільки отримувати

певні рядки, що зберігають об'єкти, з БД, а й отримувати об'єкти, пов'язані різними асоціативними зв'язками.

Іншим ключовим поняттям є Entity Data Model. Ця модель зіставляє класи сутностей з реальними таблицями в БД.

Entity Data Model складається з трьох рівнів: концептуального, рівень сховища і рівень зіставлення (маппінга).

На концептуальному рівні відбувається визначення класів сутностей, які використовуються в додатку.

Рівень сховища визначає таблиці, стовпці, відносини між таблицями і типи даних, з якими порівнюється використовувана база даних.

Рівень зіставлення (маппінга) служить посередником між попередніми двома, визначаючи зіставлення між властивостями класу суті і стовпцями таблиць.

Таким чином, ми можемо через класи, визначені у додатку, взаємодіяти з таблицями з бази даних.

Способи взаємодії з БД:

Entity Framework передбачає три можливі способи взаємодії з базою даних:

- Database first: Entity Framework створює набір класів, які відображають модель конкретної бази даних;

- Model first: спочатку розробник створює модель бази даних, по якій потім Entity Framework створює реальну базу даних на сервері;

- Code first: розробник створює клас моделі даних, які будуть зберігатися в БД, а потім Entity Framework за цією моделлю генерує базу даних і її таблиці.

2.3 Аналіз і вибір технологій для проектування бази даних

База даних – це сукупність певним чином організованих даних, що зберігаються в запам'ятовуючих пристроях ЕОМ. Зазвичай дані зберігаються на жорсткому диску сервера організації.

У загальному випадку дані в базі даних (в великих системах) є інтегрованими і розділяються. Ці два аспекти, інтеграція і поділ даних, являють собою найбільш важливі переваги використання баз даних на «великому» обладнанні і, щонайменше, один з них – інтеграція – є перевагою їх застосування і на «малому» обладнанні.

Під поняттям інтеграції даних мається на увазі можливість представити БД як об'єднання декількох окремих файлів даних, повністю або частково виключаючи надмірність зберігання інформації.

Під поняттям розділу даних мається на увазі можливість використання декількома різними користувачами окремих елементів, що зберігаються в базі даних. Мається на увазі, що кожен з користувачів зможе отримати доступ до одних і тих же даних, можливо, навіть одночасно (паралельний доступ). Такий поділ даних, з паралельним або послідовним доступом, частково є наслідком того факту, що БД має інтегровану структуру.

Одним із наслідків згаданих вище характеристик бази даних (інтеграції та роздільності) є те, що кожен конкретний користувач зазвичай має справу лише з невеликою частиною всієї бази даних, причому в опрацюванні різними користувачами частини можуть довільним чином перекриватися. Інакше кажучи, кожна БД сприймається її різними користувачами по-різному. Фактично, навіть ті два користувача бази даних, які працюють з одними і тими ж її частинами, можуть мати значно різні уявлення про них.

База даних характеризується моделлю даних, тобто формою організації даних в ній. За типом моделі даних БД діляться на мережеві, ієрархічні і реляційні. В даний час практично застосовується тільки реляційна структура, в якій база даних складається з однієї або декількох двовимірних таблиць. Кожна реляційна таблиця являє собою двовимірний масив і має такі властивості:

- кожен елемент таблиці - один елемент даних;
- всі осередки в стовпці таблиці однорідні, тобто всі елементи в стовпці мають однаковий тип (числовий, символьний та інше);
- кожен стовпець має унікальне ім'я;

- однакові рядки в таблиці відсутні;
- порядок проходження рядків і стовпців може бути довільним.

Така модель зберігання даних забезпечує зручність використання бази даних на ЕОМ. З огляду на те, що таблиці бази даних можуть бути пов'язані певними відносинами, така модель забезпечує цілісність даних і відсутність надмірності зберігання. Тому вона і використовується в більшості сучасних БД.

Система управління базами даних (СУБД) - спеціалізована програма або комплекс програм, призначена для організації і ведення бази даних. Вона забезпечує зберігання даних і взаємодію користувача з БД, дозволяючи користувачам здійснювати пошук, сортування та вибірку інформації в базі даних, а деяким користувачам - додавати, видаляти і змінювати записи в БД.

Залежно від того, з якою базою даних працює СУБД, вона може бути:

- ієрархічною;
- мережевою;
- реляційною;
- об'єктно-реляційною;
- об'єктно-орієнтованою.

В даний час реляційні СУБД є важливим інструментом в багатьох областях, починаючи з традиційних: бізнес, наукове дослідження, освіта і закінчуючи розробкою пошукових серверів в Internet [11].

Вибір СУБД являє собою складну багатопараметричну задачу і є одним з важливих етапів при розробці додатків. Обраний програмний продукт повинен задовольняти як поточним, так і майбутнім потребам підприємства, при цьому слід враховувати фінансові витрати на придбання необхідного обладнання, самої системи, розробку необхідного програмного забезпечення на її основі, а також навчання персоналу. Крім того, необхідно переконатися, що нова СУБД здатна принести підприємству реальні вигоди.

Очевидно, найбільш простий підхід при виборі СУБД заснований на оцінці того, якою мірою існуючі системи задовольняють основним вимогам створюваного проекту інформаційної системи.

Основні критерії вибору СУБД:

- передбачені типи даних;
- реалізація мови запитів;
- мобільність;
- розподіленість;
- засоби проектування;
- підтримувані мови програмування;
- можливість використання в розробці веб-додатків;
- стабільність виробника;
- поширеність СУБД.

На даний час існує декілька найбільш популярних СУБД для проектування веб-сервісу. На рис. 2.1 зображено рейтинг популярності СУБД у світі.

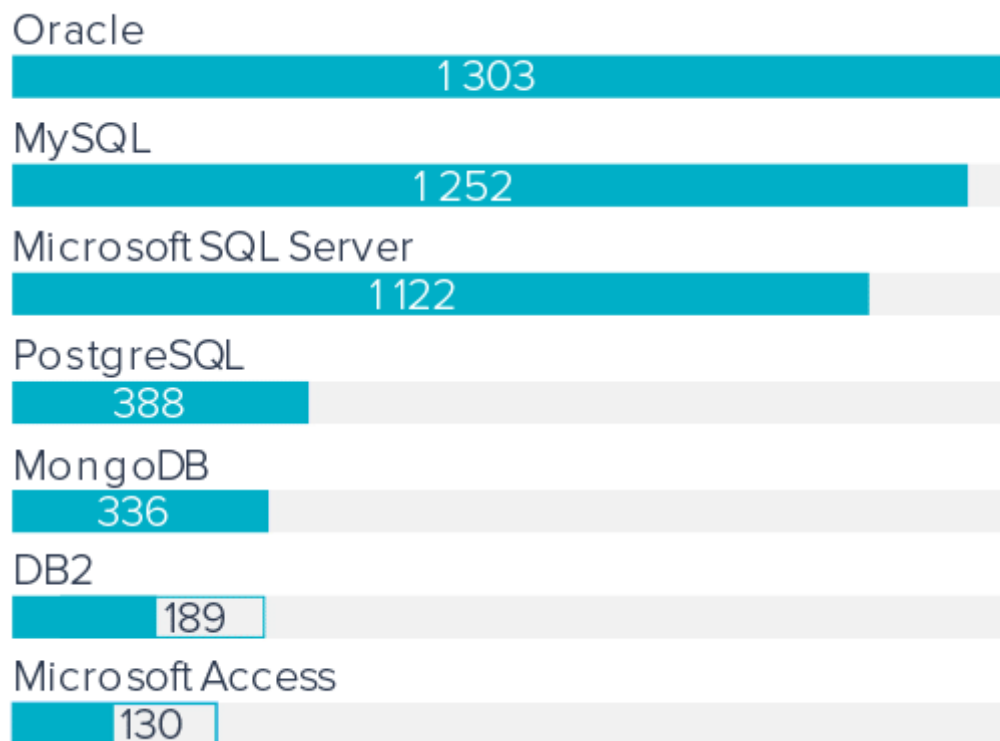


Рисунок 2.1 – Рейтинг популярності СУБД у світі

Перше місце в рейтингу займає компанія Oracle. Ця СУБД призначена для хмарних середовищ і може бути розміщена на одному або декількох серверах, це

дозволяє управляти базами даних, які містять мільярди записів. Крім того, безпека доведена до найвищого рівня, тому що кожна транзакція ізольована від інших [12]. Але існує ряд недоліків. По-перше висока ціна на продукт: розрахована на одного користувача ліцензія коштує 350 доларів, процесорна – близько 17 тисяч. По-друге високі вимоги до системних ресурсів. По-третє вона має складні конфігурації, а отже не кожен спеціаліст зможе працювати з нею.

MySQL – одна з найпоширеніших систем управління базами даних. Вона використовується, в першу чергу, для створення динамічних веб-сторінок, оскільки має чудову підтримку з боку різноманітних мов програмування [13]. Характеризується високою швидкістю, стійкістю і простотою використання, а також наявністю простої і ефективної системи безпеки. До того ж виділяється підтримкою різних типів таблиць, економним споживанням ресурсів та можливістю синхронізації з іншими базами даних. Ті недоліки, які наразі існують (менша безпечність для великих систем, платна технічна підтримка), не впливають на якість розробки та можуть бути проігнорованими [14].

Оскільки MySQL повністю задовольняє поставленим вимогам, в якості СУБД було обрано саме її.

HeidiSQL - багатофункціональний додаток з відкритим вихідним кодом для веб-розробників, що використовують одну з популярних технологій: MySQL, Microsoft SQL або PostgreSQL. Вона дозволяє переглядати і редагувати дані, створювати і змінювати таблиці, уявлення, процедури, тригери і події в розкладі, взаємодіяти з одним або декількома серверами. Вона також надає можливість експортувати структуру і дані в SQL-файли, буфер обміну або переносити на інші сервера [15].

Саме тому HeidiSQL чудово підходить для роботи з сервером та адміністрування БД.

2.4 Способи зберігання файлів

При виборі способу зберігання файлів користувачі необхідно врахувати деякі параметри. По-перше необхідно враховувати тип файлів (від цього напряду залежить об'єм зберігаємих файлів). Оскільки створюється веб-сервіс хмарного зберігання файлів, в якому не буде обмежень на тип файлу, можна припустити що користувачі зберігатимуть у хмарі важкі відео-файли, архіви з великою кількістю файлів. Якщо зберігати усю цю кількість файлів у базі даних рано чи пізно прийде необхідність масштабування бази даних.

Найчастіше можна легко масштабувати сервери додатків горизонтально: поставити десять або двадцять серверів додатків. Але сервер бази даних горизонтально масштабується дуже погано. Його можна масштабувати тільки вертикально, нарощуючи потужність, додаючи пам'ять, встановивши більш потужний процесор. Рано чи пізно масштабування перестане допомагати. Після цього або починається експоненціальне зростання ціни питання, або виявиться, що досить потужного устаткування для вирішення задачі в природі ще не існує.

Таким чином, якщо система буде продовжувати писати все в базу, то в кінці кінців це закінчиться тим, що саме база стане найслабшою ланкою. А коли база стає слабкою ланкою, це - дуже погано [16].

Ще однією проблемою зберігання файлів у базі даних є процес резервного копіювання. Чим більше база даних, тим складніше її копіювати. Зрозуміло, що можна виділити якісь окремі табличні простору, копіювати частково, але все одно - це складно. І коли ми опиняємося перед фактом, що у нас є табличка, і в ній, припустимо, 100 млн рядків з BLOB по 200000 кожен, то копіювати її доведеться всю. Втім, навіть якщо є partition, періодично копіювати таблицю повністю все одно доведеться. І коли ця табличка «доросте» до пари десятків терабайт, її копіювання триватиме дуже довго.

Враховуючи ці недоліки можна дійти висновку, що зберігання файлів у базі даних не є цілком доречним. Виходом із цієї ситуації може бути винесення з бази даних все, що можна з неї винести.

Такий спосіб значно полегшить процес зберігання файлів, оскільки масштабувати базу буде непотрібно, необхідно лише періодично додавати об'єм пам'яті. Проте, такий спосіб теж не є ідеальним.

Поки було тисячі файлів або 10 000 файлів, проблем не виникне. Файли лежать в одній папці файлової системи, з ними можна було легко працювати. Але коли файлів стає більше, дає про себе знати проблема файлових систем - система починає працювати дуже повільно, коли в одній папці багато файлів.

Очевидне рішення - створювати папки. Створюємо ієрархію папок, шлях до файлів зберігаємо в полі БД. Одна з проблем методу - організація процесу створення папок і розподілу файлів по ним. Кожний новий файл кладеться до нової папки (можливо навіть раніше створеною).

Виявилося, що це - погана ідея, так як виникають складнощі резервного копіювання. Для того щоб копіювати таку структуру, потрібно визначити, які файли змінилися, а для цього необхідно пробігти по всій структурі каталогів. Причому кількість папок може бути досить великим, тому що якщо файлів у мільйон, то розклавши їх в сто папок, ми отримаємо по 10 000 файлів у кожній.

Залежно від обсягу даних і кількості користувачів можна застосовувати різні стратегії створення папок. Найбільш ефективною виявилася та, яка має назву "стратегія Daily». Щодня створюється нова папка. У ній, при необхідності, можна створити ряд підпапок, але їх буде не дуже багато. Навіть якщо за день з'явиться 100 000 файлів, досить буде створити всього 100 папок, щоб в кожній з них було по 1000. Таким чином, можна легко копіювати цю одноденну папку. Навіть якщо виникне необхідність копіювати частіше, досить створювати резервну копію тільки однієї цієї папки, а всі інші вже не змінюються, вони в архіві.

Переваги методу:

- нам не потрібно заздалегідь створювати всі папки. Ми просто можемо щоночі, наприклад, о 12 годині, створити папку на наступний день і протягом дня з нею працювати;

- дуже легко проводити резервне копіювання.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка веб-інтерфейсу

Одна із заporук вдалого веб-сервісу це вдалий дизайн. Тому потрібно прикласти багато зусиль на створення вдалої концепції дизайну, дизайн-макету.

Мінімалістичний та плаский дизайн, зрозумілі та зручні елементи навігації, використання невеликої кількості кольорів і шрифтів все це є концепцією стилю створюваного веб-сервісу.

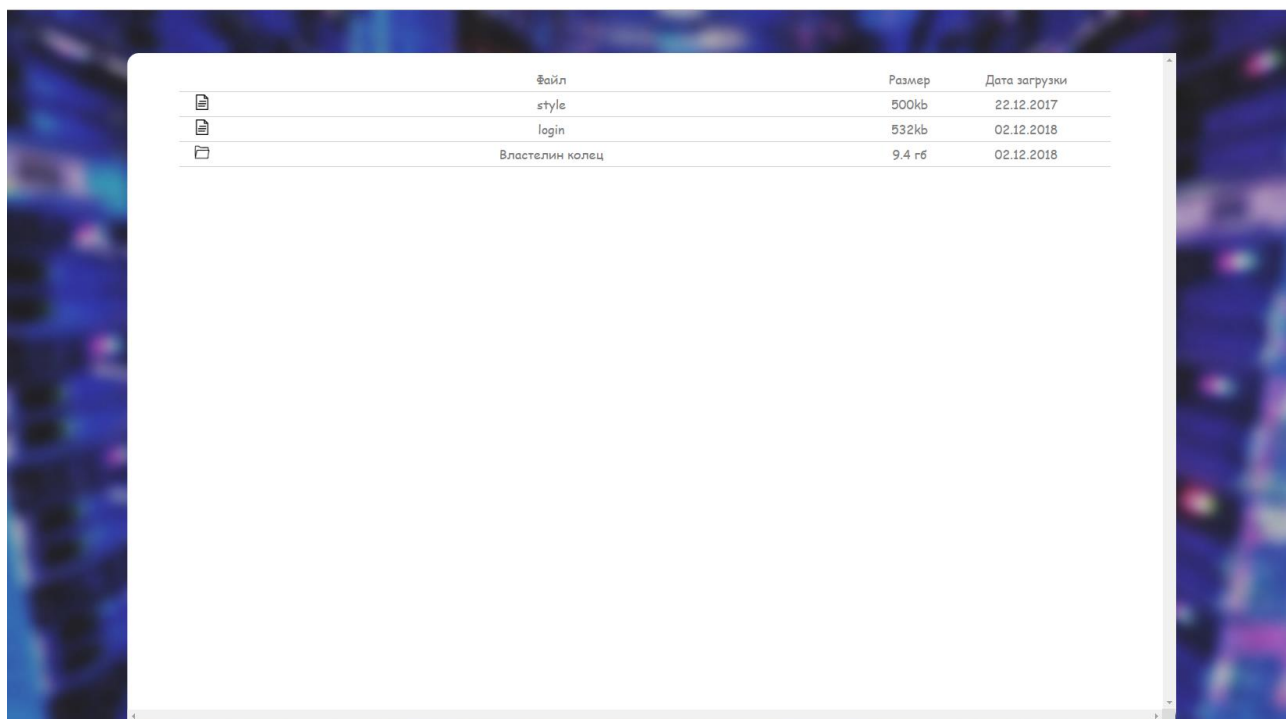
Сайт складається зі сторінок, які у свою чергу містять у собі елементи навігації, візуальні ефекти та інше. Від якості візуалізації та функціональності ресурсу залежить його відвідуваність і популярність.

На головній сторінці веб-сервісу повинні бути розташовані елементи входу/реєстрації на веб-сервіс, елементи завантаження та вивантаження файлів, як зображено на рис.3.1.



Рисунок 3.1 – Головна сторінка веб-сервісу

Нижче, на рис. 3.2, розташована таблиця з завантаженими користувачем файлами, в якій він має змогу зручно переглядати їх назву, розмір, дату завантаження.



The screenshot shows a web application window with a table of uploaded files. The table has three columns: 'Файл' (File), 'Размер' (Size), and 'Дата загрузки' (Upload Date). There are three rows of data. The first two rows represent files ('style' and 'login'), and the third row represents a folder ('Властелин колец'). Each row has a small icon to its left: a document icon for files and a folder icon for the folder.

Файл	Размер	Дата загрузки
style	500kb	22.12.2017
login	532kb	02.12.2018
Властелин колец	9.4 гб	02.12.2018

Рисунок 3.2 – Таблица з завантаженими файлами

Також перехід до таблиці можливий по натисненню на внутрішнє посилання, яке розташоване біля кнопки «Вход и регистрация», як зображено на рис.3.3.

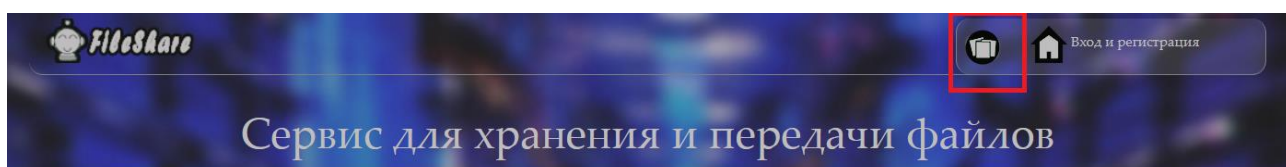


Рисунок 3.3 – Внутрішнє посилання на таблицю файлів

По натисненню кнопки «Вход и регистрация» відкриється нова сторінка, як зображено на рис.3.4. Користувач повинен ввести свій унікальний логін та пароль, далі натиснути кнопку «Вход», у разі вірно введених даних користувач

отримає доступ до усього функціоналу веб-сервісу, у разі помилкових даних користувач отримає повідомлення (рис. 3.5).

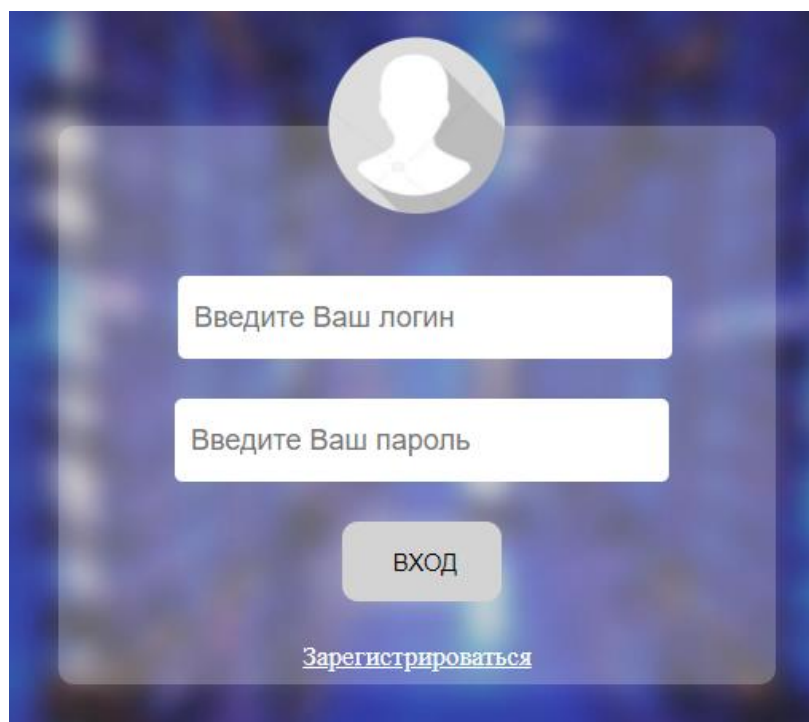


Рисунок 3.4 – Сторінка входу

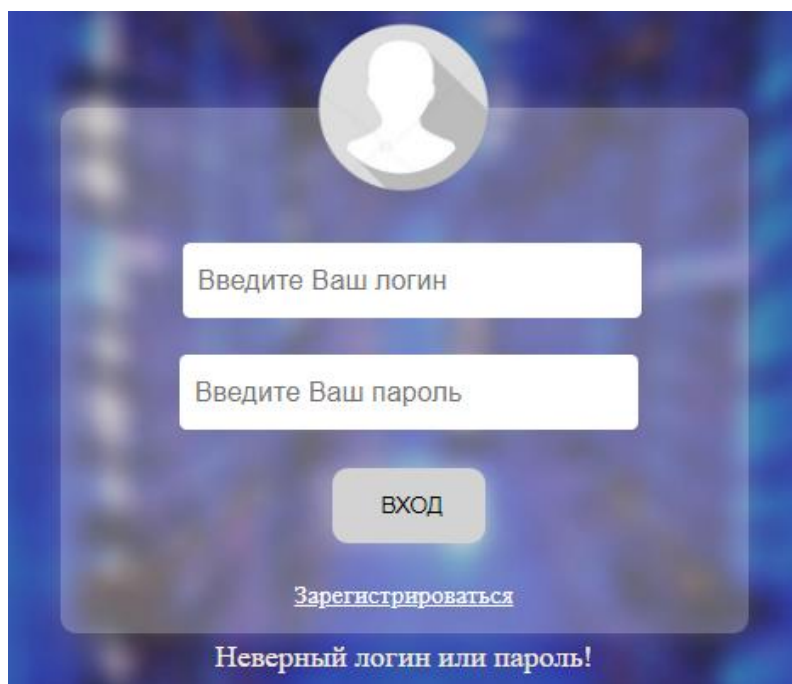


Рисунок 3.5 – Повідомлення про помилку входу

Якщо користувач не зареєстрований, він при натисненні на кнопку «Зареєструватися» отримає таку можливість. Для реєстрації необхідно вказати адресу електронної пошти, логін, пароль та підтвердження паролю, як зображено на рис.3.6.

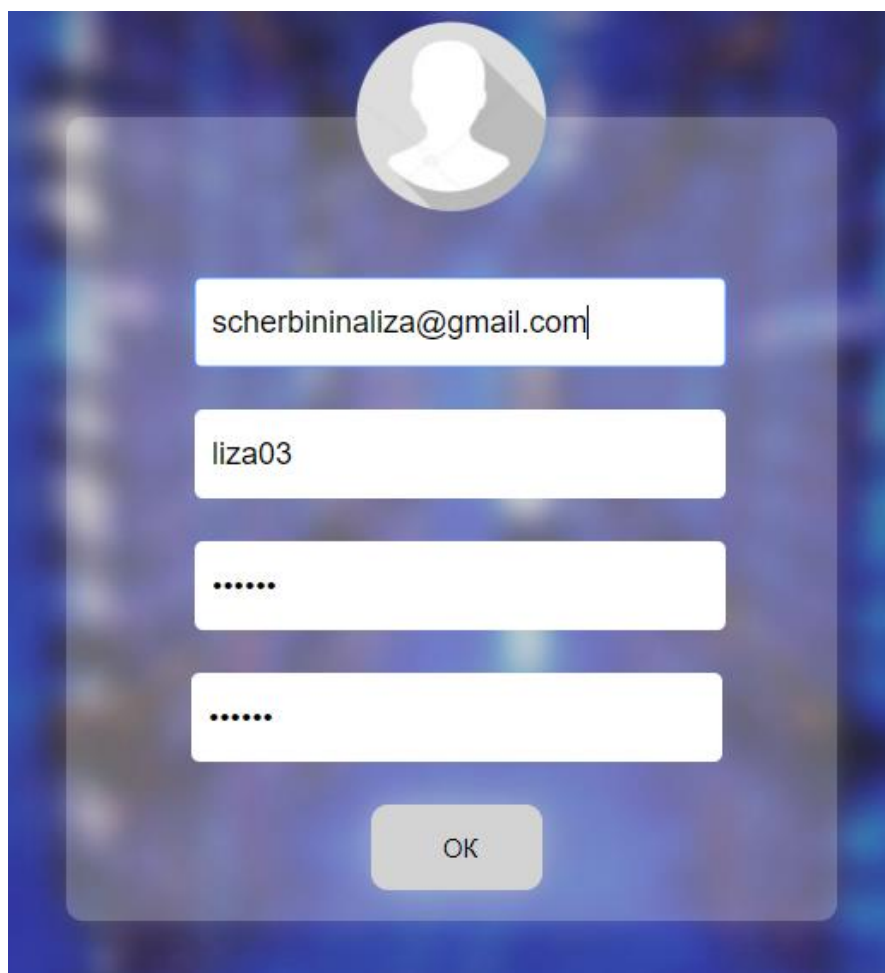
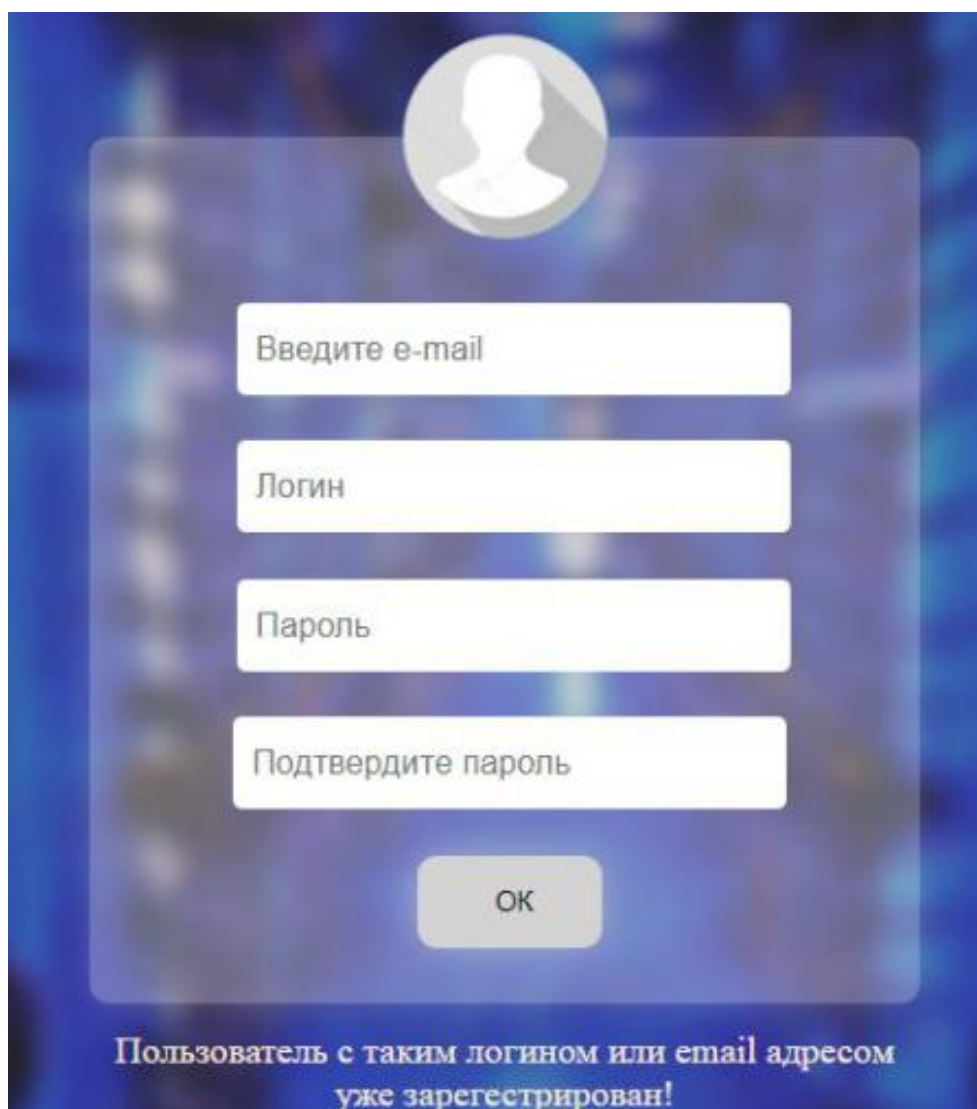
The image shows a registration form on a blue background. At the top center is a circular icon of a person's silhouette. Below it are four white input fields stacked vertically. The first field contains the email address 'scherbininaliza@gmail.com'. The second field contains the login 'liza03'. The third and fourth fields contain six dots, representing a password and its confirmation. At the bottom center of the form is a grey button with the text 'OK'.

Рисунок 3.6 – Сторінка реєстрації

В системі передбачено декілька помилкових ситуацій, в разі яких користувач отримає відповідне повідомлення. Це такі ситуації як:

- користувач вводить адресу електронної пошти або логін, які вже зареєстровані в системі (рис. 3.7);
- користувач допустився помилки при заповненні полів вводу паролю та підтвердження паролю (рис. 3.8).



The image shows a registration form with a blue background. At the top center is a circular icon containing a white silhouette of a person. Below this icon are four white input fields with rounded corners, each containing a label in Russian: "Введите e-mail", "Логин", "Пароль", and "Подтвердите пароль". Below the input fields is a grey button with the text "ОК". At the bottom of the form, there is a red error message in Russian: "Пользователь с таким логином или email адресом уже зарегистрирован!".

Введите e-mail

Логин

Пароль

Подтвердите пароль

ОК

Пользователь с таким логином или email адресом
уже зарегистрирован!

Рисунок 3.7 – Повідомлення про наявність зареєстрованих даних

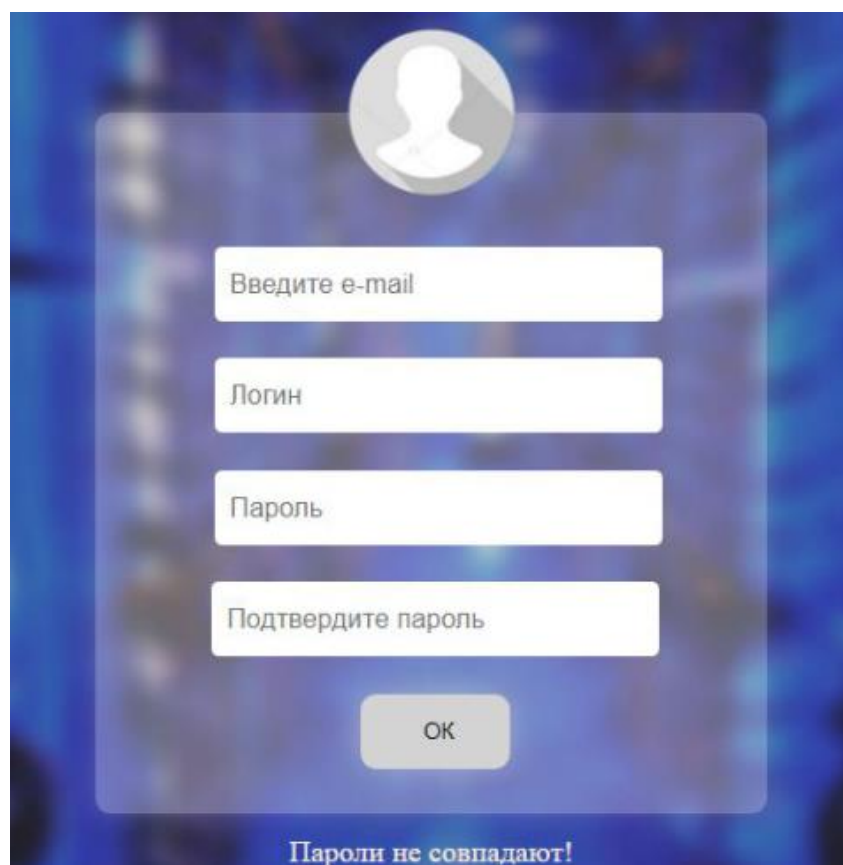


Рисунок 3.8 – Повідомлення про помилку вводу паролів

3.2 Розробка бази даних

База даних (БД) є організованою структурою, призначеною для зберігання інформації. Сучасні БД дозволяють розміщувати у своїх структурах не тільки дані, але і методи (тобто програмний код), за допомогою яких відбувається взаємодія з споживачем або іншими програмно-апаратними комплексами.

Комплекс програмних засобів, призначених для створення структури нової бази, це системи управління базами даних, які наповнюють, редагують її вміст і візуалізують інформацію. Під візуалізацією інформації бази розуміється відбір відображуваних даних відповідно до заданого критерію, їхнє упорядкування, оформлення і наступна видача на пристрій виводу або передачі по каналах зв'язку.

Існує багато систем управління базами даних. Вони можуть по-різному працювати з різними об'єктами і надають користувачу різні функції й засоби. Більшість СУБД спираються на єдиний усталений комплекс основних понять.

Для зберігання інформації, необхідної для роботи програми, була створена база даних «fileshare_bd» до складу якої входять 3 таблиці: «File», «Type», «User».

Таблиця «File», структура якої складається з 8 стовпців, є основною і зберігає в собі таку інформацію: унікальний номер файлу, номер користувача- власника, назва файлу, шлях до файлу на сервері, посилання на файл, тип файлу, дата завантаження, розмір файлу.

Таблиця «User», структура якої складається з 4 стовпців, зберігає в собі таку інформацію: унікальний номер користувача, e-mail, логін, пароль.

Таблиця «Type», структура якої містить 2 стовпці, зберігає в собі таку інформацію: унікальний номер типу файлу, тип файлу.

Схему створеної бази даних зображено на рис. 3.9

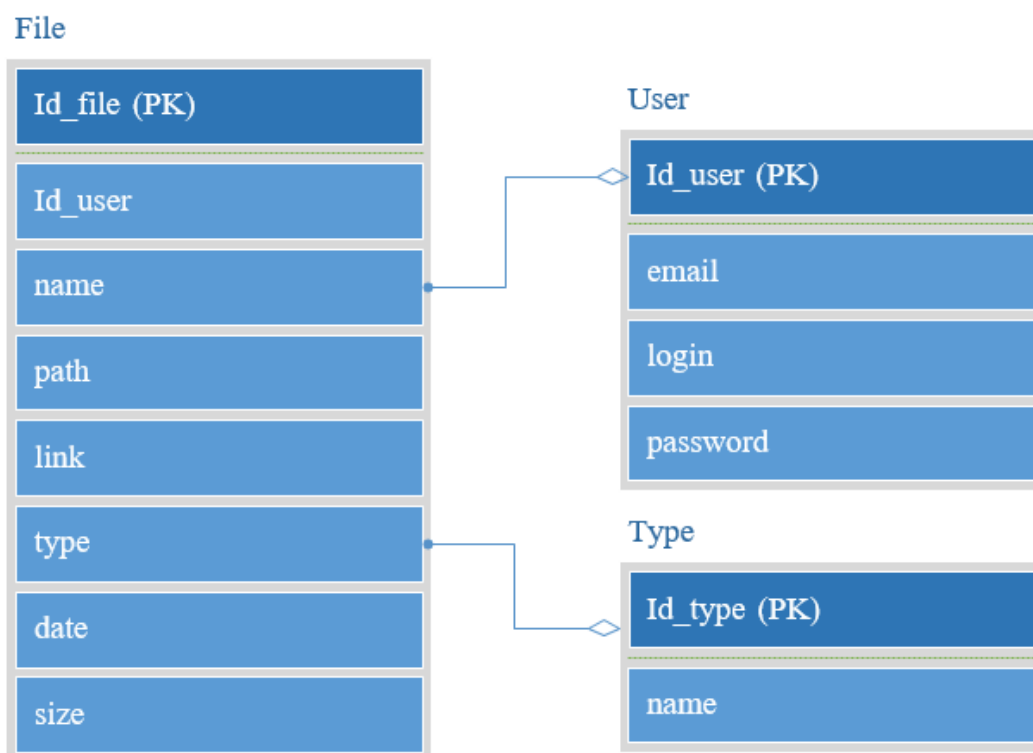


Рисунок 3.9 – Схема бази даних

3.3 Проектування серверної частини

Зважаючи на поставлені задачі та вимоги до розроблюваної системи було розроблено відповідний інтерфейс, який потрібно наділити певним функціоналом.

Як вже було сказано, веб-сторінки посилають відповідні запити до сервера за технологією HTTP. Ці запити передаються на обробку відповідним endpoint`ам, завдяки чому в продукті з'являється динамічність та логічність.

Для створеного інтерфейсу було розроблено ряд endpoint`ів, кожен з яких обробляє певні дії користувача, такі як:

- вхід до особистого кабінету;
- реєстрація нового користувача;
- завантаження файлу на сервер;
- завантаження файлу з серверу;
- вихід з системи.

По натисканню на кнопку «Вход» на сторінці входу формується запит, який зберігає в собі дані, введені користувачем в відповідні поля. Цей запит приймає перший створений endpoint – LoginAsync. Далі формується запит до бази даних за допомогою створених сервісів. Якщо користувач з таким логіном та таким паролем існує в базі, вхід виконано успішно та завантажуються головна сторінка веб-сайту. В разі відсутності даних, користувач отримує повідомлення про помилку.

Схожий функціонал реалізовано для реєстрації користувача. Створений endpoint має назву RegistrationAsync. Він працює за наступним алгоритмом. Спочатку звіряються дані в полях вводу паролю – вони повинні співпадати. Далі перевіряється чи є користувач з таким логіном або адресою електронної пошти в базі. Якщо так, користувач отримує відповідне повідомлення про помилку, якщо ні – створюється запис в базі даних та завантажуються головна сторінка сайту.

До речі, після входу або реєстрації в системі, в браузері створюються так звані Cookie з логіном користувача. Це зроблено для того, щоб користувачу не

потрібно було входити до системи кожен раз, коли завантажується нова або оновлюється стара сторінка.

Для керування завантаженням файлів на сервер було створено endpoint під назвою UploadFileAsync. HTTP технологія також дозволяє передавати файли між клієнтом та сервером. Після обрання файлу та натискання кнопки «Завантажити» endpoint зчитує дані, створює файл в папці користувача з відповідним ім'ям. Далі дані записуються в цей файл. Вся інформація записується в базу.

Варто сказати, що були розроблені спеціальні методи для створення коректного шляху до файлу на сервері, створення унікального посилання на файл на основі ідентифікатора користувача та хешованого ім'я файлу, визначення типу файлу та перерахування розміру файлу з байтів в більш звичний для користувача вигляд – кілобайти, мегабайти та гігабайти.

Завантаження файлів з сервера працює навпаки. Користувач повинен ввести посилання на файл в спеціальну форму. Ці дані передаються endpoint DownloadFileAsync. За цим посиланням відбувається пошук відповідного файлу в базі. Якщо такий існує, програма отримує шлях до файлу на сервері та відкриває його в браузері. Сучасні браузери надають користувачу можливість переглянути деякі типи файлів та завантажити їх автоматично.

Так само працює видалення файлів з серверу, але замість відображення файлу в браузері, він та всі дані про нього видаляються.

Деякий інший функціонал, такий як вихід з системи, реалізовано напряму в TS файлах і не потребує створення окремих endpoint`ів.

Окрім endpoint в програмі було створено декілька сервісів, які наділені необхідними функціями з'єднання з базою. Реалізовано відправлення запитів та отримання вибірок інформації, наприклад отримання та збереження даних про користувача і файли, пошук користувача за логіном, пошук файлу за посиланням та видалення файлу.

Детальний список створених класів представлено нижче:

- File – клас, що представляє опис файлу, має такі ж параметрами, як і в базі даних;

- Type – клас, що описує представлення типу файлу в базі даних;
- User – клас, що описує користувача;
- Function – клас, який зберігає статичні функції такі, як шифрування, створення посилання на файл, форматування розміру файлу і створення шляху до файлу;
- GenericRepository – клас для підключення до бази даних;
- FileService – зберігає функції відправки та обробки запитів до таблиці «file» в базі даних;
- TypeService – зберігає функції відправки та обробки запитів до таблиці «type» в базі даних;
- UserService – зберігає функції відправки та обробки запитів до таблиці «user» в базі даних;
- LoginAsync – endpoint, що обробляє вхід до системи;
- RegistrationAsync – endpoint, який обробляє реєстрацію користувача в системі;
- UploadFileAsync – endpoint, який обробляє функцію завантаження файлу на сервер;
- DownloadFileAsync – endpoint, який обробляє функцію вивантаження файлу з серверу.

Детальний перелік всіх параметрів і функцій класів та їх залежність один від одного зображено на рис 3.11.

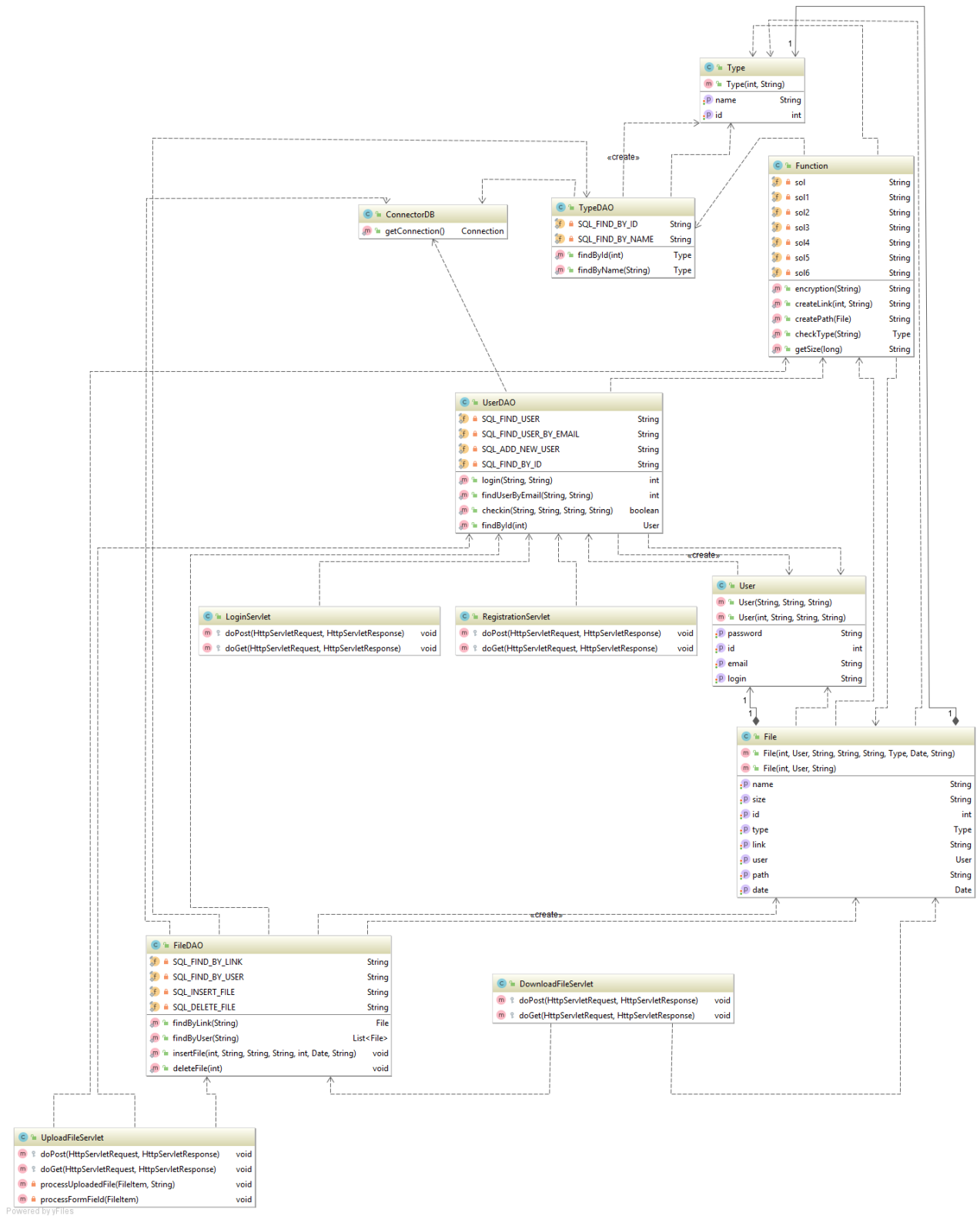


Рисунок 3.11 – Діаграма класів

4 ОХОРОНА ПРАЦІ ТА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

4.1 Загальні питання охорони праці

Темою дипломної магістерської роботи є «Дослідження методів захисту веб-сервісу для хмарного зберігання та обміну файлів». Виконання дипломної роботи здійснювалося з застосуванням персональної ЕОМ, тому даний розділ присвячений питанням охорони праці користувача ЕОМ на стадії розрахунків.

Охорона праці – система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини в процесі трудової діяльності.

Наведене вище визначення охорони праці, яке встановлене Законом України «Про охорону праці», свідчить, що охорона праці являє собою сукупність законів, нормативно-правових актів, а також комплекс різноманітних заходів та засобів, які забезпечують безпеку праці, збереження життя, здоров'я та працездатності людей при виконанні ними трудових обов'язків [17].

Професійні захворювання мають різний характер відповідно області, де працює людина. Розробка автоматизованих систем управління, створення ЕОМ полегшують і прискорюють виконання роботи. Але слід враховувати, що для запобігання отримання професійних захворювань час роботи за ЕОМ необхідно обмежувати, а саму роботу проводити на правильно організованому робочому місці.

Приміщення лабораторії, у якому виконувалася дана робота, розташовано на п'ятому поверсі семиповерхового будинку. Площа приміщення, у якому виконувався дипломний проект, складає 30 м^2 , у ньому 3 робочі місця, тобто на робоче місце приходить 10 м^2 . Об'єм приміщення лабораторії складає 90 м^3 , тобто на одне робоче місце приходить 30 м^3 . Зважаючи на те, що на одне робоче місце згідно ДСанПіН 3.3.2-007-98. [18], повинне приходитися не менш 6 м^2 та не менше 20 м^3 можна зробити висновок, що розміри приміщення відповідають нормам проектування.

4.2 Перелік шкідливих факторів

Виконання роботи здійснювалося за допомогою персонального комп'ютеру. Робота за комп'ютером, як і інші види діяльності людини на виробництві, пов'язана з небезпекою одержання травм і професійних захворювань. Перелік шкідливих і небезпечних виробничих факторів відповідно до ДСанПіН [19], а також джерела їх виникнення наведені в табл. 4.1.

Таблиця 4.1 – Перелік шкідливих і небезпечних виробничих факторів приміщенні з ЕОМ

Назва фактору	Джерела їх виникнення	Характер дії
Незадовільні метеоумови	Недостатнє опалення, стан систем природної і штучної вентиляції	Шкідливий
Незадовільна освітленість	Стан систем природного і штучного освітлення	Шкідливий
Підвищений рівень шуму	Шум створюється кондиціонерами, вентиляторами, перетворювачами напруги ЕОМ	Шкідливий
Електричний струм	Мережа перемінного струму	Небезпечний
ЕМВ	ЕЛТ	Шкідливий
Статистична електрика	Висока напруга в ЕЛТ дисплея і наявність діелектричної поверхні екрана	Шкідливий
Іонізація повітря	Статична електрика і рентгенівське випромінювання	Шкідливий
Психофізіологічна напруга	Монотонність праці, перенапруги зорового аналізатора, розумова перенапруга	Шкідливий
Пожежна небезпека приміщенням	Наявність спалених матеріалів і можливих джерел запалювання	Небезпечний, шкідливий

4.3 Виробнича санітарія

Згідно з ДСН 3.3.6.042-99 [20] треба урахувати високу нервово – емоційну перенапругу користувачів, тому в приміщеннях з ЕОМ повинні

підтримуватися тільки оптимальні параметри мікроклімату, що наведені в табл. 4.2.

Робота на персональній ЕОМ виконується сидячи і супроводжується незначним фізичним навантаженням, витрати енергії не перевищують 139 Вт, тому ця робота, відповідно до ДСН 3.3.6.042-99 [20], відноситься до категорії важкості – легка фізична Іа.

Оптимальні параметри при тривалому і систематичному впливі на людину забезпечують збереження нормального функціонального і теплового стану організму без напруги реакцій терморегуляції.

Таблиця 4.2 – Оптимальні параметри мікроклімату в приміщенні з ЕОМ

Період року	Категорія робіт з важкості	Температура, t, °C	Відносна вологість, %	Швидкість руху повітря, м/с, не більш
Холодний	Легка Іа	22–24	40–60	0,1
Теплий	Легка Іа	23–25	40–60	0,1

Оптимальні параметри забезпечують збереження нормального теплового стану і функціонування організму без потреби до терморегуляції. Цим забезпечуються комфорт і створюються передумови для високого рівня працездатності.

У холодний період року проводиться опалення від центральної тепломережі.

Для створення необхідного мікроклімату у приміщеннях з ЕОМ, відповідно до вимог ДБН В.2.5-67:2013[20], у робочому приміщенні встановлені побутові кондиціонери, що автоматично підтримують необхідні оптимальні параметри температури, незалежно від зовнішніх умов, а також проводиться природне провітрювання приміщення.

4.3.1 Освітлення

Згідно з ДБН В. 2.5-28:2018 [21] та для об'єктів, які світяться, відповідно до розміру об'єкту розрізнення та характеристики зорової роботи визначені нормативні характеристики зорової роботи та занесені до табл. 4.3.

Таблиця 4.3 – Нормативні параметри освітлення для роботи ЕОМ

Характеристика зорової роботи	Найменший розмір об'єкта розпізнавання	Розряд зорової роботи	Підрозряд зорової роботи	Контраст об'єкта розпізнавання	Характеристика фону	Освітленість при штучному освітленні, лк	КПО, D_n , при суміщеному освітленні, %
						Загальному	Боковому
Дуже високої точності	Від 0,15 до 0,3	II	в	Середній	Середній	500	1,5

В приміщенні, що розглядається, застосовують суміщене освітлення – освітлення, при якому недостатнє за нормами природне освітлення доповнюється штучним. Мінімальна освітленість при цьому складає 500лк. Штучне освітлення реалізується шляхом встановлення визначеної кількості ламп білого світла – ЛБ 80.

4.3.2 Шум і вібрації

Шум є одним з найбільш розповсюджених у виробництві шкідливих факторів. Основними джерелами шуму і вібрації є вентилятори системного блоку, накопичувач, розташовані в системному блоці комп'ютера, і принтер. Це може стати джерелом стресу і дискомфорту користувача, знижувати розумову працездатність, підвищувати втомлюваність, послаблювати увагу, сприяти появі головного болю тощо. Відповідно до ДСН 3.3.6. 037-99 [22] робочі місця у

приміщеннях програмістів обчислювальних машин рівень шуму не повинен перевищувати 50 дБА. Відповідно до ДСН 3.3.6. 039-99 [23] рівень загальної вібрації для категорії 3, технологічного типу «в» не повинен перевищувати 75 дБ.

Як захист від шуму, який створюється вентиляторами системних блоків, використовується наступне:

- звукоізоляційний корпус;
- заміна вентилятора на більш якісний;
- використання звукопоглинаючих та звукоізолюючих засобів;
- мідні радіатори як альтернативу вентилятору;
- при монтажі кулерів замість гвинтів встановлювати гумові пробки, що дозволяють ізолювати вентилятор від корпусу.

4.3.3 Електромагнітне випромінювання

Електромагнітне випромінювання шкідливо впливає на здоров'я людини. Згідно НПАОП 0.00-7.15-18 [24], потужність поглиненої дози в повітрі за рахунок супутнього не використаного рентгенівського випромінювання не повинна перевищувати 100 мР/год на відстані 5 см від поверхні пристрою, під час роботи якого воно виникає. Забезпечення захисту оператора та досягнення нормованих рівнів випромінювань ЕОМ рекомендовано застосування екранних фільтрів, локальних світлофорів та інших засобів захисту, які пройшли випробування в акредитованих лабораторіях та отримали позитивний висновок державної санітарно-епідеміологічної експертизи.

Основними принципами захисту від впливу ЕМВ є:

- тривалість роботи за ЕОМ не повинна перевищувати 4 години на день при цьому виконувати перерви через кожні 2 години роботи;
- на одну ЕОМ повинно бути виділено не менше 6м², відстань між сусідніми ЕОМ – 1,5м;
- внутрішнє екранування, що дозволяє суттєво знизити інтенсивність шкідливого опромінювання;

- для попередження, своєчасної діагностики та лікування здоров'я людини, що пов'язано з негативним впливом ЕОМ, користувачі повинні проходити попередні (під час прийому на роботу) і періодичні медичні огляди.

4.4 Електробезпека

Сучасне виробництво нерозривно пов'язане з використанням електроенергії. При виконанні роботи використовувався комп'ютер, який живиться з напругою 220В від однієї фази 3-хфазної 4-хфазної мережі з глухозаземленою нейтраллю.

Основними заходами захисту від ураження електричним струмом згідно з НПАОП 0.00-7.15-18 [24] є:

- забезпечення недоступності струмопровідних частин, що перебувають під напругою, для випадкового дотику;
- організація безпечної експлуатації електроустановок;
- компенсація ємнісної складової струму замикання на землю;
- застосування спеціальних засобів – переносних приладів і запобіжних пристроїв;
- відключення електроустаткування, що ремонтується, і вживання заходів проти помилкового його зворотного включення або само включення;
- приєднання переносного заземлення - закоротки до заземлювальної шини стаціонарного заземлювального пристрою і перевірка відсутності напруги на струмопровідних частинах, що для безпеки проведення робіт підлягають замиканню закоротко і заземленню.

Головне призначення захисного заземлення – знизити потенціал на корпусі електроустаткування до безпечного значення.

Для захисту від ураження електричним струмом для ЕОМ застосовується занулення – це навмисне електричне з'єднання з нульовим захисним провідником металевих не струмоведучих частин електроустановки, які можуть опинитися під напругою.

4.4.1 Індивідуальне завдання

Розрахунок захисного заземлення

Дані для виконання індивідуального завдання зображено у табл. 4.4 та табл. 4.5

Таблиця 4.4 - Дані для виконання індивідуального завдання

№	Трансформаторна підстанція напругою U , кВт	Розміри будинку		Розрахунковий опір природного заземлювача, R_e , Ом	Довжина лінії електропередачі	
		Довжина L , м	Ширин а B , м		$l_{к.л.}$, км	$L_{в.л.}$, км
1	6/ 0,4	24	12	15	70	65

Таблиця 4.5 - Дані для виконання індивідуального завдання

Параметри вертикального електрода		Параметри горизонтального електрода	Питомий опір землі ρ обмірюване, Ом·м	Кліматична зона
Довжина l_v , м	Діаметр d , мм ²	Переріз полоси, мм ²		
5	12	4 x 40	120	I

1. Визначення розрахункового струму замикання на землю і відповідне йому нормативне значення опору розтікання струму захисного заземлення.

$$I_z = \frac{U_l}{350} (35l_k + l_v) \quad (4.1)$$

$$I_z = \frac{(6/0.4)}{350} (35 * 70 + 65) = 43.11(A),$$

де, U_l - лінійна напруга мережі (на високій стороні трансформаторної підстанції), кВ;

l_v, l_k - довжина електрично пов'язаних відповідно кабельних і повітряних ліній, км;

2. Визначення необхідного опору штучного заземлювача.

$$R_z = \frac{125}{I}, \quad (4.2)$$

$$R_z = \frac{125}{43.11} = 2.89(\text{Ом}),$$

$$R_u = \frac{R_{ee}R}{R_e - R_z}, \quad (4.3)$$

$$R_u = \frac{15 * 2.89}{15 - 2.89} = 3.57(\text{Ом}),$$

де, R_e - опір розтікання струму природних заземлювачів, Ом;

R_u - необхідний опір штучного заземлювача, Ом;

R_z - розрахункове нормоване опір ЗУ, Ом;

3. $R_z = R_u$

4. Визначення розрахункового питомого опору землі за формулою:

$$\rho = \rho_{изм} \cdot \Psi' \quad (4.4)$$

$$\rho = 120 * 1.4 = 168(\text{Ом} * \text{м})$$

5. Обчислення опору розтікання струму одиночного вертикального заземлювача R_v , Ом.

$$R_v = \frac{\rho_e}{2\pi l} \left(\ln \frac{2l}{d} + \frac{1}{2} \ln \frac{4t+l}{4t-l} \right) \quad (4.5)$$

$$R_{\epsilon} = \frac{168}{2\pi \cdot 5} \left(\ln \frac{2 \cdot 5}{0.012} + \frac{1}{2} \ln \frac{4 \cdot 3,3 + 5}{4 \cdot 3,3 - 5} \right) = 37.99 (\text{Ом}),$$

де, ρ_{ϵ} – розрахункове питомий опір ґрунту, Ом·м;

l – довжина вертикального стрижня, м;

d – діаметр перерізу, м;

t – відстань від поверхні ґрунту до середини довжини вертикального стрижня, м.

6. Розрахунок наближеної (мінімальної) кількості вертикальних стрижнів

$$n' = \frac{R_{\epsilon}}{R_u}, \quad (4.6)$$

$$n' = \frac{37.99}{3.57} = 10.64 = 10,$$

R_{ϵ} – опір розтікання струму одиночного вертикального заземлювача, Ом;

R_u – необхідний опір штучного заземлювача, Ом

7. Визначення конфігурації групового заземлювача (ряд або контур) з урахуванням можливості його розміщення на відведеній території та відповідну довжину горизонтальної смуги:

по контуру $l_{\Gamma} = 1,05an = 10.5 \cdot 10 = 105$ м

ряд $l_{\Gamma} = 1,05a(n-1) = 10.5 \cdot 9 = 94.5$ м

$$a = k \cdot l_{\epsilon}, \quad (4.7)$$

$$a = 5 \cdot 2 = 10,$$

де, k – коефіцієнт кратності, що дорівнює 2;

l_{ϵ} – довжина вертикального стрижня.

n – кількість вертикальних стрижнів.

8. Обчислення опору розтікання струму горизонтального стрижня R_{Γ}

$$R_{\varepsilon} = \frac{\rho}{2\pi l} \ln \frac{2l^2}{bt}, \quad (4.8)$$

$$R_{\varepsilon} = \frac{168}{2\pi * 5} \ln \frac{2 * 5^2}{12 * 3.3} = 1.24(Ом),$$

де ρ – розрахунковий питомий опір ґрунту, Ом·м;

l – довжина горизонтальної смуги, м;

b – ширина полоси, м;

t – відстань від поверхні ґрунту до середини ширини горизонтальної смуги.

9. Вибір коефіцієнтів використання вертикальних стрижнів і горизонтальної смуги з урахуванням числа вертикальних стрижнів і відносини відстані між стрижнями до їх довжини

$$(\eta_{\varepsilon}) = 0.83$$

$$(\eta_{\Gamma}) = 0.80$$

10. Розрахувати еквівалентний опір розтікання струму групового заземлювача

$$R_{\varepsilon p} = \frac{R_{\varepsilon} R_{\Gamma}}{R_{\varepsilon} \eta_{\Gamma} + R_{\Gamma} \eta_{\varepsilon} \cdot n}, \quad (4.9)$$

$$R_{\varepsilon p} = \frac{37.99 * 1.24}{37.99 * 0.83 + 1.24 * 0.8 * 10} = 1.13(Ом),$$

Результати розрахунків наведені в табл. 4.6

Таблиця 4.6 – результати розрахунків

$\rho_{\Gamma p}$ Ом·м	l_{ε} , м	n , шт	l_{Γ} , м	η_{ε}	η_{Γ}	R_{ε} , Ом	R_{Γ} , Ом	$R_{\Gamma p}$, Ом	R_{Σ} , Ом
168	10.5	10	105	0.83	0.80	37.99	1.24	1.13	3.57

4.5 Пожежна безпека

По категорії вибухові та пожежа небезпеки, згідно ДСТУ Б.В.1.1- 36:2016 [25] дане приміщення відноситься до категорії В пожежонебезпечні через присутність твердих спалених матеріалів, таких як: робочі столи, ізоляція, папір та інше, ступень вогнестійкості II, згідно ДБН В.1.1-7:2016 [26].

Для даного класу будівель і місцевості із середньою грозовою діяльністю 10 і більше грозових годин на рік, тобто для умов міста Харкова встановлена III рівень захисту від блискавок відповідно де ДСТУ Б.В.1.1- 36:2016 [25].

Пожежна безпека людини забезпечується використанням вуглекислотного вогнегасника ВВК-5, ємністю 5 літрів відповідно з речовиною гасіння вогню малі електропровідності. Застосування пінних вогнегасників виключено, тому ще ЕОМ може перебувати під напругою. Рибаче місце відповідає всім вимогам пожежної безпеки.

Пожежі на промислових підприємствах виникають у більшості випадків від несправностей технологічного обладнання, електроустаткування, контрольно-вимірювальних та захисних приладів, необережного поводження з вогнем та порушення правил пожежної безпеки обслуговуючим персоналом.

Згідно з вимогами ДБН В.2.5-56-2015 [27] пожежна безпека забезпечується наступними мірами:

- системою запобігання пожеж;
- системою протипожежного захисту;

Система запобігання пожеж передбачає запобігання утворення пального середовища і запобігання утворення в пальному середовищі джерел запалювання.

Коли від пожежі захищаються приміщення з ЕОМ, то їх рекомендується оснащувати вуглекислотними вогнегасниками. Вогнегасник знаходиться на видному і легко доступному місці. Відстань від можливого осередку пожежі до місця розташування вогнегасника має бути не більше ніж 30м. також необхідним заходом безпеки є евакуаційні виходи (не менше двох).

Організаційними заходами протипожежної профілактики є:

- вступний інструктаж при надходженні на роботу;
- навчання виробничого персоналу протипожежним правилам;
- наявність плану евакуації.

4.6 Охорона навколишнього природного середовища

Проблема охорони й оптимізації навколишнього природного середовища виникла як неминучий наслідок сучасної промислової революції.

Збільшення використання енергії призводить до порушення екологічної рівноваги природного середовища, яке складалася століттями.

Поряд з цим, підвищення технічної оснащеності підприємств, застосування нових матеріалів, конструкцій і процесів, збільшення швидкостей і потужностей виробничих машин впливають на навколишнє середовище.

Основними задачами Закону України "Про охорону навколишнього природного середовища" [28], прийнятого 26 червня 1991 року, є регулювання відносин в області охорони природи, використання і відтворення природних ресурсів, забезпечення екологічної безпеки, попередження і ліквідація наслідків негативного впливу на навколишнє середовище господарської й іншої діяльності людини, збереження природних ресурсів, генетичного фонду, ландшафтів і інших природних об'єктів.

При масовому використанні моніторів та комп'ютерів не можна не враховувати їхній вплив на навколишнє середовище на всіх стадіях – при виготовленні, експлуатації та після закінчення терміну служби.

Міжнародні екологічні стандарти, що діють на сьогоднішній день в усьому світі, визначають набір обмежень до технологій виробництва та матеріалів, які можуть використовуватися в конструкціях пристроїв. Так, за стандартом ТСО-95, вони не повинні містити фреонів (турбота про озоновий шар), полівінілхлориді, бромідів (як засобів захисту від загоряння).

У стандарті ТСО-99 закладене обмеження за кадмієм у світлочутливому шарі екрана дисплея та ртуті в батарейках; є чіткі вказівки відносно пластмас,

лаків та покриттів, що використовуються. Відмовитися від свинцю в ЕЛТ поки неможливо. Поверхня кнопок не повинна містити хром, нікель та інші матеріали, які визивають алергічну реакцію. ГДК пилу дорівнює $0,15 \text{ мг/м}^3$, рекомендовано $0,075 \text{ мг/м}^3$; ГДК озону під час роботи лазерного принтеру – $0,02 \text{ мг/м}^3$. Особливо жорсткі вимоги до повторно використовуваних матеріалів.

5 ЦИВІЛЬНИЙ ЗАХИСТ

Цивільний захист – це функція держави, спрямована на захист населення, території, навколишнього природного середовища та майна від надзвичайних ситуацій шляхом запобігання таких ситуацій, ліквідації їх наслідків та надання допомоги постраждалим в мирний час та в особливий період [29].

У даному розділі дипломної роботи розглядається питання: «Сили і засоби, які залучаються для проведення рятувних та інших невідкладних робіт при гасінні пожеж та порятунку людей з палаючих будинків». У результаті виникнення й розвитку будь-якої надзвичайної ситуації можуть з'явитися постраждалі або людські жертви.

Актуальність розгляду цієї теми обумовлена ризиком при роботі великої кількості людей в офісі з чималою наявністю комп'ютерної техніки, електропристроїв та інше. У разі пожежі всі працівники ризикують отримати значні травми та опіки у разі непрофесійних оперативних дій при проведенні пожежно-рятувальних робіт та у разі незнання послідовності необхідних особистих дій при виникненні небезпечних ситуацій.

5.1 Оперативні дії та основні завдання органів управління та підрозділів під час проведення пожежно-рятувальних робіт

Оперативні дії проводяться з дотриманням безпеки праці і можуть проводитися в умовах високого психологічного та фізичного навантаження, підвищеного ризику, прямої небезпеки для життя і здоров'я учасників гасіння пожеж. Особи, залучені до оперативних дій під час гасіння пожеж, повинні дотримуватися вимог охорони та безпеки праці [30].

Основним оперативним завданням особового складу пожежно-рятувальних підрозділів ОРС ЦЗ під час гасіння пожеж є рятування людей у разі виникнення загрози їх життю та гасіння пожеж.

Виконання оперативного завдання забезпечується такими силами:

- особовим складом органів управління та пожежно-рятувальних підрозділів ОРС ЦЗ, у тому числі курсантами, науковими та науково-педагогічним складом навчальних закладів та науково-дослідних установ системи ДСНС;

- особовим складом (працівниками і членами) підрозділів місцевої, відомчої і добровільної пожежної охорони.

Для гасіння пожеж можуть залучатися в установленому чинним законодавством порядку особовий склад Національної поліції, Національної гвардії, Збройних Сил України, працівники державних, регіональних, комунальних, об'єктових аварійно-рятувальних служб а також аварійно-рятувальні служби громадських організацій, населення.

Пожежно-рятувальне відділення на основному пожежному автомобілі (караул у складі одного відділення) є первинним тактичним пожежно-рятувальним підрозділом, здатним самостійно виконувати окремі оперативні завдання з рятування людей та гасіння пожеж. Караул у складі двох і більше пожежно-рятувальних відділень на основних та спеціальних пожежних автомобілях є основним тактичним підрозділом, здатним самостійно вирішувати оперативні завдання відповідно до своїх тактичних можливостей.

Для виконання оперативного завдання використовуються такі засоби:

- пожежно-рятувальні автомобілі, аварійно-рятувальна техніка, техніка пристосована для цілей пожежогасіння та інші транспортні засоби;
- пожежно-, аварійно-рятувальні засоби та обладнання;
- засоби зв'язку та освітлення;
- засоби індивідуального захисту органів дихання, зору та шкіри;
- вогнегасні речовини (вода, піна, порошки, гази тощо);
- системи та обладнання протипожежного захисту;
- інженерно-технічні засоби об'єктів господарства (пожежні водойми, пірси, градирні, водонапірні вежі, фонтани тощо).

Оперативні дії з гасіння пожеж на об'єктах, на які передбачено складання оперативних планів (карток) пожежогасіння, планів локалізації і ліквідації аварій

та аварійних ситуацій, слід здійснювати з урахуванням особливостей, визначених у цих планах (картках).

До оперативних дій під час гасіння пожеж входять:

- розвідка пожежі;
- рятування людей і евакуація майна на пожежі;
- оперативне розгортання сил і засобів;
- гасіння пожежі;
- виконання спеціальних робіт;
- згортання сил і засобів;
- повернення до місця постійної дислокації.

5.2 Рятування людей при пожежі

Рятувальні роботи організовуються і проводяться, якщо:

- є загроза людям, у тому числі від небезпечних факторів пожежі;
- люди не можуть самостійно покинути небезпечні місця;
- є загроза розповсюдження вогню та диму на шляхи евакуації людей;
- передбачається застосування небезпечних для життя людей вогнегасних речовин і сполук.

Способи рятування людей визначають КГП та особи, які проводять рятувальні роботи, враховуючи обстановку та стан осіб, яких рятують. Рятування людей на пожежі проводиться одночасно з розгортанням сил і засобів для гасіння пожежі. Подача стволів для створення безпечних умов рятування людей обов'язкова, якщо людям безпосередньо загрожує вогонь і шляхи рятування відрізані чи можуть бути відрізані вогнем.

У разі, якщо сил і засобів недостатньо для одночасного рятування людей і гасіння пожежі основні зусилля особового складу працюючих підрозділів зосереджуються на рятуванні людей, а КГП зобов'язаний викликати додаткові сили і засоби.

Для рятування людей використовують найкоротші і найбезпечніші шляхи:

- основні та запасні виходи;

- віконні прорізи, балкони, лоджії, галереї та переходи з використанням зовнішніх пожежних драбин, ручних і автомобільних драбин, автопідіймачів та інших рятувальних пристроїв, що є на оснащенні пожежно-рятувальних підрозділів;

- люки в перекриттях, якщо через них можна вийти з будівлі чи перейти до безпечної її частини;

- спеціально зроблені прорізи в перегородках, перекриттях і стінах для рятування людей.

Основними способами рятування та евакуації людей є:

- самостійний вихід людей;
- виведення людей у супроводі пожежних-рятувальників, коли шляхи евакуації задимлені або вік і стан людей, яких рятують, не дозволяє їм самостійно вийти з небезпечної зони (діти, вагітні, люди похилого віку, хворі);
- винесення людей, які не можуть самостійно рухатися;
- спуск людей по зовнішніх пожежних, ручних та автомобільних драбинах, за допомогою автопідіймачів, рятувальних мотузок та інших рятувальних пристроїв, якщо основні шляхи евакуації (рятування) відрізані вогнем чи димом;
- за допомогою вертольотів [31].

Під час проведення рятувальних робіт необхідно:

- ужити заходів щодо попередження паніки, використовуючи технічні засоби та інші конструктивні можливості об'єкта і пожежно-рятувальних підрозділів;

- залучити адміністрацію і обслуговуючий персонал (за потреби);
- викликати бригади екстреної медичної допомоги, за потреби - інші аварійні служби;

- надавати в необхідних випадках домедичну допомогу постраждалим силами особового складу пожежно-рятувальних підрозділів до моменту прибуття бригад екстреної медичної допомоги;

- визначити місця для розміщення людей, яких урятовано та евакуйовано;
- за потреби залучити психологів для надання допомоги потерпілим.

У разі наявності інформації про перебування в небезпечній зоні людей, яких пожежні-рятувальники не можуть знайти в указаних місцях, необхідно ретельно оглянути та перевірити всі задимлені і суміжні з місцем пожежі приміщення, де можуть знаходитися люди. Пошук людей припиняється тільки після того, як встановлено, що з небезпечної зони всіх людей евакуйовано та врятовано.

5.3 Гасіння пожеж

Гасіння пожежі - дії, спрямовані на припинення горіння в осередку пожежі, обмеження впливу її небезпечних факторів та усунення умов для самовільного відновлення пожежі після гасіння.

Обстановка, яка може бути під час гасіння пожежі:

- наявність великої кількості людей, які потребують допомоги, і виникнення серед них паніки;
- складне планування приміщень;
- розповсюдження вогню пустотами, конструкціями, каналами, системами вентиляції і пневмотранспорту, через віконні прорізи, лоджії, балкони, горючими матеріалами, технологічним обладнанням як у вертикальному, так і горизонтальному напрямках;
- швидке зростання температури та переміщення теплових потоків у напрямку відкритих прорізів;
- виділення диму, токсичних продуктів та швидке їх поширення;
- наявність обладнання під електричною напругою, пошкодження ізоляції електропроводів та електрообладнання;
- наявність ЛЗР та ГР, можливість розливу та викиду нафтопродуктів;
- утворення вибухонебезпечних газо-, паро-, пилоповітряних сумішей та сумішей продуктів термічного розкладання речовин і матеріалів з повітрям;
- вибухи посудин під тиском;
- можливість викиду радіоактивних та небезпечних хімічних речовин;

- деформація та руйнування конструктивних елементів будівель, споруд, технологічного обладнання;
- відсутність джерел протипожежного водопостачання або їх несправність;
- наявність у будівлях великої кількості культурних, наукових та інших цінностей;
- наявність інших небезпечних факторів.

Ліквідація пожежі - стадія гасіння пожежі, коли припинено горіння, дію небезпечних факторів пожежі та усунуто умови для самовільного відновлення пожежі.

Ліквідація пожежі досягається:

- дією на поверхню матеріалів, що горять, охолоджувальними вогнегасними речовинами;
- створенням у зоні горіння чи навколо неї негорючого газового або парового середовища;
- створенням між зоною горіння і горючим матеріалом та повітрям (іншим окисником) ізолюючого шару з вогнегасних речовин або з негорючих матеріалів;
- хімічним уповільненням реакції горіння шляхом застосування порошкових, газових та аерозольних вогнегасних речовин.

За потреби для недопущення самовільного відновлення пожежі за рішенням КГП на місці ліквідованої пожежі проводиться розбирання конструктивних елементів будівель, споруд, матеріалів та їх охолодження (проливання) вогнегасними речовинами в місцях інтенсивного виділення тепла.

Вирішальним напрямком оперативних дій на пожежі є напрямок, на якому утворилася небезпека для людей, загроза вибуху, руйнування конструкцій, викиду радіоактивних і небезпечних хімічних речовин, найбільш інтенсивне поширення вогню та на якому оперативні дії пожежно-рятувальних підрозділів на цей час можуть забезпечити успіх гасіння пожежі.

Сили і засоби першочергово вводяться на вирішальному напрямку оперативних дій.

Під час гасіння пожеж особовим складом пожежно-рятувальних підрозділів виконуються такі спеціальні роботи:

- рятування людей;
- надання домедичної допомоги постраждалим;
- роботи в задимлених і загазованих середовищах;
- видалення диму;
- розкриття та розбирання конструкцій;
- відключення електромереж і обладнання;
- підйом на висоту та спуск з неї;
- організація зв'язку та освітлення на місці пожежі;
- проведення інших робіт за рішенням КГП.

6 ТЕХНІКО-ЕКОНОМІЧНА ЧАСТИНА

6.1 Резюме

Розроблене програмне забезпечення зберігання та обміну файлів призначено для експлуатації різними типами користувачів. Даний веб-сервіс дозволить істотно спростити і підвищити ефективність обміну великими за об'ємом файлами. Істотний упор при розробці системи робиться на простоту експлуатації і надійність при використанні.

Програмне забезпечення, розроблене в дипломному проекті, призначено для використання на персональних комп'ютерах з Intel Core i3-9600K і вище або сумісних з ними. Аналіз питань про ринок збуту, конкуренції, стратегії маркетингу і так далі дозволяє робити висновок про доцільність застосування розроблювальної системи.

Витрати на розробку складають 50000 грн.

6.2 Опис програмного продукту

Найменування товару - дослідження методів захисту веб-сервісу для хмарного зберігання та обміну файлів.

Призначення – завантаження та зберігання файлів у хмарному сховищі. Область використання – орієнтований на використання як звичайними користувачами так і великими компаніями. Характеристики продукту приведені в табл. 6.1.

Таблиця 6.1 - Характеристики ПП

Найменування	Значення
Операційна система	Windows 10
Оперативна пам'ять	4096 Мбайт и вище
CPU	Intel Core i3-9600K і вище

6.3 Дослідження й аналіз ринку збуту

- Сегментація ринку по споживачах:

Таблиця 6.2 - Сегментація ринку по основних споживачах

Галузь використання	Код споживача	Споживач			
		I	II	III	IV
Корпорації	A	+	+	+	+
Підприємства	B	+	+	+	+
Користувачі	B	-	-	-	+

I - системні адміністратори;

II - адміністратори груп;

III - спостерігачі ресурсів;

IV - ліцензовані користувачі.

Як видно з табл. 6.2, даний виріб призначений для використання співробітниками підприємств, корпорацій та звичайними користувачами. Уточнимо ємність сегментів ринку (дані в табл. 6.3).

Таблиця 6.3 - Аналіз ємності сегментів ринку

Галузі використання	Кількість об'єктів	Передбачуване число продажів одному об'єктові	Передбачувана ємність сегмента
Корпорації	10	1	10
Підприємства	5	1	5
Користувачі	30	1	30
Разом місткість ринку	45		45

Як видно з табл. 6.3, найбільше число передбачуваних продажів одному об'єктові приходить на звичайних користувачів. Це обумовлено специфікою програмного продукту.

Основними вимогами споживачів є простота використання, швидкість і надійність керування, інтуїтивний-зрозумілий графічний інтерфейс користувача, простота відображення.

Продаж розробленого продукту охоплює всю Україну, але не виключений продаж і за її межами. Максимальна кількість потенційних споживачів у розглянутому регіоні - 300.

Прогноз обсягів продажів програмного комплексу приведений у табл. 6.4.

Таблиця 6.4 - Прогноз обсягів продажів ПП

Періоди	Кількість
Для першого року реалізації	
Січень	3
Лютий	2
Березень	2
Квітень	4
Травень	5
Червень	1
Липень	0
Август	0
Вересень	2
Жовтень	1
Листопад	0
Грудень	0
Усього	20
Для другого року реалізації	
I квартал	6
II квартал	5
III квартал	3
IV квартал	1
Усього	15
Для третього року реалізації	
Усього	10

Параметрична сегментація ринку:

Для проведення багатofакторної сегментації продукту оцінимо його характеристики, що відповідають обраним нами параметрам (по п'ятибальній шкалі). Багатofакторна сегментація приведена в табл. 6.5.

Таблиця 6.5 - Параметрична сегментація ринку

Фактори, що характеризують товар	Категорія споживачів			Підсумкова оцінка	Відсоток до загального підсумку
	А	Б	В		
Ціна	3	4	5	12	18.75
Вимоги до ЕОМ	3	3	5	11	17.18
Простота використання	5	5	5	15	23.4
Надійність	5	5	5	15	23.4
Швидкість запуску	3	4	4	11	17.18
Разом	19	21	24	64	100

Виходячи з даних табл. 6.5, можна зробити висновок про те, що такі фактори як простота використання та надійність є найбільш важливими, а сегмент ринку В пред'являє найбільше високі вимоги до сукупності якісних параметрів розроблювального виробу.

6.4 Оцінка конкурентоздатності

За наявними в розроблювачів відомостями розроблене програмне забезпечення має декілька аналогів.

Розроблений програмний продукт поєднує в собі прикладне значення зі зручністю в звертанні і наочністю відображуваної інформації.

Як сервіс розроблений програмний продукт підтримує довідкову систему, що дозволяє одержати достатні зведення про роботу програми.

Оскільки існує чимало аналогів розробленого програмного забезпечення, то розрахунок узагальненого показника якості будемо робити в порівнянні даного ПП з усередненим показником аналогів.

Величина відносного показника якості обчислюється по формулі:

$$M_{i'} = \frac{P_{iд}}{P_i} \text{ чи} \quad (6.1)$$

$$M_{iн} = \frac{P_i}{P_{iг}}, \quad (6.2)$$

при цьому $M_{iн} > 1.0$. Вихідні дані для розрахунку приведені в табл. 6.6.

Таблиця 6.6 - Розрахунок узагальненого показника якості

Параметри	Одиниці виміру	Вагомість V_i	Абсолютне значення параметрів		Узагальнене значення показників			
					Новий ПП		Усереднений показник аналогів ПП	
			Новий ПП P_i	Усереднений показник аналогів ПП $P_{iг}$	Відносний одиничний показник $M_{iн}$	$V_i \times M_{iн}$	Відносний одиничний показник $M_{iг}$	$V_i \times M_{iг}$
1. Ціна ПП	грн.	0,2	50000	60000	0,83	0,166	1,0	0,2
2. Простота використання		0,2	5	4	1,25	0,25	1,0	0,2
3. Швидкість запуску	сек.	0,2	2	2,5	0,8	0,16	1,0	0,2
4. Надійність		0,4	5	4	1,25	0,5	1,0	0,4
Усього		1				1,076		1

6.5 Стратегія маркетингу

Поширення товару буде вироблятися шляхом прямих продажів.

Розраховуємо основну заробітну плату розроблювача ($Z_{зп}$ представленого ПП). Розрахунок виконуємо по формулі:

$$Z_{зп} = \overline{Z_m} \times T, \quad (6.3)$$

де T - час розробки ПП;

$\overline{Z_m}$ - середня заробітна плата розроблювача:

$$\overline{Z_m} = \frac{\sum_{i=1}^n Z_{mi}}{n}, \quad (6.4)$$

де Z_{mi} - заробітна плата i -го розроблювача;

n - кількість розроблювачів.

Таблиця 6.7 - Основна заробітна плата розроблювачів

Посада	Заробітна плата, грн.	Кількість розроблювачів
Розробник додатків	10000	1

Використовуючи дані, приведені в табл. 6.7, і виходячи з того, що розробка ПП велася 4 місяців, одержимо:

$$Z_{зп} = 10000 \times 4 = 40000 \text{ грн.}$$

Експлуатаційні витрати:

$$E_p = T_{мв} \times C_{мч}, \quad (6.5)$$

де $C_{мч}$ - вартість машино-часа роботи ЕОМ ($C_{мч} = 2$ грн);

$T_{мв}$ - час налагодження програми на ЕОМ:

$$T_{мв} = T \times \Phi \times T_{ч}, \quad (6.6)$$

де Φ - кількість робочих днів у місяці (22 дня);

$T_{\text{ч}}$ - кількість годин, пророблених на ЕОМ у день (5 ч).

$$T_{\text{мв}} = 4 \times 22 \times 5 = 440 \text{ ч.}$$

$$E_p = 440 \times 2 = 880 \text{ грн.}$$

Потреби в матеріальних ресурсах і устаткуванні для виробництва програмного продукту приведені в табл. 6.8 і табл. 6.9.

Таблиця 4.8 - Витрати на обладнання

Обладнання	Призначення	Кількість	Вартість у грн.
Intel Core i3-9600K	Для написання програми і її налагодження, а також для підготовки документів	1	10000
Принтер Epson XP-117		1	1500
Усього Зоб			11500

Вартість основних виробничих фондів визначається по формулі:

$$C_{\text{ОПФ}} = \frac{Z_{\text{об}} \times T}{12}, \quad (6.7)$$

де Зоб - витрати на обладнання (дані в табл. 6.8).

$$C_{\text{ОПФ}} = \frac{11500 \times 4}{12} = 3833.33 \text{ грн.}$$

Таблиця 6.9 - Витрати на матеріали

Матеріали	Призначення	Вартість одиниці в грн.	Кількість	Сума у грн.
CD диск	Збереження вихідних текстів і виконавчого модуля	5	2	10
Картридж	Для виробництва документації	200	1	200
Папір	Документування	70	1 пачка (250 аркушів)	70
Разом				280

Інші статті витрат на розробку програмного продукту приведені в табл. 6.10.

Прибуток Π обчислюється як 30% від витрат на розробку ПП

$$\Pi = 0.3 \times Z_p, \quad (6.8)$$

$$\Pi = 0.3 \times 60054,99 = 18016,49 \text{ грн.}$$

Таблиця 4.10 - Розрахунок витрат на розробку програмного продукту

№ п/п	Найменування статей витрат	Значення в грн.
1	Вартість основних фондів	3833,33
2	Вартість матеріалів	280
3	Основна заробітна плата розроблювачів	40000
4	Додаткова заробітна плата розроблювачів (10% від п3)	4000
5	Єдині соціальні відрахування (22% від п3+п4)	9680
6	Експлуатаційні витрати	880
7	Накладні витрати (до 70% від п1)	881,66
8	Комунальний податок (10% від мін. зар. плати)	500
Разом Z_p		60054,99

Максимальна ціна розроблювального ПП буде

$$C_{\max} = 1.2 \times (Z_p + 1.3 \times \Pi), \quad (6.9)$$

$$C_{\max} = 1.2 \times (60054.99 + 1.3 \times 18016.49) = 100171.71 \text{ грн.}$$

Отримана ціна є максимальною. Однак ця ціна може бути зменшена і складатися з витрат на тиражування ($Z_{\text{тир}}$) і адаптацію ($Z_{\text{ад}}$) даного продукту споживачам.

Витрати на тиражування складаються з вартості диска, машинного часу, необхідного для розробки і налагодження програми, а також оплати праці виконавця.

Мінімальну ціну C_{\min} визначаємо по формулі:

$$C_{\min}=1.2 \times (Z_{\text{тир}} + Z_{\text{ад}} + 1.3 \times \Pi'), \quad (6.10)$$

де $Z_{\text{ад}}$ - витрати на адаптацію (приймаємо 5% від Z_p);

Π' - прибуток з одного продажу, грн:

$$\Pi'=0.3 \times (Z_{\text{тир}} + Z_{\text{ад}}), \quad (6.11)$$

$Z_{\text{тир}}$ - витрати на тиражування ПП:

$$Z_{\text{тир}}=C_{\text{мч}} \times T_{\text{к}} + Z_{\text{д}} + Z_{\text{и}}, \quad (6.12)$$

де $T_{\text{к}}$ - час копіювання системи, година (приймаємо 0.005 години);

$Z_{\text{д}}$ - вартість дискети, грн (приймаємо 1.6 грн);

$Z_{\text{и}}$ - зарплата виконавця, грн/година:

$$Z_{\text{и}} = \frac{10000}{22 \times 5} = 90,9 \text{ грн/година},$$

$$Z_{\text{тир}}=2 \times 0.005 + 5 + 90.9 = 95.01 \text{ грн},$$

$$Z_{\text{ад}}=0.05 \times 60054.99 = 3002.74 \text{ грн},$$

$$\Pi'=0.3 \times (95.01 + 3002.74) = 929.32 \text{ грн}.$$

Мінімальна ціна буде:

$$C_{\min}=1.2 \times (95.01 + 3002.74 + 1.3 \times 929.32) = 5167.03 \text{ грн}.$$

Виходячи з отриманих результатів C_{\min} і C_{\max} установимо продажну ціну без ПДВ. Ціна програмного продукту знаходиться в межах

$$5167.03 \text{ грн} \leq C_{\text{прод}} \leq 100171.71 \text{ грн}.$$

Приймаємо продажну ціну програмного продукту без ПДВ рівної 50000 грн.

Реклама продукту буде побудована в такий спосіб. З огляду на специфіку розроблювального продукту реклама повинна здійснюватися серед потенційних

споживачів товару. Рекламувати даний ПП найбільш актуально у соціальних мережах.

Ціна одного рекламного оголошення дорівнює 10000 грн. Передбачається одночасно з виходом продукту випустити і розіслати по 1 рекламному оголошенню у 5 соціальних групах.

Таким чином, ціна реклами буде складати:

$$5 \times 10000 = 50000 \text{ грн.}$$

ВИСНОВКИ

У дипломній роботі вирішена науково-технічна задача – дослідження методів захисту веб-сервісу для хмарного зберігання та обміну файлів.

У процесі виконання дипломної роботи було проаналізовано найпоширеніші загрози веб-застосунків. Досліджено існуючі методи хешування, порівняно кожний з них та на основі дослідження обрано метод SCrypt для використання при розробці веб-сервісу. Проведено аналіз щодо способів зберігання файлів користувача у хмарному сховищі. Були описані основні методи шифрування бази даних з метою підвищення захищеності даних користувачів. Проаналізовано та обрано необхідні технології для створення серверної частини та для створення бази даних.

У проектному розділі було виконано розробку веб-інтерфейсу, бази даних, алгоритму хешування паролів і проектування серверної частини згідно поставлених вимог.

Після проведення аналізу питань з охорони праці та навколишнього середовища були зроблені необхідні висновки та враховано усі побажання для максимально безпечного процесу створення програмного продукту з найменшою кількістю ризиків. Робоче місце налаштовано згідно норм НПАОП 0.00-1.28-2010. Мікроклімат у приміщення відповідає стандартам ДБН В.2.5.-67-2013. Правила протипожежної безпеки відповідають нормам ДБН В.2.5-56-2015.

Були оглянуті основні питання цивільного захисту населення, описані основні оперативні дії та завдання органів управління під час проведення пожежно-рятувальних робіт. Наведено порядок дій під час рятування людей від пожежі. Описано порядок дій під час гасіння пожежі.

У техніко-економічній частині було проведено оцінку конкурентоздатності продукту. Виходячи з розрахунку мінімальної та максимальної ціни було встановлено ціну продукту без ПДВ, рівною 50000грн.

Цей показник на 20% менший за ціну конкурентів. Тож, можна зробити висновок, що створений програмний продукт конкурентоздатний.

Програмний продукт повинен користуватися великим попитом на ринку, оскільки має меншу ціну, вбирає всі позитивні якості аналогів, має адаптивний дизайн та захищеність персональних даних користувачів.

У додатку А наведено програмний код розробленого веб-сервісу.

Розроблений програмний продукт розроблявся згідно з ДСТУ та відповідає вимогам технічного завдання.

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

- 1 Клаверов В.Б. Проблемы противодействия компьютерной преступности/
Клаверов В.Б. – Режим доступа до ресурсу:
<http://www.securitylab.ru/contest/382194.php>
- 2 Report: Cenzic Application Vulnerability Trends Report: 2014 [Электронный
ресурс] // Cenzic. – 2014. – Режим доступа до ресурсу:
<https://info.cenzic.com/2013-Application-Security-Trends-Report.html>.
- 3 OWASP Secure Coding Practices Quick Reference Guide. // OWASP. – 2010.
- 4 OWASP Top 10 - 2017 The Ten Most Critical Web Application Security
Risks, 2017.
- 5 The Bobby Tables Guide to SQL Injection. Archived from the original on
November 7, 2017. Retrieved October 30, 2017
- 6 Martin Anderson. Cross-site scripting enabled on 1000 major sites – including
financial sites. The Stack. 24 лютого 2016., 205с.
- 7 CERT Vulnerability Notes Database. Software Engineering Institute. Original
Release Date: 2008. – 21с.;
- 8 NIST Comments on Cryptanalytic Attacks on SHA-1 - NIST Information
Technology Laboratory / 2006.
- 9 Введения в ASP.NET Core // Режим доступа до ресурсу:
<https://metanit.com/sharp/aspnet5/1.1.php>
- 10 ASP.NET Core - нова епоха в розвитку ASP.NET // Режим доступа до
ресурсу: <https://metanit.com/sharp/entityframework/1.1.php> /. - 2009р.
- 11 Дейт. К. Дж. Введения в системы баз даних - Introduction to Database
Systems. - 8-е изд. - М.: «Вильямс», 2006.
- 12 Драч В.Е., Родионов А.В., Чухраева А.И. Выбор системы управления
базами данных для информационной системы промышленного предприятия //
Электромагнитные волны и электронные системы. 2018. Т. 23. № 3. С. 71-80.
- 13 MySQL / Режим доступа до ресурсу:
<https://uk.wikipedia.org/wiki/MySQL>.

14 Самые популярные СУБД: рейтинг 2018-го года. Режим доступа до ресурсу: <https://itsource.com.ua/blog/samye-populjarnye-subd-rejting-2018-go-goda/>

15 HeidiSQL — MySQL, MSSQL and PostgreSQL made easy. Режим доступа до ресурсу: <https://www.heidisql.com/> .- 2019р.

16 Encyclopedia of Cryptography and Security / Ed. by Henk C. A. van Tilborg and Sushil Jajodia. — Springer, 2011. — P. 307—312.

17 Закон України «Про охорону праці» у редакції від 27.12.2019, підстава - 341-IX

18 ДСанПіН 3.3.2-007-98 Державні санітарні правила і норми. Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. — Чинний від 10.12.1998 р.

19 ДСанПіН. Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу // Зареєстровано в Міністерстві юстиції України 6 травня 2014 р. за № 472/25249.

20 ДБН В.2.5-67:2013 Опалення, вентиляція та кондиціонування.

21 ДБН В. 2.5-28:2018 Державні будівельні норми України. Природне і штучне освітлення. — Чинний від 01.03.2019 р.

22 ДСН 3.3.6.037-99. Санітарні норми виробничого шуму, ультразвуку та інфразвуку // Затверджено постановою Головного санітарного лікаря України від 01 грудня 1999 року №37.

23 ДСН 3.3.6. 039-99 Державні санітарні норми виробничої загальної та локальної вібрації // Затверджено постановою Головного санітарного лікаря України від 01 грудня 1999 року №39.

24 НПАОП 0.00-7.15-18. Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями. Затверджено наказом Міністерства соціальної політики України від 14.02.2018 № 207.

25 ДСТУ Б.В.1.1-36:2016. Визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою України 2016.

26 ДБН В.1.1-7:2016. Державні будівельні норми України. Пожежна безпека об'єктів будівництва. Загальні положення. – Чинний від 01.06.2017 р.

27 ДБН В.2.5-56-2015. Системи протипожежного захисту. –Чинний від 01.07.2015.

28 Закон України «Про охорону навколишнього природного середовища» від 26.06.1991 р. №1264-ХІІ у редакції Редакція від 18.12.2019, підстава - 139-ІХ

29 Стеблюк М. І. Цивільна оборона : підруч., 3-тє вид., перероб. і доп./ М. І. Стеблюк.– К.: Знання, 2004. – 490 с.

30 Статут дій органів управління та підрозділів оперативно- рятувальної служби цивільного захисту під час гасіння пожеж – Міністерство юстиції України, від 10 липня 2018 р. № 802/32254

31 Кодекс цивільного захисту України – ВРУ №5403-VІ, від 2.10.2012

ДОДАТОК А

Лістинг програмного коду

A.1 IFileService.cs

```
using FileSharing.Common.Models;
using System.IO;

namespace FileSharing.BL.Interfaces
{
    public interface IFileService
    {
        string Upload(Stream fileContent, string fileName,
string contentType);

        FileModel Download(string fileKey);
    }
}
```

A.2 FileHelper.cs

```
using FileSharing.Common.Interfaces;
using System.IO;

namespace FileSharing.Common.Helpers
{
    public class FileHelper : IFileHelper
    {
        public void Save(Stream file, string path)
        {
            using (FileStream fileStream = File.Create(path))
            {
                file.CopyTo(fileStream);
            }
        }

        public byte[] Get(string path)
        {
            if (File.Exists(path))
            {
                return File.ReadAllBytes(path);
            }
            else
            {
                throw new FileNotFoundException($"File on the
path {path} not found");
            }
        }
    }
}
```

```
}
```

A.3 UrlGenerator.cs

```
using FileSharing.Common.Interfaces;
using System;

namespace FileSharing.Common.Helpers
{
    public class UrlGenerator : IUrlGenerator
    {
        public (string url, string key) CreateUrl(string
startPath)
        {
            var startUri = new Uri(startPath);

            string uniqueUrlPart = Guid.NewGuid().ToString();
            Uri.TryCreate(startUri, uniqueUrlPart, out Uri
resultUri);

            return (resultUri.ToString(), uniqueUrlPart);
        }
    }
}
```

A.4 FileInfoFluentApiConfiguration.cs

```
using FileSharing.DAL.Entities.Concrete;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace FileSharing.DAL.Implementation.FluentApiConfigurations
{
    public static class FileInfoFluentApiConfiguration
    {
        public static void
TableConfiguration(EntityTypeBuilder<FileInfoEntity> entity)
        {
            entity.HasKey(e => e.Id);

            entity.Property(e => e.Id)
                .ValueGeneratedOnAdd();

            entity.HasIndex(e => e.Id);

            entity.Property(e => e.Name)
                .HasMaxLength(100);

            entity.Property(e => e.Path)
                .HasMaxLength(260)
                .IsRequired();

            entity.Property(e => e.Url)
```



```

        .HasMaxLength(2000)
        .IsRequired();

entity.Property(e => e.Key)
    .HasMaxLength(40)
    .IsRequired();

entity.Property(e => e.ContentType)
    .HasMaxLength(100)
    .IsRequired();
    }
}
}

```

A.5 GenericRepository.cs

```

using FileSharing.DAL.Entities.Interfaces;
using FileSharing.DAL.Interfaces.Repositories;
using Microsoft.EntityFrameworkCore;
using System;
using System.Linq;
using System.Linq.Expressions;

namespace FileSharing.DAL.Implementation.Repositories
{
    public class GenericRepository<TEntity> :
    IRepository<TEntity>
        where TEntity : class, IEntity
    {
        private readonly LndContext _context;
        private readonly DbSet<TEntity> _dbSet;
        private readonly IConfigurationProvider
        _configurationProvider;
        protected readonly IEntityFilterProvider<TEntity>
        _entityFilterProvider;

        public GenericRepository(
            LndContext context,
            IMapper mapper,
            IEntityFilterProvider<TEntity> entityFilterProvider)
        {
            _context = context;
            _entityFilterProvider = entityFilterProvider;
            _configurationProvider = mapper.ConfigurationProvider;
            _dbSet = _context.Set<TEntity>();
        }

        public virtual async Task<IReadOnlyCollection<TEntity>>
        GetAllAsync() =>
            await _dbSet
                .AsNoTracking()
                .ToListAsync()

```

```

        .ConfigureAwait(false);

        public virtual async
        Task<IReadOnlyCollection<TCompositeEntity>>
        GetAllCompositeAsync<TCompositeEntity>() =>
            await _dbSet
                .AsNoTracking()
                .ProjectTo<TCompositeEntity>(_configurationProvider)
                .ToListAsync()
                .ConfigureAwait(false);

        public virtual Task<TEntity> GetByUniqueIdAsync(Guid uniqueId)
=>
            _dbSet.FirstOrDefaultAsync(t => t.UniqueId == uniqueId);

        public virtual Task<TEntity> GetByUniqueIdAsync(
            Guid uniqueId,
            ISpecification<TEntity> specification) =>
            _dbSet
                .ApplySpecification(specification)
                .FirstOrDefaultAsync(t => t.UniqueId == uniqueId);

        public virtual Task<TCompositeEntity>
        GetCompositeByUniqueIdAsync<TCompositeEntity>(
            Guid uniqueId)
            where TCompositeEntity : class =>
            _dbSet
                .Where(t => t.UniqueId == uniqueId)
                .ProjectTo<TCompositeEntity>(_configurationProvider)
                .FirstOrDefault();

        public virtual Task<TCompositeEntity>
        GetCompositeFirstOrDefaultAsync<TCompositeEntity>(
            ISpecification<TEntity> specification)
            where TCompositeEntity : class =>
            _dbSet
                .ApplySpecification(specification)
                .ProjectTo<TCompositeEntity>(_configurationProvider)
                .FirstOrDefault();

        public virtual Task<TCompositeEntity>
        GetCompositeFirstOrDefaultAsync<TCompositeEntity>(
            Expression<Func<TEntity, bool>> predicate)
            where TCompositeEntity : class =>
            _dbSet
                .Where(predicate)
                .ProjectTo<TCompositeEntity>(_configurationProvider)
                .FirstOrDefault();

        public virtual async Task<IReadOnlyCollection<TEntity>>
        GetAsync(
            Expression<Func<TEntity, bool>> predicate) =>

```

```

        await _dbSet
            .Where(predicate)
            .AsNoTracking()
            .ToListAsync()
            .ConfigureAwait(false);

        public virtual async Task<IReadOnlyCollection<TEntity>>
        GetAsync(
            ISpecification<TEntity> specification) =>
            await _dbSet
                .ApplySpecification(specification)
                .AsNoTracking()
                .ToListAsync()
                .ConfigureAwait(false);

        public virtual async
        Task<IReadOnlyCollection<TCompositeEntity>>
        GetCompositeAsync<TCompositeEntity>(
            Expression<Func<TEntity, bool>> predicate)
            where TCompositeEntity : class =>
            await _dbSet
                .Where(predicate)
                .ProjectTo<TCompositeEntity>(_configurationProvider)
                .AsNoTracking()
                .ToListAsync()
                .ConfigureAwait(false);

        public virtual async
        Task<IReadOnlyCollection<TCompositeEntity>>
        GetCompositeAsync<TCompositeEntity>(
            ISpecification<TEntity> specification)
            where TCompositeEntity : class =>
            await _dbSet
                .ApplySpecification(specification)
                .ProjectTo<TCompositeEntity>(_configurationProvider)
                .AsNoTracking()
                .ToListAsync()
                .ConfigureAwait(false);

        public virtual Task<TProperty>
        GetPropertyValueAsync<TProperty>(
            Expression<Func<TEntity, TProperty>> propertyExpression,
            Expression<Func<TEntity, bool>> whereExpression = null,
            ISpecification<TEntity> specification = null)
        {
            IQueryable<TEntity> entities = _dbSet
                .ApplySpecification(specification);
            if (whereExpression != null)
            {
                entities = entities
                    .Where(whereExpression);
            }
        }

```

```

        return entities
            .Select(propertyExpression)
            .FirstOrDefaultAsync();
    }

    public virtual Task<TProperty>
    GetPropertyValueAsync<TProperty>(
        ISpecification<TEntity> specification,
        Expression<Func<TEntity, TProperty>> propertyExpression) =>
        _dbSet
            .ApplySpecification(specification)
            .Select(propertyExpression)
            .FirstOrDefaultAsync();

    public virtual async Task<IReadOnlyCollection<TProperty>>
    GetPropertyValuesAsync<TProperty>(
        Expression<Func<TEntity, bool>> whereExpression,
        Expression<Func<TEntity, TProperty>> propertyExpression) =>
        await _dbSet
            .Where(whereExpression)
            .Select(propertyExpression)
            .ToListAsync()
            .ConfigureAwait(false);

    public virtual Task<TEntity> FirstOrDefaultAsync(
        Expression<Func<TEntity, bool>> condition = null,
        ISpecification<TEntity> specification = null)
    {
        IQueryable<TEntity> entities = _dbSet
            .ApplySpecification(specification);

        return condition is null
            ? entities.FirstOrDefaultAsync()
            : entities.FirstOrDefaultAsync(condition);
    }

    public virtual Task CreateAsync(TEntity entity)
    {
        _dbSet.Add(entity);

        return Task.CompletedTask;
    }

    public virtual Task CreateRangeAsync(IEnumerable<TEntity>
entities)
    {
        _dbSet.AddRange(entities);

        return Task.CompletedTask;
    }

```

```

public virtual Task UpdateAsync(TEntity entity)
{
    TEntity updatedEntity = _dbSet.Find(entity.Id);

    Contract.Requires<ArgumentNullException>(
        updatedEntity != null,
        $"There is no {typeof(TEntity).Name} entity with UniqueId
{entity.UniqueId} in the database.");

    _dbSet.Update(entity);

    return Task.CompletedTask;
}

public virtual Task UpdateRangeAsync(IEnumerable<TEntity>
entities)
{
    _dbSet.UpdateRange(entities);

    return Task.CompletedTask;
}

public virtual async Task SoftDeleteAsync<TDeletableEntity>(
    Guid uniqueId)
    where TDeletableEntity : class, IEntity, IDeletableEntity
{
    DbSet<TDeletableEntity> dbSet =
_context.Set<TDeletableEntity>();

    TDeletableEntity softDeletedEntity = await dbSet
        .FirstOrDefaultAsync(x => x.UniqueId == uniqueId);

    Contract.Requires<ArgumentNullException>(
        softDeletedEntity != null,
        $"There is no {typeof(TDeletableEntity).Name} entity with
Id {uniqueId} in the database.");

    softDeletedEntity.IsDeleted = true;
    dbSet.Update(softDeletedEntity);
}

public virtual async Task DeleteAsync(Guid uniqueId)
{
    TEntity deletedEntity = await GetByIdAsync(uniqueId)
        .ConfigureAwait(false);

    Contract.Requires<ArgumentNullException>(
        deletedEntity != null,
        $"There is no {typeof(TEntity).Name} entity with Id
{uniqueId} in the database.");

    _dbSet.Remove(deletedEntity);
}

```

```

    }

    public virtual Task<bool> AnyAsync(
        Expression<Func<TEntity, bool>> condition = null,
        ISpecification<TEntity> specification = null)
    {
        IQueryable<TEntity> entities = _dbSet
            .ApplySpecification(specification);

        return condition is null
            ? entities.AnyAsync()
            : entities.AnyAsync(condition);
    }

    public virtual Task<int> CountAsync(Expression<Func<TEntity,
bool>> condition) =>
        _dbSet.CountAsync(condition);

    public virtual void AttachRange(IEnumerable<TEntity> entities)
=>
        _dbSet.AttachRange(entities);

    public virtual Task<FilteredResponseModel<TEntity>>
GetFilterModelAsync(
    QueryFilterRequestModel request,
    ISpecification<TEntity> specification) =>
        _entityFilterProvider
            .BuildResponseModelAsync(
                request,
                () => GetFilterListAsync(_dbSet, request,
_entityFilterProvider, specification),
                () => GetFilterCountAsync(_dbSet, request,
_entityFilterProvider, specification));
}

    public virtual Task<FilteredResponseModel<TCompositeEntity>>
GetFilterModelAsync<TCompositeEntity>(
        IEntityFilterProvider<TCompositeEntity>
entityFilterProvider,
        QueryFilterRequestModel request,
        ISpecification<TEntity> specification = null)
    where TCompositeEntity : class
    {
        IQueryable<TCompositeEntity> compositeEntities = _dbSet
            .ApplySpecification(specification)
            .ProjectTo<TCompositeEntity>(_configurationProvider);

        return entityFilterProvider
            .BuildResponseModelAsync(
                request,
                () => GetFilterListAsync(compositeEntities, request,
entityFilterProvider),

```

```

        () => GetFilterCountAsync(compositeEntities, request,
entityFilterProvider));
    }

```

```

    public virtual async
Task<IReadOnlyCollection<TCompositeEntity>>
GetFilteredCollectionAsync<TCompositeEntity>(
    QueryFilterRequestModel request,
    IEntityFilterProvider<TCompositeEntity>
entityFilterProvider,
    ISpecification<TEntity> specification = null,
    bool isApplyPagination = true)
    where TCompositeEntity : class
    {
        IQueryable<TCompositeEntity> compositeEntities = _dbSet
            .ApplySpecification(specification)
            .ProjectTo<TCompositeEntity>(_configurationProvider);

        IReadOnlyCollection<TCompositeEntity> filteredEntities =
await compositeEntities
    .ApplyFilter(entityFilterProvider, request,
isApplyPagination)
    .AsNoTracking()
    .ToListAsync()
    .ConfigureAwait(false);

        return filteredEntities;
    }

```

```

    public virtual async Task<IReadOnlyCollection<TEntity>>
GetFilteredCollectionAsync(
    QueryFilterRequestModel request,
    ISpecification<TEntity> specification = null,
    bool isApplyPagination = true)
    {
        IReadOnlyCollection<TEntity> filteredEntities = await _dbSet
            .ApplySpecification(specification)
            .ApplyFilter(_entityFilterProvider, request,
isApplyPagination)
            .AsNoTracking()
            .ToListAsync()
            .ConfigureAwait(false);

        return filteredEntities;
    }

```

```

    protected virtual async
Task<IReadOnlyCollection<TFilterableEntity>>
GetFilterListAsync<TFilterableEntity>(
    IQueryable<TFilterableEntity> query,
    QueryFilterRequestModel request,

```

```

        IEntityTypeFilterProvider<TFilterableEntity>
entityFilterProvider,
        ISpecification<TFilterableEntity> specification = null)
    where TFilterableEntity : class =>
    await query
        .ApplySpecification(specification)
        .ApplyFilter(entityFilterProvider, request)
        .AsNoTracking()
        .ToListAsync()
        .ConfigureAwait(false);

    protected virtual Task<int>
GetFilterCountAsync<TFilterableEntity>(
    IQueryable<TFilterableEntity> query,
    QueryFilterRequestModel request,
    IEntityTypeFilterProvider<TFilterableEntity>
entityFilterProvider,
    ISpecification<TFilterableEntity> specification = null)
    where TFilterableEntity : class =>
    query
        .ApplySpecification(specification)
        .ApplyFilter(entityFilterProvider, request, false)
        .CountAsync();
}
}

```

A.6 FileSharingContext.cs

```

using FileSharing.DAL.Entities.Concrete;
using FileSharing.DAL.Implementation.FluentApiConfigurations;
using Microsoft.EntityFrameworkCore;

namespace FileSharing.DAL.Implementation
{
    public sealed class FileSharingContext : DbContext
    {
        public DbSet<FileInfoEntity> Files { get; set; }

        public
FileSharingContext(DbContextOptions<FileSharingContext> options)
            : base(options)
        {
            Database.EnsureCreated();
        }

        protected override void OnModelCreating(ModelBuilder
modelBuilder)
        {
            FileInfoFluentApiConfiguration.TableConfiguration(modelBuilde
r.Entity<FileInfoEntity>());

```



```

    }
}

```

A.7 UnitOfWork.cs

```

using FileSharing.DAL.Entities.Concrete;
using FileSharing.DAL.Interfaces;
using FileSharing.DAL.Interfaces.Repositories;
using System;

namespace FileSharing.DAL.Implementation
{
    public class UnitOfWork : IUnitOfWork
    {
        private readonly FileSharingContext _context;

        private readonly Lazy<IRepository<FileInfoEntity>>
        _fileInfoRepository;

        public UnitOfWork(
            FileSharingContext context,
            Lazy<IRepository<FileInfoEntity>>
fileInfoRepository)
        {
            _context = context;
            _fileInfoRepository = fileInfoRepository;
        }

        public IRepository<FileInfoEntity> FileInfoRepository =>
        _fileInfoRepository.Value;

        public void SaveChanges()
        {
            _context.SaveChanges();
        }
    }
}

```

A.8 FileSharing.Infrastructure.cs

```

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>netstandard2.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference
Include="Microsoft.EntityFrameworkCore.SqlServer" Version="3.1.3"
/>

```

```

    <PackageReference
Include="Microsoft.Extensions.Configuration.Abstractions"
Version="3.1.3" />
    <PackageReference
Include="Microsoft.Extensions.DependencyInjection.Abstractions"
Version="3.1.3" />
  </ItemGroup>

  <ItemGroup>
    <ProjectReference
Include="..\FileSharing.BL.Interfaces\FileSharing.BL.Interfaces.csproj" />
    <ProjectReference
Include="..\FileSharing.BL.Services\FileSharing.BL.Services.csproj" />
    <ProjectReference
Include="..\FileSharing.Common\FileSharing.Common.csproj" />
    <ProjectReference
Include="..\FileSharing.DAL.Implementation\FileSharing.DAL.Implementation.csproj" />
  </ItemGroup>
</Project>

```

A.9 DependencyInjectionConfiguration.cs

```

using FileSharing.BL.Interfaces;
using FileSharing.BL.Services;
using FileSharing.Common.Helpers;
using FileSharing.Common.Interfaces;
using FileSharing.DAL.Implementation;
using FileSharing.DAL.Implementation.Repositories;
using FileSharing.DAL.Interfaces;
using FileSharing.DAL.Interfaces.Repositories;
using FileSharing.Infrastructure.Helpers;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using System;

namespace FileSharing.Infrastructure.Configurations
{
    public static class DependencyInjectionConfiguration
    {
        public static void RegisterServices(
            this IServiceCollection services,
            IConfiguration configuration)
        {
            string connectionString =
configuration.GetConnectionString("FileSharingConnection");

```

```

        services.AddDbContextPool<FileSharingContext>(options =>
options.UseSqlServer(connectionString));
        services.AddSingleton<Func<FileSharingContext>>>(
            provider => () => new FileSharingContext(
                new
DbContextOptionsBuilder<FileSharingContext>()

                .UseSqlServer(connectionString).Options));

        services.AddScoped<IUnitOfWork, UnitOfWork>();
        services.AddTransient(typeof(Lazy<>),
typeof(Lazier<>));
        services.AddScoped(typeof(IRepository<>),
typeof(GenericRepository<>));

        services.AddScoped<IFileHelper, FileHelper>();
        services.AddScoped<IUrlGenerator, UrlGenerator>();
        services.AddScoped<IFileService, FileService>();
    }

}

```