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 та створення методів шифрування повідомлень користувача веб-сервісу для інтернет листування

Виконав студент 6 курсу, групи КІТ-М119б

Марценюк Б.В.
(підпис, прізвище та ініціали)

Керівник Філоненко А.М.
(підпис, прізвище та ініціали)

Рецензент Григоренко І.В.
(підпис, прізвище та ініціали)

Нормоконтроль Філоненко А.М.
(підпис, прізвище та ініціали)

Харків 2020

ЗМІСТ

ПЕРЕЛІК ПОЗНАК ТА СКОРОЧЕНЬ

DES	– Data Encryption Standart
AES	– Advanced Encryption Standard
TLS	– Transport layer security
RSA	– Rivest, Shamir и Adleman
GSM	– Groupe Spcial Mobile
E2EE	– End-to-end encryption
MITM	– Man in the middle
SSL	– Secure Sockets Layer
HTTP	– Hyper Text Transfer Protocol
HPKP	– HTTP Public Key Pinning
DNS	– Domain Name System
БД	– Бази даних
JSP	– Java Server Pages
JSF	– Java Server Faces
AOT	– Ahead-of-Time
RMI	– Remote Method Invocation
SQL	– Structured Query Language
JDBC	– Java DataBase Connectivity
MVC	– Model-View-Controller
СУБД	– Система управління базами даних
DAO	– Data access object

ВСТУП

На сьогодні в інформаційному просторі, швидкими темпами впроваджуються новітні досягнення комп'ютерних і телекомунікаційних технологій. Комп'ютерні системи активно впроваджуються у фінансові, промислові, торгові і соціальні сфери. Внаслідок цього різко зріс інтерес широкого кола користувачів до проблем захисту інформації. Захист інформації - це сукупність організаційно-технічних заходів і правових норм для попередження заподіяння збитку інтересам власника інформації. В останні роки з розвитком комерційної і підприємницької діяльності збільшилося число спроб несанкціонованого доступу до конфіденційної інформації.

Серед всього спектру методів захисту даних від небажаного доступу особливе місце займають криптографічні методи. Криптографія - наука про математичні методи забезпечення конфіденційності і автентичності інформації. Для сучасної криптографії характерне використання відкритих алгоритмів шифрування, що припускають використання обчислювальних засобів.

Витік комерційної інформації призводить до небажаних наслідків. Збитки від діяльності конкурентів, що використовують методи шпигунства є дуже великими. Таким чином, задачі безпеки будь-яких видів доводиться вирішувати щораз при розгляді всіляких аспектів людської діяльності. Але, як бачимо, всі види безпеки тісно пов'язані з інформаційною безпекою і, більш того, їх неможливо забезпечити без забезпечення.

У сучасному інформаційному суспільстві велику загрозу конфіденційності та цілісності інформації представляє кіберзлочинність. Зростання кількості кібератак та доступність програмно-технічних засобів для їх реалізації зумовлює необхідність розробки сучасних засобів інформаційної безпеки громадян та держави в цілому.

Широке впровадження інформаційних технологій робить закономірною та актуальною проблему захисту інформації. Дослідження показують, що лише

половина фахівців з інформаційної безпеки вважають свою компанію чи установу такою, що готова протистояти сучасним інформаційним загрозам, зокрема і таким, що можуть призвести до неконтрольованого поширення інформації за межі інформаційних систем, у яких вона обробляється.

Метою дипломної роботи є дослідження методів шифрування повідомлень веб-сервісу інтернет листування та впровадження вдосконаленого методу шифрування до створюваного веб-сервісу. На основі мети були поставлені задачі:

- провести дослідження існуючих методів шифрування, виділити основні переваги та недоліки кожного;
- провести аналіз існуючих методів хешування;
- описати основні способи злому хешей;
- провести аналіз використовуваних технологій та методів розробки;
- розробити сласну модель веб-сервісу інтернет листування;
- запровадити до нього вдосконалений метод шифрування;
- провести економічне обґрунтування доцільності створюваного програмного продукту.

1 ДОСЛІДЖЕННЯ МЕТОДІВ ШИФРУВАННЯ ТА ХЕШУВАННЯ

Інформація, що може бути прочитана, осмислена і зрозуміла без яких-небудь спеціальних мір, називається відкритим текстом (plaintext, clear text). Метод перекручування відкритого тексту таким чином, щоб сховати його суть, називається шифруванням (encryption або enciphering). Шифрування відкритого тексту приводить до його перетворення в незрозумілу абракадабру, іменовану шифртекстом (ciphertext). Шифрування дозволяє сховати інформацію від тих, для кого вона не призначається, незважаючи на те, що вони можуть бачити сам шифртекст. Протилежний процес по звертанню шифртекста в його вихідний вид називається розшифруванням (decryption або deciphering).

Криптографічна стійкість виміряється тим, скільки знадобиться часу і ресурсів, щоб із шифртекста відновити вихідний відкритий текст. Результатом стійкої криптографії є шифртекст, що винятково складно зламати без володіння визначеними інструментами по дешифруванню. Використовуючи весь обчислювальний потенціал сучасної цивілізації - навіть мільярд комп'ютерів, що виконують мільярд операцій у секунду - неможливо дешифрувати результат стійкої криптографії. Хтось може вирішити, що стійка криптографія зможе устояти навіть проти самого серйозного криптоаналітика. Ніким не доведене, що краще шифрування, доступне сьогодні, зможе вистояти проти обчислювальних можливостей комп'ютерів, доступних завтра [1].

Криптографічний алгоритм, або шифр, - це математична формула, що описує процеси шифрування і розшифрування. Щоб зашифрувати відкритий текст, криптоалгоритм працює в сполученні з ключем - словом, числом або фразою. Те саме повідомлення одним алгоритмом, але різними ключами буде перетворюватися в різний шифртекст. Захищеність шифртекста цілком залежить від двох речей: стійкості криптоалгоритму і таємності ключа. Криптоалгоритм плюс усілякі ключі і протоколи, що приводять їх у дію, складають криптосистему. У традиційній криптографії, також називаної шифруванням

таємним, або симетричним, той самий ключ використовується як для шифрування, так і для розшифрування даних. Data Encryption Standart (DES) - приклад симетричного алгоритму, що широко застосовувався на Заході з 70-х років у банківській і комерційних сферах [2]. В даний час його переміняє Advanced Encryption Standard (AES).

Симетричне шифрування має ряд переваг. Перше - швидкість криптографічних операцій. Воно особливо корисно для шифрування даних, що залишаються у вас. Однак, симетричне шифрування, використане саме по собі як засіб захисту коштовних даних, що пересилаються, може виявитися досить витратним просто через складність передачі таємного ключа. Для встановлення криптографічного зв'язку за допомогою симетричного алгоритму, відправникові й одержувачеві потрібно попередньо погодити ключ і тримати його в таємниці. Якщо вони знаходяться в географічно вилучених місцях, то повинні вдатися до допомоги довіреного посередника, наприклад, надійного кур'єра, щоб уникнути компрометації ключа в ході транспортування. Зловмисник, що перехопив ключ на шляху, зможе пізніше читати, змінювати і підробляти будь-яку інформацію, зашифровану або завірену цим ключем. Глобальна проблема симетричних шифрів складається в складності керування ключами.

1.1 Симетричне шифрування

Симетричне шифрування - це спосіб шифрування даних, при якому один і той же ключ використовується і для кодування, і для відновлення інформації [2]. До 1970-х років, коли з'явилися перші асиметричні шифри, воно було єдиним криптографічним методом. Схема симетричного шифрування зображено на рис. 1.1.

Рисунок 1.1 – Схема симетричного шифрування

В цілому симетричним вважається будь-який шифр, який використовує один і той же секретний ключ для шифрування і дешифрування.

Наприклад, якщо алгоритм передбачає заміну букв числами, то і у відправника повідомлення, і у його одержувача повинна бути одна і та ж таблиця відповідності букв і чисел: перший з її допомогою шифрує повідомлення, а другий - розшифровує.

Однак такі найпростіші шифри легко зламати - наприклад, знаючи частотність різних букв в мові, можна співвідносити найчастіші букви з найчисельнішими числами або символами в коді, поки не вдасться отримати осмислені слова. З використанням комп'ютерних технологій таке завдання стало займати настільки мало часу, що використання подібних алгоритмів втратило будь-який сенс.

Тому сучасні симетричні алгоритми вважаються надійними, якщо відповідають наступним вимогам:

- вихідні дані не повинні містити статистичних патернів вихідних даних (найбільш частотні символи осмисленого тексту не повинні відповідати найбільш частотним символам шифру);

- шифр повинен бути нелінійним (тобто в шифрованих даних не повинно бути закономірностей, які можна відстежити, маючи на руках кілька відкритих текстів і шифрів до них).

Більшість актуальних симетричних шифрів для досягнення результатів, які відповідають цим вимогам, використовують комбінацію операцій підстановки (заміна фрагментів вихідного повідомлення, наприклад букв, на інші дані, наприклад цифри, за певним правилом або за допомогою таблиці відповідностей) і перестановки (перемішування частин вихідного повідомлення за певним правилом), по черзі повторюючи їх. Одне коло шифрування, що складається з цих операцій, називається раундом.

Залежно від принципу роботи алгоритми симетричного шифрування діляться на два типи:

- блокові;

- потокові.

Блокові алгоритми шифрують дані блоками фіксованої довжини (64, 128 або іншу кількість біт в залежності від алгоритму). Якщо все повідомлення або його фінальна частина менше розміру блоку, система доповнює його передбаченими алгоритмом символами, які так і називаються доповненням.

До актуальних блокових алгоритмів відносяться:

- AES;

- RC5;

- Blowfish;

- Twofish.

Потокове шифрування даних передбачає обробку кожного біта інформації з використанням гамування, тобто зміни цього біта за допомогою відповідного йому біта псевдовипадкової секретної послідовності чисел, яка формується на основі ключа і має ту ж довжину, що і шифрованих повідомлення. Як правило, біти вихідних даних порівнюються з бітами секретної послідовності за

допомогою логічної операції XOR (виключає АБО, на виході дає 0, якщо значення бітів збігаються, і 1, якщо вони різняться).

Потокове шифрування в даний час використовують такі алгоритми:

- RC4;
- Salsa20;
- HC-256;
- WAKE.

Симетричні алгоритми вимагають менше ресурсів і демонструють більшу швидкість шифрування, ніж асиметричні алгоритми. Більшість симетричних шифрів імовірно стійко до атак за допомогою квантових комп'ютерів, які в теорії становлять загрозу для асиметричних алгоритмів.

Слабке місце симетричного шифрування - обмін ключем. Оскільки для роботи алгоритму ключ повинен бути і у відправника, і у одержувача повідомлення, його необхідно передати; однак при передачі по незахищених каналах його можуть перехопити і використовувати сторонні. На практиці в багатьох системах ця проблема вирішується шифруванням ключа за допомогою асиметричного алгоритму.

Симетричне шифрування використовується для обміну даними в багатьох сучасних сервісах, часто в поєднанні з асиметричним шифруванням. Наприклад, месенджери захищають за допомогою таких шифрів листування (при цьому ключ для симетричного шифрування зазвичай доставляється в асиметрично зашифрованому вигляді), а сервіси для відеозв'язку - потоки аудіо і відео. В захищеному транспортному протоколі TLS симетричне шифрування використовується для забезпечення конфіденційності даних.

Симетричні алгоритми не можуть застосовуватися для формування цифрових підписів і сертифікатів, тому що секретний ключ при використанні цього методу повинен бути відомий всім, хто працює з шифром, що суперечить самій ідеї електронного підпису (можливості перевірки її справжності без звернення до власника).

1.2 Асиметричне шифрування

Асиметричне шифрування засноване на парах чисел. Одне з цих чисел - відкритий ключ, який доступний всім. За допомогою цього числа хто завгодно може зашифрувати повідомлення. Але розшифрувати його за допомогою цього ж числа не вийде.

Для дешифровки беруть друге число - закритий ключ. Він повинен бути секретним.

Це не можуть бути два випадкових ключа. Відкритий і закритий ключ завжди пов'язані між собою алгоритмом, який їх видає. Сенс в тому, що всередині цього алгоритму є третя, теж секретне, число, яке пов'язане з обома ключами.

Найпростіший спосіб встановити такий зв'язок - взяти два великих простих числа і перемножити їх. Ми отримаємо ще більше число, яке і буде лежати в основі нашого алгоритму. А всередині цього алгоритму буде така математика, яка залежить від розкладання чисел на множники. Якщо ми не знаємо ні одне з початкових простих чисел, то розкласти на множники таке величезне число буде дуже складним завданням. Схема асиметричного шифрування зображено на рис. 1.2.

Рисунок 1.1 – Схема асиметричного шифрування

Всі асиметричні алгоритми діляться по виду математичної задачі, на якій будується шифр. Завдання має бути складною не для людини, а з точки зору математики - тієї, яку складно вирішити навіть з потужним комп'ютером.

Розкладання великої кількості на множники, яке лежить в основі алгоритму RSA, - нескладне завдання, якщо таких множників всього два. Наприклад, якщо взяти число 45, то воно легко розкладається на множники так:

45 ділиться на 2? Ні.

45 ділиться на 3? Так, виходить 15.

15 ділиться на 2? Ні.

15 ділиться на 3? Так, виходить 5.

5 ділиться на 2, 3 або 4? Ні, залишається тільки саме число 5.

Виходить 45 = 5 × 3 × 3.

Якщо ми візьмемо число 474 689, то таких перевірок і дій нам потрібно буде виконати рівно 479, тому що 474 689 = 479 × 991. Для людини це вже складніше, але комп'ютер це зробить так само швидко, як і в випадку з числом 45.

А ось для числа з 617 знаків - саме стільки застосовується в алгоритмі RSA-2048 - буде потрібно сотні років машинних обчислень, щоб розкласти його на два множники.

Перевага таких алгоритмів в тому, що для передачі зашифрованих повідомлень можна використовувати відкритий канал зв'язку. Навіть якщо зловмисник перехопить повідомлення, він не зможе прочитати його без секретного ключа. Але щоб все було саме так, потрібно, щоб ключ був досить довгий - 1024 біт і вище.

Недолік асиметричного шифрування очевидний - воно працює тільки в одну сторону. Щоб таке спілкування було двостороннім, кожен повинен надати іншому свій відкритий ключ [3].

Асиметричне шифрування застосовують в двох випадках:

- Коли потрібно встановити захищений канал зв'язку в інтернеті. Ми знаємо, що спочатку наші дані може перехопити хто завгодно, тому нам потрібно обмінюватися тільки тими ключами, які можна показувати іншим.

- Для створення цифрових підписів і сертифікатів. Це дозволяє перевіряти справжність цифрових документів, а також переконатися, що його відправив саме власник сертифіката.

Кращий ефект досягається при комбінації обох видів шифрування. Відбувається це так:

- за допомогою асиметричного алгоритму сервера відсилається сесійний ключ для симетричного шифрування;

- відбувається обмін інформацією по симетричному алгоритму.

Тут можливі варіанти, але загальний сенс зазвичай не змінюється.

1.3 Наскрізне шифрування

Наскрізне шифрування (End-to-end) – являє собою систему зв'язку, де тільки прохідні користувачі можуть читати повідомлення. В принципі, він не дозволяє потенційним перехоплювачам, в тому числі постачальникам телекомунікаційних послуг, інтернет-провайдерам і навіть постачальникам послуг зв'язку, отримати доступ до криптографічних ключів, необхідним для дешифровки розмови. У багатьох системах обміну повідомленнями, включаючи електронну пошту і багато мереж чатів, повідомлення проходять через посередників і зберігаються у третьої сторони, звідки їх отримує одержувач. Навіть якщо повідомлення зашифровані, вони шифруються тільки «при передачі» і, таким чином, доступні постачальнику послуг, незалежно від того, чи використовується шифрування диска на стороні сервера [4].

Шифрування диска на стороні сервера просто запобігає тому, щоб цією інформацією не скористалися неавторизовані користувачі, воно не заважає самій компанії переглядати інформацію, оскільки у них є ключ і вони можуть просто розшифрувати ці дані. Це дозволяє третій стороні надавати пошук і інші функції

або сканувати незаконний і неприйнятний контент, але також означає, що вони можуть бути прочитані і використані будь-якою особою, яка має доступ до збережених повідомлень в сторонній системі, незалежно від того, навмисно чи це зроблено, або через чорний хід. Це може розглядатися як проблема в багатьох випадках, коли конфіденційність дуже важлива, наприклад, компанії, репутація яких залежить від їх здатності захищати дані третіх осіб, переговори і обмін інформацією, які досить важливі, щоб мати ризик цільового «злому» або спостереження, і коли мова йде про делікатних питаннях, таких як здоров'я і інформація про неповнолітніх. Наскрізне шифрування призначене для запобігання читання або таємного зміни даних, крім істинного відправника і одержувача (одержувачів). Повідомлення зашифровані відправником, але третя сторона не має можливостей для їх дешифрування і зберігає їх в зашифрованому вигляді. Одержувачі отримують зашифровані дані і самі розшифровують їх. Оскільки ніякі треті сторони не можуть розшифрувати передані або збережені дані, наприклад, компанії, що використовують наскрізне шифрування, не можуть передавати тексти повідомлень своїх клієнтів владі.

Термін «наскрізне шифрування» спочатку означав тільки те, що повідомлення ніколи не розшифровується під час його передачі від відправника до одержувача. В 2003 році E2EE був запропонований в якості додаткового рівня шифрування для GSM або TETRA на додаток до існуючого шифрування радіозв'язку, що захищає зв'язок між мобільним пристроєм і мережевою інфраструктурою. Це було стандартизовано SFPG для TETRA. Що в TETRA E2EE ключі генеруються центром управління ключами (KMC) або механізмом управління ключами (KMF), а не взаємодіючими користувачами. Пізніше, приблизно в 2014 році, значення терміна «наскрізне шифрування» почало розвиватися, вимагаючи, щоб не тільки передача даних залишалася зашифрованої під час транспортування, але також щоб постачальник послуг зв'язку не міг розшифрувати повідомлення, якщо доступ до закритого ключа або

можливість непомітно ввести зловмисний відкритий ключ в рамках атаки «людина посередині». Це нове значення стало загальноприйнятим.

Наскрізне шифрування забезпечує максимальний рівень захисту для користувачів, які серйозно піклуються про конфіденційність даних. Для управління приватними даними Вам більше не доведеться покладатися на сторонні сервіси, які можуть:

- недостатньо захищати ваші дані, піддаючи ризику злому хакерами, спеціальними службами та іншими особами;

- сканувати ваші дані і обмінюватися отриманою інформацією для отримання вигоди, рекламних цілей і проведення досліджень без вашого відома.

1.3.1 Атаки "людина посередині"

Наскрізне шифрування гарантує безпечну передачу даних між кінцевими точками. Але замість того, щоб намагатися зламати шифрування, перехоплювач може видати себе за одержувача повідомлення (під час обміну ключами або шляхом заміни свого відкритого ключа на одержувача), так що повідомлення шифруються за допомогою ключа, відомого зловмисникові. Після дешифровки повідомлення механізм відстеження може потім зашифрувати його за допомогою ключа, яким вони діляться з фактичним одержувачем, або своїм відкритим ключем в разі асиметричних систем, і знову відправити повідомлення, щоб уникнути виявлення. Це відомо як атака «людина посередині» (MITM).

Всі криптографічні системи, захищені від атак MITM, надають деякий метод автентифікації для повідомлень. Більшість з них вимагає обміну інформацією (наприклад, відкритими ключами) на додаток до повідомлення по захищеному каналу. Такі протоколи, часто використовують протоколи узгодження ключів, були розроблені з різними вимогами безпеки для безпечного каналу, хоча деякі намагалися взагалі скасувати вимогу для будь-якого безпечного каналу. Інфраструктуру відкритого ключа, такі як Transport Layer Security, може затвердіти протокол управління передачею проти MITM атак. У

таких структурах клієнти і сервери обмінюються сертифікатами, які видаються і перевіряються довіреною третьою стороною, званої центром сертифікації (ЦС). Якщо вихідний ключ для автентифікації цього CA ні сам по собі об'єктом атаки MITM, то сертифікати, видані CA, можуть використовуватися для автентифікації повідомлень, відправлених власником цього сертифіката. Використання взаємної автентифікації, при якій і сервер, і клієнт перевіряють зв'язок один з одним, охоплює обидва кінці атаки MITM. Якщо особистість сервера або клієнта не перевірена або визнана недійсною, сеанс завершиться. Однак поведінка за умовчанням більшості підключень полягає в перевірці достовірності тільки сервера, що означає, що взаємна автентифікація не завжди використовується, і атаки MITM все ще можуть відбуватися. Атестації, такі як усні повідомлення про загальне значення (як в ZRTP), або записані свідчення, такі як аудіовізуальні записи хеша відкритого ключа, використовуються для відбиття атак MITM, оскільки візуальні носії набагато складніше і вимагають багато часу імітувати, ніж проста передача пакетів даних. Однак ці методи вимагають участі людини в циклі, щоб успішно ініціювати транзакцію. У корпоративному середовищі успішна перевірка справжності (позначена зеленим замком браузера) не завжди має на увазі безпечне з'єднання з віддаленим сервером. Політика корпоративної безпеки може передбачати додавання настроюються сертифікатів в веб-браузери робочих станцій, щоб мати можливість перевіряти зашифрований трафік. Як наслідок, зелений замок означає, що клієнт успішно пройшов автентифікацію нема на віддаленому сервері, а тільки на корпоративному сервері / проксі, використовуваному для перевірки SSL/TLS. Закріплення відкритого ключа HTTP (HPKP), іноді зване «закріпленням сертифіката», допомагає запобігти атаці MITM, в якій скомпрометований сам центр сертифікації, за рахунок того, що сервер надає список «закріплених» хеш відкритих ключів під час першої транзакції. Для подальших транзакцій потрібно, щоб один або кілька ключів в списку використовувалися сервером для автентифікації цієї транзакції. DNSSEC розширює протокол DNS, щоб використовувати підписи для автентифікації

записів DNS, запобігаючи прості атаки MITM, направляючі клієнта на шкідливий IP-адреса [5].

Перевірка затримки потенційно може виявити атаку в певних ситуаціях, наприклад, при тривалих обчисленнях, які займають десятки секунд, як хеш-функції. Щоб виявити потенційні атаки, сторони перевіряють розбіжності в часі відповіді. Наприклад: припустимо, що двом сторонам зазвичай потрібен певний час для виконання певної транзакції. Однак, якщо однієї транзакції потрібно було багато часу для досягнення іншої сторони, це може вказувати на втручання третьої сторони, що вносить додаткову затримку в транзакцію. Теоретично квантова криптографія забезпечує докази несанкціонованого доступу до транзакцій за допомогою теореми про заборону клонування. Протоколи, засновані на квантової криптографії, зазвичай розпізнають частину або всю свою класичну зв'язку з безумовно безпечної схемою автентифікації, наприклад автентифікація Вегмана-Картера.

1.4 Довжина ключа й захищеність шифрування

Процес злому алгоритму шифрування це, по суті, пошук ключа для доступу до зашифрованих даних в звичайному тексті. Для симетричних алгоритмів, злом зазвичай означає спробу визначити ключ, що використовувався при шифруванні тексту. Для алгоритмів з використанням публічного ключа, процес злому алгоритму означає отримання секретної інформації, що була поширена між двома отримувачами.

Один з методів злому симетричного алгоритму шифрування це просто спробувати кожен відомий ключ у рамках цілого алгоритму, допоки не знайдеться вірний. Для алгоритмів з використанням публічного ключа, так як одна половина ключів є в доступі завчасно, інша половина (приватний ключ) може бути вирахувана використовуючи публічний, через комплекс складних математичних розрахунків. Ручний пошук ключа для злому алгоритму називається методом грубої сили (brute force attack).

Злом алгоритму знайомить з ризиком перехоплення чи навіть розкриття особистості та підробки підтвердження. Також повідомляє нас про ризик перехоплення та розкриття приватної інформації.

Міцність ключа алгоритму вираховується шляхом знаходження найшвидшого методу для злому цього ж алгоритму і порівняння цього способу з методом грубої сили.

Для симетричних ключів, міцність шифрування часто описується в вигляді розміру чи довжини ключів, що використовуються для виконання шифрування: загалом, довші ключі надають кращий рівень захищеності. Довжина ключа вимірюється у бітах. Наприклад, 128-бітні ключі, що використовуються для шифру з симетричними ключами, RC4, що пітримується протоколом SSL надають значно кращий криптографіний захист ніж 40-бітні ключі, що використовуються тим самим шифром. Грубо кажучи, 128-бітне RC4 шифрування в $3x10^{26}$ разів захищенішим, ніж 40-бітне RC4 шифрування. Ключ шифрування вважається повністю захищеним, якщо найкращий відомий метод атаки з метою взлому не є швидшим ніж спроба атаки методом грубої сили, для тестування кожного можливого ключа.

Різні шифри можуть потребувати ключі різноманітної довжни для отримання одного рівня захищеності. Шифр RSA, що використовується для шифрування з використанням публічного ключа, наприклад, може використовувати лише підмножину всіх можливих значень для ключа даної довжини, все через природу математичної проблеми(задачі), на якій він базується. Інші шифри, такі як ті, що використовуються для симетричного шифрування, можуть використовувати всі можливі значення для ключа даної довжини, а не підмножину цих значень.

Так як, зламати ключ шифрування RSA є досить незначним завданням, то шифрування RSA з використанням публічного ключа має використовувати дуже довгі ключі, як мінімум 1024 біти, щоб бути впевненим, що рівень захисту досить високий, з криптографічної точки зору. Але з іншого боку, шифрування з

використанням симетричного ключа може отримати приблизно той же рівень захисту з використанням всього лише 80-бітного ключа для більшості алгоритмів.

1.5 Аналіз відомих методів хешування

Одна з основних відмінностей хешування від шифрування полягає у тому, що: хешування – процес незворотній. Це означає, що, маючи хеш деякої сутності, неможливо відновити саму сутність. Або простою мовою: не можна отримати вихідний пароль при наявності його хешу.

Алгоритм хешування перетворює вихідний рядок в інший рядок фіксованого розміру, який можна розглядати як її "відбиток пальця" - єдиний і неповторний, що належить тільки цьому рядку. Це відмінний захист для паролів. Навіть якщо база даних з паролями раптом буде зламана, зловмисник не зможе дістати самі паролі: йому будуть доступні тільки хеши.

Ось стандартні дії при реєстрації / автентифікації:

- користувач створює обліковий запис;
- пароль проходить через хеш-функцію і записується в базу даних;
- коли користувач намагається увійти в обліковий запис, введений ним пароль проходить через хеш-функцію і порівнюється з хешем, збереженим в базі даних;
- якщо хеш-кодування збігаються, користувач отримує доступ до захищених розділів. В іншому випадку система запитує авторизовані дані знову;
- кроки 3 і 4 повторюються кожен раз, коли користувач проходить авторизацію;

У кроці 4 ні в якому разі не можна повідомляти користувачеві, що було введено невірно: логін або пароль. Потрібно відображати повідомлення загального характеру, наприклад: "Невірні дані".

Приклади поширених хеш-функцій, які використовуються для хешування паролів: SHA256, SHA512, RipeMD, WHIRLPOOL.

Існує безліч способів добування паролів з хешей. Існує кілька простих технік, які допомагають захиститися від більшості атак. Неможливо захиститися на всі 100%, але кожен рівень захисту - додаткова перешкода для хакера.

1.6 Способи злому хешей

У наш час існує велика кількість способів злому хешей, таких як:

- словникові атаки;
- брутфорс;
- таблиці пошуку, обернені таблиці пошуку;
- райдужні таблиці пошуку.

1.6.1 Словникові атаки і брутфорс

Ось найпростіший спосіб зламати хеш. Беремо комбінацію, яка може виявитися паролем, хешуємо її і порівнюємо з хешем, який хочемо розшифрувати. І так до тих пір, поки не буде знайдено збіг.

Існує кілька варіантів реалізувати описаний підхід. Самі часто використовувані: словникова атака (Dictionary Attack) і брутфорс ("метод грубої сили", Brute Force).

Словникові атаки засновані на файлах, в яких містяться деякі слова, фрази, поширені паролі, яка претендує бути паролем. Для кожної комбінації вже підібрано хеш, який порівнюється з хешем, який потрібно зломати. Багато словники побудовані на основі реальних баз даних з паролями користувачів, що підвищує ефективність злому. Деякі словникові атаки є "розумними" і для кожної існуючої комбінації формують набір похідних комбінацій, які можуть також виявитися потрібним паролем.

В процесі брутфорс-атаки "перебираються" довільні комбінації на роль шуканого пароля. Ці атаки дуже дороги в плані обчислювальних ресурсів і не настільки ефективні, як інші види атак, але ними дуже активно користуються [6].

Перевага методу: запобігти цим атаки ніяк не вийде.

Недолік: ефективність цих атак можна звести до мінімуму, якщо організувати правильне управління паролями.

1.6.2 Таблиці пошуку, обернені таблиці пошуку, райдужні таблиці

Таблиці пошуку - неймовірно ефективний метод злому хешей одного і того ж типу. В основі методу лежить ідея підготовки можливих паролів і відповідних їм хешів і їх зберігання в якійсь таблиці (наприклад, в якій-небудь структурі даних). Кращі реалізації таблиць пошуку здатні обробляти сотні комбінацій в секунду, тоді як їх загальна кількість може налічувати кілька мільярдів

Зворотні таблиці пошуку дозволяють хакерам запускати словникові і брутфорс-атаки одночасно для декількох хешів без необхідності попередньої підготовки таблиці пошуку.

Ця атака заснована на цікавому факті: багато користувачів мають однакові паролі. Зловмисник бере зламану базу даних з хешами паролів користувачів, знаходить групи однакових хешів і направляє на них атаку.

Райдужна таблиця — спеціальний варіант таблиць пошуку для звернення криптографічних хеш-функцій, які використовують механізм розумного компромісу між часом пошуку таблицею і займаною пам'яттю. Райдужні таблиці використовуються для розкриття паролів, перетворених за допомогою важкозворотної хеш-функції, а також для атак на симетричні шифри на основі відомого відкритого тексту. Використання функції формування ключа з застосуванням солі робить цю атаку неможливою.

Райдужні таблиці є розвитком ідеї таблиці хеш-ланцюгів. Вони ефективно вирішують проблему колізій шляхом введення послідовності функцій редукції R_1, R_2, …,R_k. Функції редукції застосовуються по черзі, перемежаючись з функцією хешування: H, R_1, H, R_2, …, H, R_k.

При такому підході два ланцюжки можуть злитися тільки за умови збігу значень на одній і тій же ітерації. Отже, достатньо перевіряти на колізії тільки кінцеві значення ланцюжків, що не вимагає додаткової пам'яті [7].

На кінцевому етапі складання таблиці можна знайти всі злиті ланцюжки, залишити з них тільки один і згенерувати нові, щоб заповнити таблицю необхідною кількістю різних ланцюжків. Отримані ланцюжки не є повністю вільними від колізій, тим не менш, вони не зіллються повністю.

Використання послідовностей функцій редукції змінює спосіб пошуку по таблиці. Оскільки хеш може бути знайдений в будь-якому місці ланцюжка, необхідно згенерувати k різних ланцюжків:

- перший ланцюжок будується на припущенні, що шуканий хеш зустрінеться на останній позиції в табличному ланцюжку, тому складається з єдиного значення $R_k(h)$;

- другий ланцюжок будується на припущенні, що шуканий хеш зустрінеться на передостанній позиції в табличному ланцюжку, тому виглядає так: $R_k(H(R_{k-1}(h)))$;

- аналогічно, нарощуючи довжину ланцюжка і застосовуючи функції редукції з меншими номерами, отримуємо інші ланцюжки. Останній ланцюжок буде мати довжину k і містити всі функції редукції: $R_k(H(R_{k-1}(H(\ldots H(R_1(h))\ldots))))$

Також зміниться і визначення помилкового спрацювання: якщо ми неправильно «вгадаємо» позицію шуканого хешу, це буде відкидати тільки один з k згенерованих ланцюжків; при цьому все ще може залишатися можливість знайти вірний хеш для даного табличного ланцюжка, але на іншій позиції.

Хоча райдужні таблиці вимагають відстеження більшої кількості ланцюжків, вони мають велику щільність кількості паролів на один табличний запис. На відміну від таблиці хеш-ланцюжків, застосування декількох функцій редукції зменшує кількість потенційних колізій у таблиці, що дозволяє нарощувати без небезпеки отримати велику кількість злиття ланцюжків.

Один з поширених методів захисту від злому за допомогою райдужних таблиць — використання необоротних хеш-функцій, які включають salt («сіль», «модифікатор»). Існує безліч можливих схем змішування пароля. Наприклад, розглянемо таку функцію для створення хеш від пароля:

хеш = MD5(пароль + сіль)

Для такого відновлення пароля зловмиснику необхідні таблиці для всіх можливих значень солі. При використанні такої схеми, сіль повинна бути досить довгою (6-8 символів), інакше зловмисник може обчислити таблиці для кожного значення солі, випадкової і різною для кожного пароля. Таким чином два однакових пароля будуть мати різні значення хешів, якщо тільки не буде використовуватися однакова сіль.

По суті, сіль збільшує довжину і, можливо, складність пароля. Якщо таблиця розрахована на деяку довжину або на деякий обмежений набір символів, то сіль може запобігти відновленню пароля. Наприклад, у старих Unix-паролях використовувалася сіль, розмір якої становив лише 12 біт. Для злому таких паролів зловмисникові потрібно було порахувати всього 4096 таблиць. Тому в сучасних програмах намагаються використовувати більш довгу сіль. Наприклад, в алгоритмі хешування bcrypt використовується сіль розміром 128 біт. Така довжина солі робить попередні обчислення просто безглуздими. Іншим можливим способом боротьби проти атак, що використовують попередні обчислення, є розтягнення ключа. Наприклад:

ключ = хеш (пароль + сіль)

for 1 to 65536 do

ключ = хеш (ключ + пароль + сіль)

Цей спосіб знижує ефективність застосування попередніх обчислень, так як використання проміжних значень збільшує час, який необхідно для обчислення одного пароля, і тим самим зменшує кількість обчислень, які зловмисник може провести у встановлені часові рамки.

Даний метод застосовується в наступних алгоритмах хешування: MD5, в якому використовується 1000 повторень, і bcrypt. Альтернативним варіантом є використання посилення ключа, який часто приймають за розтягнення ключа. Застосовуючи даний метод, ми збільшуємо розмір ключа за рахунок додавання

випадкової солі, яка потім спокійно видаляється, на відміну від розтягування ключа, коли сіль зберігається і використовується в наступних ітераціях.

2. АНАЛІЗ ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ТА МЕТОДІВ РОЗРОБКИ

2.1 Вибір мови програмування клієнтської та серверної частини

Мов програмування, використовуваних для серверної веб-розробки, досить багато: PHP, Ruby, Java, C, Python, Perl і інші.

З технічної точки зору для більшості проектів немає будь-яких обмежень при виборі мови, тобто практично будь-який функціонал сайту або програми може бути успішно реалізований на будь-якому з них, тому призначення мови не накладає ніяких лімітувань на проект. Діаграма популярності мов програмування зображено на рис. 2.1.

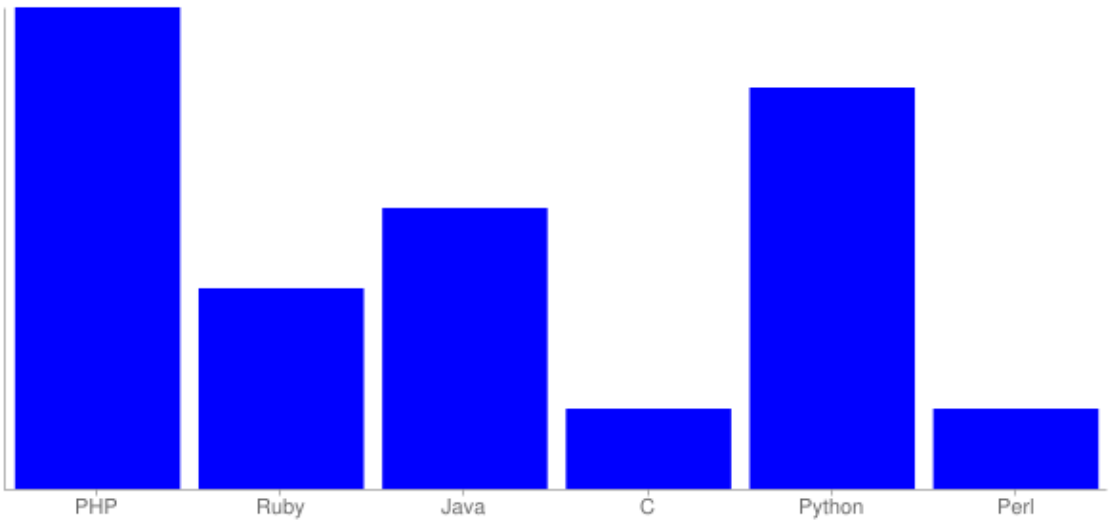

Рисунок 2.1 – Діаграма популярності мов програмування для серверної частини

Для вибору мови програмування розглянуто такі критерії, як наявність об’єктно-орієнтованої парадигми, застосування багатопоточності, багатоплатформенність, статична типізація, швидкість виконання розробленого модуля, зручна робота с БД, оскільки вони є найактуальнішими при виборі мови. Додатковим побажанням є можливість використовувати лише одну мову програмування як для побудування серверної частини, так і для клієнтської частини.

PHP, Rerl, Python, Ruby, звичайно, повністю підходять на роль основної мови розробки, проте у випадку, коли програмний продукт буде використовуватись лише у вигляді веб-сайту. Мова програмування Java надає користувачу можливість створити веб-сервіс інтернет листування на будь-якій платформі, на будь-якій операційній системі, тобто основна частина коду працюватиме будь-де, розробнику необхідно лише створити візуальну частину продукту згідно необхідної платформи. Використовуючи технології JFX можливо створити десктопний додаток, за допомогою технології JSP або JSF продукт легко перетворити у веб-сайт, за допомогою MIDP додаток портується на ОС Android.

Тож враховуючи всі переваги перед іншими мовами, для програмної реалізації було обрано мову програмування Java, оскільки ця мова дозволить швидко та якісно створити готовий продукт.

Програмування на певній мові вимагає наявність відповідного середовища розробки. Для мови Java таким середовищем є САПР IntelliJ IDEA, що призначена для розробки сучасного програмного забезпечення. Дана САПР включає зручний інструментарій для створення програмних додатків різного рівня складності. Також дане середовище дозволяє розробити зручний інтерфейс користувача, динамічно прослідкувати хід виконання програми, надає розробникові можливість навігації по коду [8].

2.2 Огляд мови програмування Java

Мови програмування за класифікацією бувають двох типів:

- мови програмування низького рівня (мова програмування, близька до програмування безпосередньо в машинних кодах використовуваного реального або віртуального процесора;

- мови програмування високого рівня (мова програмування, розроблена для швидкості і зручності використання програмістом. Основна риса високорівневих мов – це абстракція, тобто введення смислових конструкцій,

коротко описують такі структури даних і операції над ними, опис яких на машинному коді дуже довгий і складний для розуміння.

На сьогоднішній момент мова Java є однією з найбільш поширених і популярних мов програмування. Перша версія мови з'явилася ще в 1996 році в надрах компанії Sun Microsystems. Java замислювалась як універсальна мова програмування, яку можна застосовувати для різного роду завдань. Поточною версією є Java 12, яка вийшла в березні 2019 року. Java перетворилася з просто універсальної мови в цілу платформу і екосистему, яка об'єднує різні технології, які використовуються для цілого ряду завдань: від створення десктопних додатків до написання великих веб-порталів і сервісів. Крім того, мова Java активно застосовується для створення програмного забезпечення для цілого ряду пристроїв: звичайних ПК, планшетів, смартфонів і мобільних телефонів і навіть побутової техніки. Досить згадати популярність мобільної ОС Android, більшість програм для якої пишуться саме на Java [9].

У створенні мови програмування Java було п'ять початкових цілей:

- синтаксис мови повинен бути «простим, об'єктно-орієнтовним та звичним»;
- реалізація має бути «безвідмовною та безпечною»;
- повинна зберегтися «незалежність від архітектури та переносність»;
- висока продуктивність виконання;
- мова має бути «інтерпретованою, багатонитевою, із динамічним зв'язуванням модулів» [10].

Під «незалежністю від архітектури» мається на увазі те, що програма, написана на мові Java, працюватиме на будь-якій підтримуваній апаратній чи системній платформі без змін у початковому коді та перекомпіляції.

Цього можна досягти, компілюючи початковий Java код у байт-код, який є спрощеними машинними командами. Потім програму можна виконати на будь-якій платформі, що має встановлену віртуальну машину Java, яка інтерпретує байткод у код, пристосований до специфіки конкретної операційної системи і

процесора. Зараз віртуальні машини Java існують для більшості процесорів і операційних систем.

Стандартні бібліотеки забезпечують загальний спосіб доступу до таких платформозалежних особливостей, як обробка графіки, багатопотоковість та робота з мережами. У деяких версіях задля збільшення продуктивності JVM байт-код можна компілювати у машинний код до або під час виконання програми.

Основна перевага використання байт-коду — це портативність. Тим не менш, додаткові витрати на інтерпретацію означають, що інтерпретовані програми будуть майже завжди працювати повільніше, ніж скомпільовані у машинний код, і саме тому Java одержала репутацію «повільної» мови. Проте, цей розрив суттєво скоротився після введення декількох методів оптимізації у сучасних реалізаціях JVM.

Одним із таких методів є just-in-time компіляція (JIT), що перетворює байт-код Java у машинний під час першого запуску програми, а потім кешує його. У результаті така програма запускається і виконується швидше, ніж простий інтерпретований код, але ціною додаткових витрат на компіляцію під час виконання. Складніші віртуальні машини також використовують динамічну рекомпіляцію, яка полягає в тому, що віртуальна машина аналізує поведінку запущеної програми й вибірково рекомпілює та оптимізує певні її частини. З використанням динамічної рекомпіляції можна досягти більшого рівня оптимізації, ніж за статичної компіляції, оскільки динамічний компілятор може робити оптимізації на базі знань про довкілля періоду виконання та про завантажені класи. До того ж він може виявляти так звані гарячі точки — частини програми, найчастіше внутрішні цикли, які займають найбільше часу при виконанні. JIT-компіляція та динамічна рекомпіляція збільшує швидкість Java-програм, не втрачаючи при цьому портативності.

Існує ще одна технологія оптимізації байт-коду, широко відома як статична компіляція, або компіляція ahead-of-time (AOT). Цей метод передбачає,

як і традиційні компілятори, безпосередню компіляцію у машинний код. Це забезпечує хороші показники в порівнянні з інтерпретацією, але за рахунок втрати переносності: скомпільовану таким способом програму можна запустити тільки на одній, цільовій платформі.

Швидкість офіційної віртуальної машини Java значно покращилася з моменту випуску ранніх версій, до того ж, деякі випробування показали, що продуктивність JIT-компіляторів у порівнянні зі звичайними компіляторами у машинний код майже однакова. Проте ефективність компіляторів не завжди свідчить про швидкість виконання скомпільованого коду, тільки ретельне тестування може виявити справжню ефективність у даній системі [11].

Одна з особливостей концепції віртуальної машини полягає в тому, що помилки (виключення) не призводять до повного краху системи. Крім того, існують інструменти, які «приєднуються» до середовища періоду виконання і кожен раз, коли сталося певне виключення, записують інформацію з пам'яті для зневаження програми. Ці інструменти автоматизованої обробки виключень надають основну інформацію щодо виключень в програмах на Java.

Стандартні бібліотеки забезпечують загальний спосіб доступу до таких платформозалежних особливостей, як обробка графіки, багатопотоковість та робота з мережами.

Таким чином можна виділити такі основні можливості мови програмування Java:

- автоматичне управління пам'яттю;
- розширені можливості обробки виняткових ситуацій;
- багатий набір засобів фільтрації введення-виведення;
- набір стандартних колекцій: масив, список, стек і т. д;
- наявність простих засобів створення мережевих додатків (у тому числі з використанням протоколу RMI);
- наявність класів, що дозволяють виконувати HTTP-запити і обробляти відповіді;

- вбудовані в мову засоби створення багатопоточних додатків, які потім були перенести на багато мов (наприклад Python);

- уніфікований доступ до баз даних : на рівні окремих SQL-запитів - на основі JDBC, SQLJ; на рівні концепції об'єктів, що володіють здатністю до зберігання в базі даних - на основі Java Data Objects (англ.) і Java Persistence API;

- підтримка узагальнень;

- підтримка лямбда, замикань, вбудовані можливості функціонального програмування [12].

2.3 Огляд шаблону MVC

Модель–вигляд–контролер (Model-view-controller, MVC) – архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Мета шаблону – гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

У рамках архітектурного шаблону модель–вигляд–контролер (MVC) програма поділяється на три окремі, але взаємопов'язані частини з розподілом функцій між компонентами. Модель (Model) відповідає за зберігання даних і забезпечення інтерфейсу до них. Вигляд (View) відповідальний за представлення цих даних користувачеві. Контролер (Controller) керує компонентами, отримує

сигнали у вигляді реакції на дії користувача (зміна положення курсора миші, натискання кнопки, ввід даних в текстове поле) і передає дані у модель.

Модель є центральним компонентом шаблону MVC і відображає поведінку застосунку, незалежну від інтерфейсу користувача. Модель стосується прямого керування даними, логікою та правилами застосунку.

Вигляд може являти собою будь-яке представлення інформації, одержуване на виході, наприклад графік чи діаграму. Одночасно можуть співіснувати кілька виглядів (представлень) однієї і тієї ж інформації, наприклад гістограма для керівництва компанії й таблиці для бухгалтерії.

Контролер одержує вхідні дані й перетворює їх на команди для моделі чи вигляду. Модель інкапсулює ядро даних і основний функціонал їхньої обробки і не залежить від процесу вводу чи виводу даних. Вигляд може мати декілька взаємопов'язаних областей, наприклад різні таблиці і поля форм, в яких відображаються дані.

У функції контролера входить відстеження визначених подій, що виникають в результаті дій користувача. Контролер дозволяє структурувати код шляхом групування пов'язаних дій в окремий клас. Наприклад у типовому MVC-проекті може бути користувацький контролер, що містить групу методів, пов'язаних з управлінням обліковим записом користувача, таких як реєстрація, авторизація, редагування профілю та зміна пароля.

Зареєстровані події транслюються в різні запити, що спрямовуються компонентам моделі або об'єктам, відповідальним за відображення даних. Відокремлення моделі від вигляду даних дозволяє незалежно використовувати різні компоненти для відображення інформації. Таким чином, якщо користувач через контролер внесе зміни до моделі даних, то інформація, подана одним або декількома візуальними компонентами, буде автоматично відкоригована відповідно до змін, що відбулися [13].

2.4 Технології проектування бази даних

Для зберігання інформації користувачів, їх приватних переписок необхідно створити БД.

У колишні часи СУБД займалися збереженням таблиць і інших допоміжних об'єктів в файлах, надавали призначений для користувача інтерфейс для перегляду, редагування таблиць, зміни їх структури, створення і видалення таблиць. Крім того, вони пропонували командну керуючу мову, на якій можна було писати програми, а також давати в командному рядку прості керуючі команди. Такі СУБД були розраховані на однокористувальницькі і однозадачні операційні системи на зразок DOS. Як приклади можна назвати схожі між собою СУБД DBase, FoxPro, Clipper, Paradox.

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

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

В сучасних СУБД застосовується клієнт-серверний підхід. В рамках цього підходу СУБД вдає із себе сервер (постачальник послуг), який приймає мережеві з'єднання, які виходять від програм-клієнтів (споживачів послуг). Клієнтські програми можуть працювати як на тому ж комп'ютері, що і програма-сервер, так і на інших комп'ютерах. Як тільки з'єднання встановлено, клієнт може відправляти запити в сторону сервера, і отримувати від нього відповіді. Таким чином СУБД відповідає тільки за обробку запитів і зберігання даних. Всю інтерактивну частину (тобто пов'язану з взаємодією з споживачем) беруть на себе клієнтські програми.

Клієнт-серверна організація СУБД дає значні переваги:

- централізація зберігання даних. Всі об'єкти баз даних зберігаються на одному комп'ютері або на групі комп'ютерів, спеціально призначених для цього.

Ці комп'ютери володіють достатньою швидкодією і обсягом оперативної і постійної пам'яті для виконання покладених на них завдань. Вони можуть перебувати в приміщенні, що охороняється, їх мережевий зв'язок із зовнішнім світом може бути особливо захищений. Ці машини можуть бути підключені до систем безперебійного живлення, так як для багатьох БД дуже важлива надійність в зберіганні даних і безперебійність роботи;

- централізація обробки клієнтських запитів. Оскільки запити від клієнтів можуть надходити в непередбачувані моменти часу і на їх виконання може знадобитися заздалегідь невідомий час, можливі конфлікти запитів. СУБД повинна ставити в чергу на обслуговування такі запити або по можливості обслуговувати їх одночасно. При цьому слід враховувати можливість конфліктних ситуацій, коли, наприклад, кілька запитів намагаються оновити одну і ту ж підмножину записів в таблиці. Або якщо в результаті виконання одного запиту оновлюється підмножина записів, а інший запит вимагає вибірку інформації з цієї ж (частково оновленої) підмножини.

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.

Серед найбільш поширених можна виділити три основні клієнт-серверні СУБД:

- SQLite**;**

- MySQL**;**

- PostgreSQL**.**

Так як SQLite базується на файлах, то вона надає досить широкий набір інструментів для роботи з нею, в порівнянні з мережевими СУБД. При роботі з цією СУБД звернення відбуваються безпосередньо до файлів (в ці файлах зберігаються дані), замість портів і гнізд в мережевих СУБД. Саме тому SQLite дуже бисть, а також потужна завдяки технологіям обслуговуючих бібліотек.

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. 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.

Переваги MySQL:

- простота в роботі - встановити MySQL досить просто. Існують різноманітні програми, наприклад GUI, дозволяє досить легко працювати з БД;

- багатий функціонал - MySQL підтримує більшість функціоналу SQL;

- безпека - велика кількість функцій забезпечують безпеку, які підтримується за замовчуванням;

- масштабованість - MySQL легко працює з великими обсягами даних і легко масштабується;

- швидкість - спрощення деяких стандартів дозволяє MySQL значно збільшити продуктивність.

PostgreSQL є більш професійною з усіх трьох розглянутих нами СУБД. Вона вільно розповсюджується і максимально відповідає стандартам SQL. PostgreSQL або Postgres намагаються повною мірою використовувати ANSI / ISO SQL стандарти своєчасно з виходом нових версій.

Від інших СУБД PostgreSQL відрізняється підтримкою об'єктно-орієнтованого або реляційного підходу до баз даних. Наприклад, повна підтримка надійних транзакцій, тобто атомарність, послідовність, міцність. Завдяки потужним технологіям Postgre дуже продуктивна. Паралельність досягнута не за рахунок блокування операцій читання, а завдяки реалізації управління різноманітним паралелізмом (MVCC), що також забезпечує відповідність ACID. PostgreSQL дуже легко розширювати своїми процедурами, які називаються збережені процедури. Ці функції спрощують використання постійно повторюваних операцій.

Переваги PostgreSQL:

- відкрите ПЗ відповідає стандарту SQL - PostgreSQL - безкоштовне ПЗ з відкритим вихідним кодом. Ця СУБД є дуже потужною системою;

- велике співтовариство - існує досить велика спільнота в якому ви запросто знайдете відповіді на свої питанн;

- велика кількість доповнень - незважаючи на величезну кількість вбудованих функцій, існує дуже багато доповнень, що дозволяють розробляти дані для цієї СУБД і управляти ними.

Недоліки PostgreSQL:

- продуктивність - при простих операціях читання PostgreSQL може значно уповільнити сервер і бути повільніше своїх конкурентів, таких як MySQL;

- популярність - за своєю природою, популярністю ця СУБД похвалитися не може, хоча і є досить велика спільнота;

- хостинг - в силу названих вище чинників іноді досить складно знайти хостинг з підтримкою цієї СУБД.

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

Для управління базами даних необхідно використовувати певну програму адміністрування. Однією з найпоширеніших є HeidiSQL.

Щоб управляти базою даних з HeidiSQL, користувач має увійти на локальний або віддалений сервер MySQL з прийнятним паролем, створивши сесію. В рамках цієї сесії користувач може управляти базами даних MySQL на сервері MySQL, і від'єднатися після закінчення роботи. Можливості програми цілком достатні для більшості операцій із загальними та просунутими базами даних, таблицями та записами, але розробка залишається у активному стані, щоб забезпечити повну функціональність, котра очікується від фронтенду MySQL [14].

Можливості та переваги HeidiSQL:

- можливість підключення до декількох серверів в одному вікні;
- можливість підключення до серверу з командного рядка;
- створення та редагування таблиць, даних, збережених процедур, тригерів;
- можливість експорту бази даних в SQL файл для подальшого імпорту в іншу систему;
- підтримка імпорту текстових файлів;
- пошук по базі даних та багато іншого [15].

2.5 Огляд шаблону DAO

Між БД та системою у цілому доцільно створити певний прошарок, який відповідатиме за передачу запитів в БД і обробку отриманих від неї відповідей. DAO абстрагує сутності системи і робить їх відображення на БД, визначає загальні методи використання з'єднання, його отримання, закриття та (або) повернення в Connection Pool.

Об'єкт доступу до даних (DAO) - об'єкт що надає абстрактний інтерфейс до деяких видів баз даних чи механізмів персистентності реалізуючи певні операції без розкриття деталей бази даних. Він надає відображення від

програмних викликів до рівня персистентності. Така ізоляція розділює запити до даних в термінах предметної області та їх реалізацію засобами СКБД.

Цей паттерн проектування можна застосовувати до більшості мов програмування, видів програмного забезпечення з потребою персистентності та більшості типів баз даних, але він традиційно асоціюється з застосунками Java EE та реляційними БД доступ до яких здійснюють через JDBC API що пов'язано з походженням паттерна із збірки найкращих практик Sun Microsystems для цієї платформи [16].

Вершиною ієрархії DAO є абстрактний клас або інтерфейс з описом загальних методів, які будуть використовуватися при взаємодії з базою даних. Як правило, це методи пошуку, видалення по ключу, оновлення і т. д.

Реалізація DAO на рівні класу має на увазі використання одного єдиного з’єднання з базою для виклику більш ніж одного методу успадкованого DAO класу. Певні можливості надаються незалежно від того, який механізм зберігання використовується і без необхідності спеціальним чином відповідати цьому механізму зберігання. Цей шаблон проектування застосовується до безлічі мов програмування, більшості програмного забезпечення, що потребує зберігання інформації і до більшої частини баз даних. Для з’єднання з базою даних використовується певний драйвер, який надається розробником відповідної СУБД.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка структури та проектування 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 «messenger_bd» до складу якої входять 4 таблиці: «User», «User_from», «User_to», «Msg».

Таблиця «User», структура якої складається з 4 стовпці, є сховищем персональних акаунтів користувачів, вона зберігає таку інформацію як електронну адресу користувача, його ім’я та пароль (у зашифрованому вигляді). Саме з цією таблицею працює функціонал входу та реєстрації.

Таблиця «User_from», структура якої складається з 2 стовпці, зберігає в собі таку інформацію: унікальний номер користувача, який надсилає повідомлення та само його ім’я.

Таблиця «User_to», структура якої містить 5 стовпців, зберігає в собі таку інформацію: унікальний номер користувача який отримує повідомлення, номер користувача який надіслав повідомлення, ім’я того, кому повідомлення

надіслано, текст повідомлення та час.

Для зручного відображення переписки двох користувачів створена додаткова таблиця «Msg». Таблиця містить 5 стовпців: унікальний номер повідомлення, ім’я від кого повідомлення, ім’я кому надіслано повідомлення, текст та час повідомлення. За замовчуванням ця таблиця пуста, та заповнюється автоматично за допомогою запиту:

```
INSERT INTO msg
(from_user, to_user, msg_text, msg_time)
SELECT user_from.name_from, user_to.name_to, user_to.msg_text,
user_to.time_msg
FROM user_from, user_to
WHERE user_from.name_from='name_from' AND user_to.name_to='name_to'
UNION
SELECT user_from.name_from, user_to.name_to, user_to.msg_text,
user_to.time_msg
FROM user_from, user_to
WHERE user_from.name_from=' name_to ' AND user_to.name_to='name_from'
```

Запит спрацьовує у той момент, коли користувач натискає на ім’я іншого користувача, якому бажає надіслати повідомлення. У таблиці генеруються список двох користувачів з сортуванням по часу надіслання повідомлення.

Схему створеної бази даних зображено на рис. 3.1

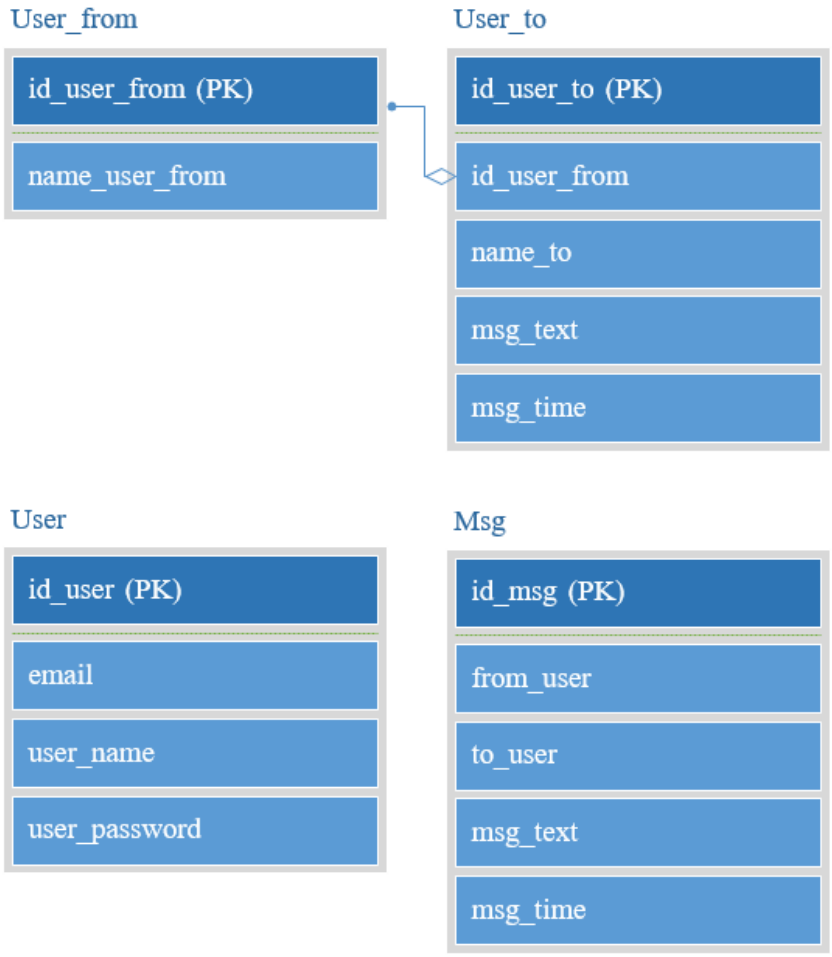

Рисунок 3.1 – Схема бази даних

3.2 Розробка інтерфейсу користувача

Ясність - це перша і найголовніша задача будь-якого інтерфейсу. Щоб інтерфейс ефективно допомагав користувачам домагатися своїх цілей, він повинен мати наступні характеристики.

- він повинен бути дружнім, а його призначення - очевидним для користувача;

- користувачі повинні розуміти, з чим вони взаємодіють через інтерфейс;

- процес взаємодії з інтерфейсом повинен бути передбачуваним.

Інтерфейс було розроблено у лаконічному вигляді. Мінімалістичний та плаский дизайн, зрозумілі та зручні елементи навігації, використання невеликої кількості кольорів і шрифтів все це є концепцію стилю створюваного веб-сервісу.

Після запуску веб-сервісу користувачу необхідно підключитися до серверу, що зображено на рис. 3.2

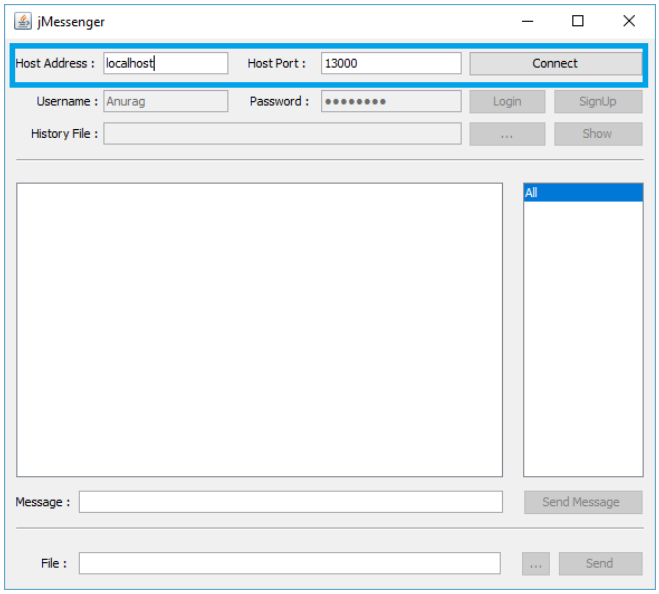

Рисунок 3.2 – Підключення до серверу

Наступним кроком користувачу надається можливість входу або реєстрації, що зображено на рис. 3.3.

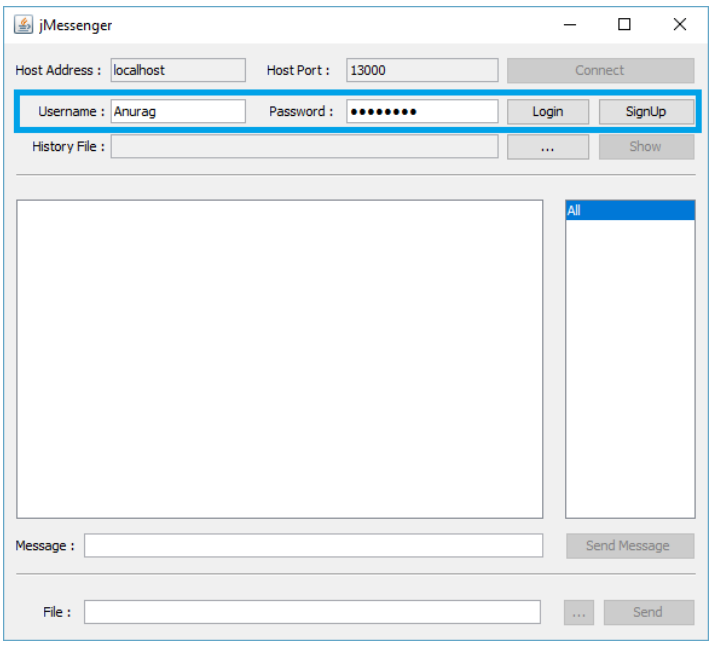

Рисунок 3.3 – Вхід або реєстрація

Після входу користувач отримає повідомлення про успішний вхід до веб-сервісу, що зображено на рис. 3.4.

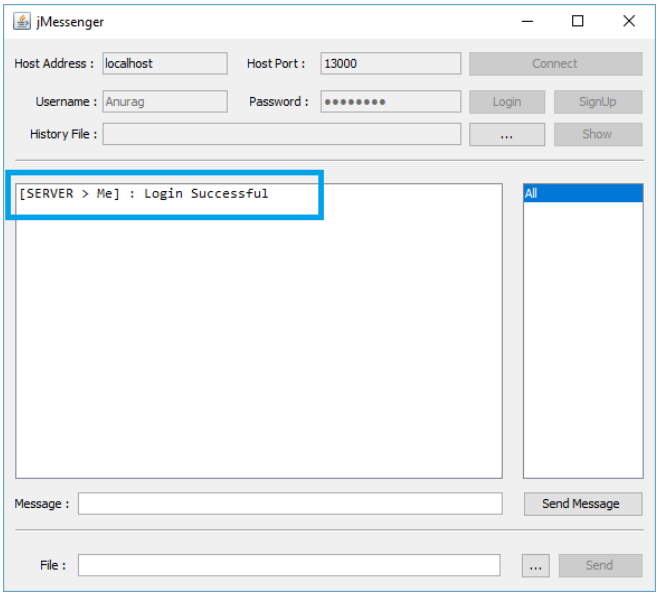

Рисунок 3.4 – Повідомлення про успішний вхід

У разі реєстрації, користувач отримає повідомлення про успішну реєстрацію та про успішний вхід, що зображено на рис. 3.5.

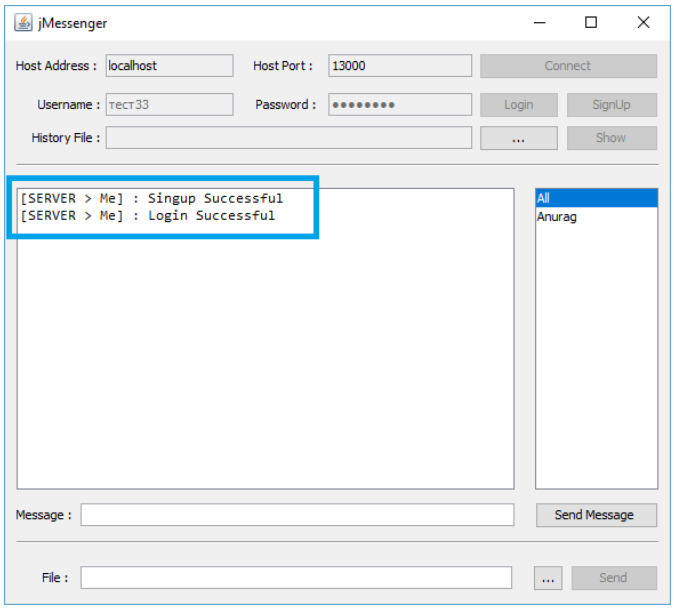

Рисунок 3.5 – Повідомлення про успішну реєстрацію та вхід

Після входу користувач може побачити список користувачів, які знаходяться в даним момент у мережі, що зображено на рис. 3.6

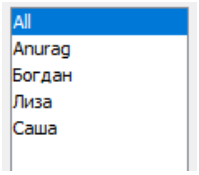

Рисунок 3.6 – Список користувачів у мережі

Користувач може обрати у меню справа людину, котрій бажає написати, та почати вести діалог, що зображено на рис. 3.7.

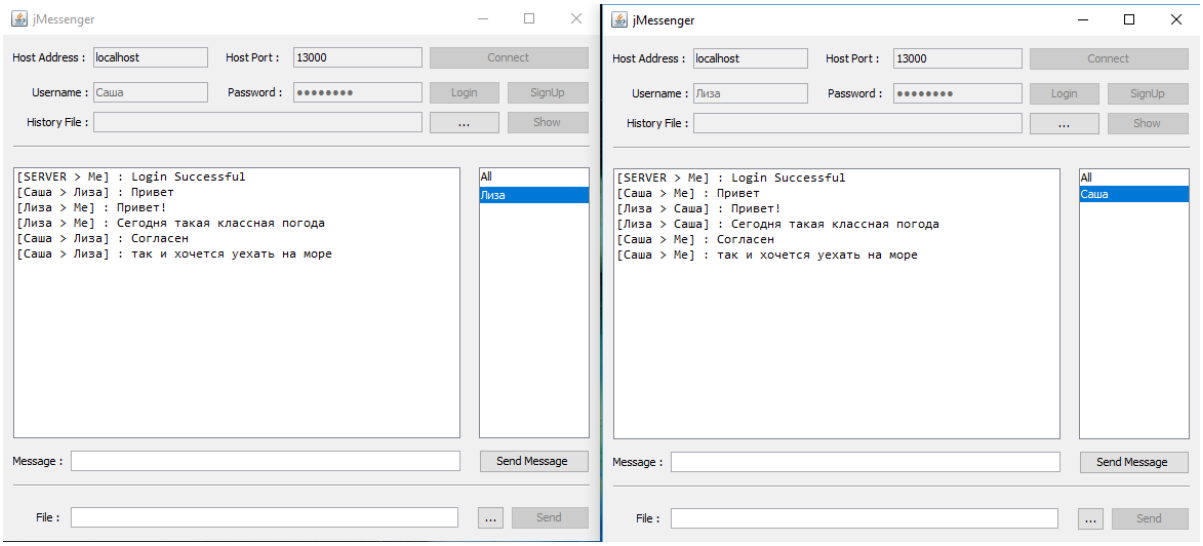

Рисунок 3.7 – Діалог між двома користувачами

Користувачу надається можливість надсилати файли, після цього співрозмовник отримає повідомлення, що зображено на рис. 3.8.

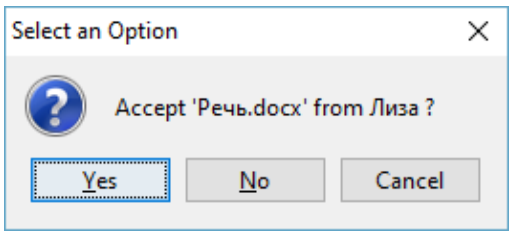

Рисунок 3.8 – Вікно підтвердження прийому файлу

3.3 Розробка алгоритму шифрування

В наш час спостерігається дедалі більше випадків взлому користувацьких облікових записів з ціллю заволодіння особистими даними. Для того щоб захистити користувачів створеного веб-сервісу від небажаного доступу до їх облікових записів було вирішено захистити паролі методом шифрування.

Зважаючи на те, що окрім безпосереднього злому облікових записів, можливий випадок злому серверу з базою даних та вихідним кодом, необхідно розробити такий алгоритм, завдяки якому буде дуже складно підібрати пароль, навіть знаючи алгоритм шифрування.

Для проектування алгоритму було проаналізовано існуючі методи шифрування. Багато з них засновані на хешуванні даних з повторенням в n разів. Такий пароль можливо дізнатись, до того ж існують готові таблиці для підбору хешованих даних. Зі збільшенням числа повторів n збільшується час підбору, але не надійність. Деякі алгоритми додатково використовують ключ, який додається до паролю та хешується разом з ним. Ключ може бути унікальним для кожного користувача або однаковим для всіх. Але такий ключ потрібно зберігати в базі, що означає, що у випадку злому бази даних ключі стануть відомими хакеру.

Ґрунтуючись на описаному вище аналізі, було вирішено об’єднати повторне хешування з додаванням ключа, але збільшити кількість ключів до 3, додати в програмний код декілька непотрібних ключів, для того щоб заплутати зломщика, розділяти пароль на декілька частин та на певних ітераціях вставляти між частинами ключі та додавати до хеш-коду довжину проміжних результатів. Детальний алгоритм зображено на рис. 3.9.

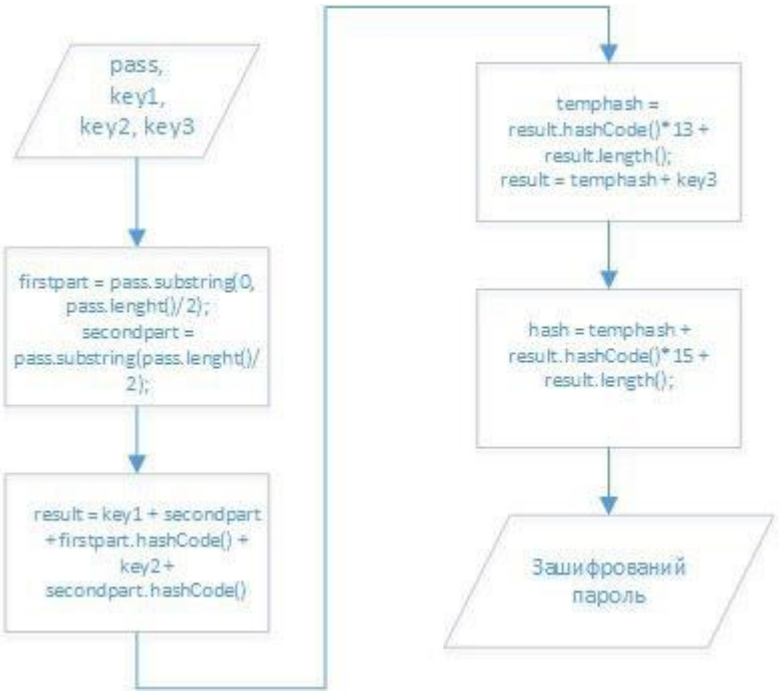

Рисунок 3.9 – Алгоритм шифрування

Неможливо забезпечити абсолютну надійність системи, але розроблений таким чином алгоритм збільшує кількість часу, необхідного на підбор паролю, в десятки разів, оскільки, навіть знаючи алгоритм, дуже складно передбачити всі складові проміжних результатів та віднайти їх. До того ж для отримання вихідного паролю з захешованих за таким алгоритмом даних необхідно володіти дуже потужним обладнанням. Для заволодіння користувацькими обліковими записами веб-сервісу інтернет листування ці затрати не доцільні, що значно зменшує вірогідність хакерської атаки.

3.4 Проектування серверної частини

При проектуванні програмного продукту необхідно було створити окремо додаток, який буде виступати у ролі серверу, та клієнтську частину.

Програмний продукт побудовано на базі багатопотоковості мови програмування Java. Існує головний потік, який виступатиме у ролі серверу, будь які повідомлення та події оброблятимуться саме головним потоком. Користувачі виступають у ролі дочірнього потоку. Два користувача не можуть безпосередньо

спілкуватися один з одним, їм необхідно мати посередника, ним виступає саме головний потік. Кожне повідомлення має певний тип, такий як login, message, signup, upload_req. Коли до серверу (головного потоку) приходить повідомлення, перевіряється тип, та виконуються дії, реалізовані для даного типу.

Для серверної частини було розроблено такі класи контролери для керування формами:

- Database – реалізує зв’язок с базою даних, реалізує функції створення нового юзера, перевірку правильності введення логіну;

- Message – клас який реалізує інтерфейс Serializable, для передачі повідомлень серверу;

- ServerFrame – клас який керує формую, яка надає можливість обрати базу даних, зі збереженими даними користувачів, реалізує функції кожної кнопки та ініціює усі компоненти форми;

- ServerThread – клас, який керує головним потоком програми;

- SocketServer – клас, який обробляє повідомлення і події, які надсилаються до сервера від кожного клієнта (обробка реєстрації, входу, надісланого повідомлення).

Детальний перелік всіх параметрів і функцій класів та їх залежність один від одного серверної частини програмного продукту зображено на рис 3.10.

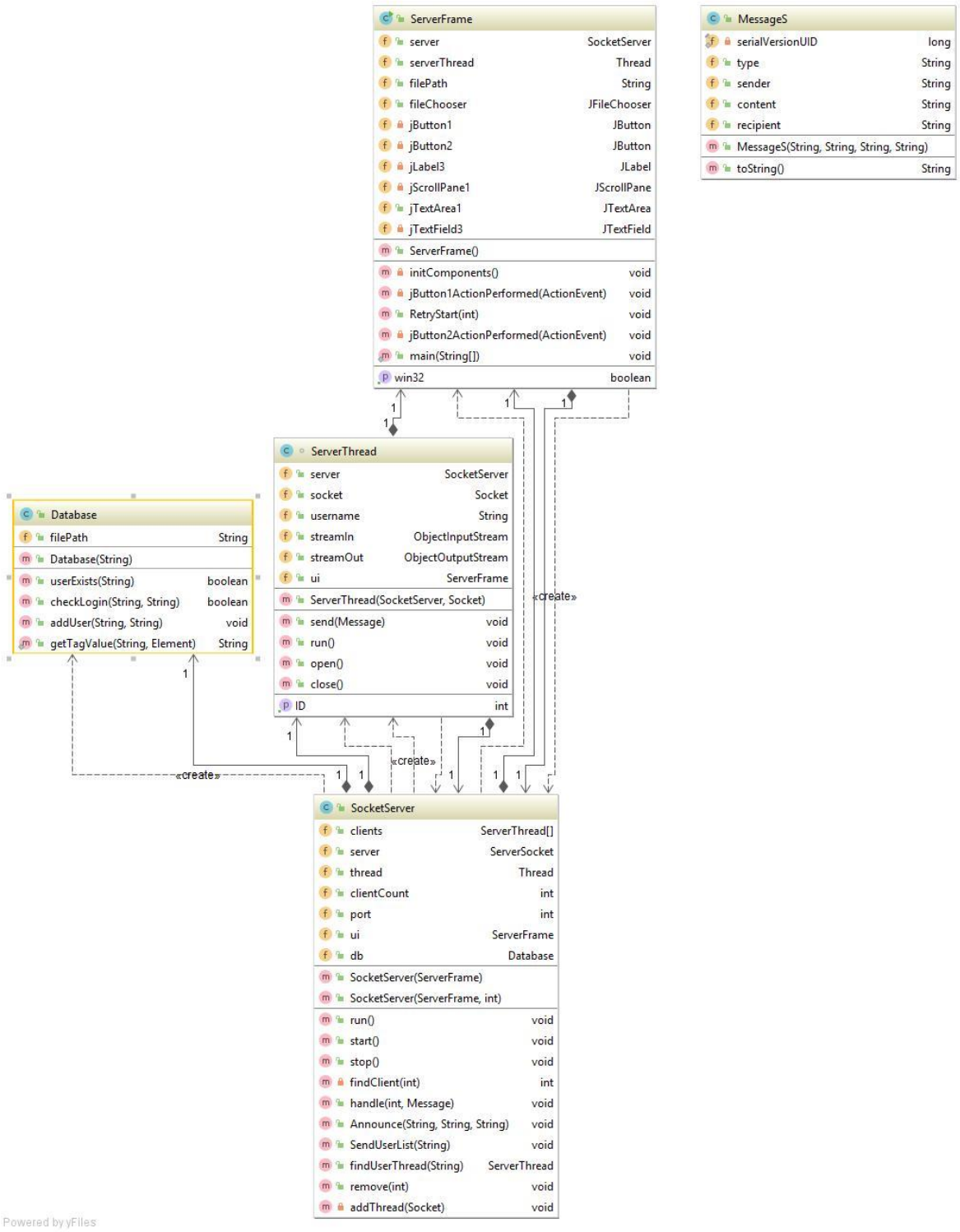

Рисунок 3.10 – Діаграма класів серверної частини

Для клієнтської частини було розроблено такі класи контролери:

- Download – клас, який керує завантаженням файлів;

- History – клас, який керує зберіганням історії;

- Message – аналог класу Message з серверної частини;

- SocketClient - клас який керує формую, яка надає можливість підключитися до сервера, реалізує функції кожної кнопки та ініціює усі компоненти форми, створює дочірні потоки, надсилає повідомлення певного типу до сервера;

- Upload – клас, який керує вивантаженням файлів;

- ChatFrame – керування формою чату;

- HistoryFrame – керування формою історії повідомлень.

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

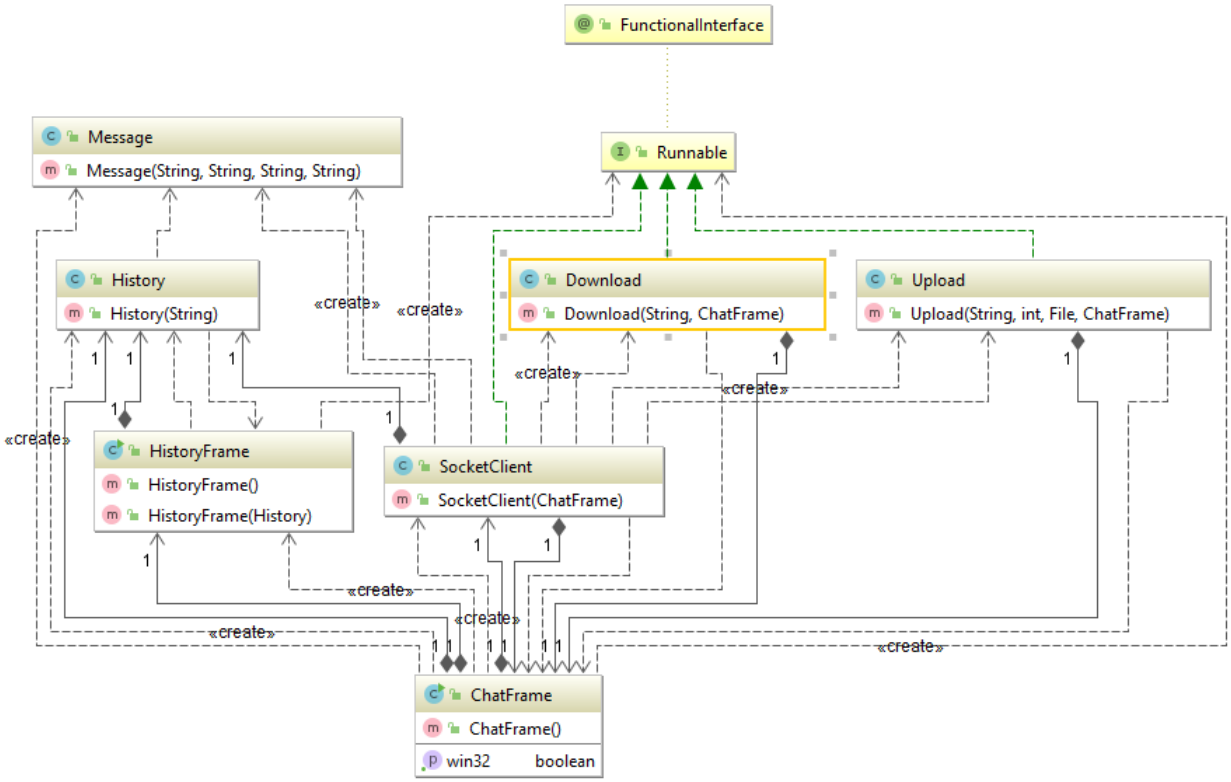

Рисунок 3.11 – Діаграма класів клієнтської частини

4 ТЕХНІКО-ЕКОНОМІЧНА ЧАСТИНА

4.1 Резюме

Розроблене програмне забезпечення для інтернет листування призначено для експлуатації різними типами користувачів.

Даний веб-сервіс дозволить надсилати повідомленнями у мережі Інтернет своїм товаришам та співробітникам на підприємстві. 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 i3-8300i 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 45000 грн.

4.2 Опис програмного продукту

Найменування товару – дослідження та створення методів шифрування повідомлень користувача веб-сервісу для інтернет листування Призначення – надсилання та отримання повідомлень у мережі Інтернет. Область використання – орієнтований на використання як звичайними користувачами так і великими підприємствами. Характеристики програмного продукту (ПП) приведені в табл. 4.1.

Таблиця 4.1 - Характеристики ПП

Найменування	Значення
Операційна система	Windows 10
Оперативна пам'ять	4096 Мбайт и вище
CPU	Intel core i5-9600K
Середовище програмування	IntelliJ IDEA 2018.3.5
Мова програмування	Java

4.3 Дослідження й аналіз ринку збуту

— Сегментація ринку по споживачах:

Таблиця 4.2 - Сегментація ранка по основних споживачах

Галузь використання	Код споживача	Споживач			
		I	II	III	IV
Підприємства	А	+	+	+	+
Користувачі	Б	-	-	-	+

I - системні адміністратори;
II - адміністратори груп;
III - спостерігачі ресурсів;
IV - ліцензовані користувачі.

Як видно з табл. 4.2, даним програмним продуктом має можливість користуватися як звичайний користувач мережі Інтернет, так і співробітники будь-якого підприємства, у своїх комерційних цілях. Уточнимо ємність сегментів ринку (дані в табл. 4.3).

Таблиця 4.3 - Аналіз ємності сегментів ринку

Галузі використання	Кількість об'єктів	Передбачуване число продажів одному об'єктові	Передбачувана ємність сегмента
Підприємства	10	1	10
Користувачі	70	1	70
Разом місткість ринку	80		80

Як видно з табл. 4.3, найбільше число передбачуваних продажів одному об'єктові приходиться на звичайних користувачів. Це обумовлено специфікою програмного продукту.

Основними вимогами споживачів є безкоштовний обмін інформаційними повідомленнями зі своїми друзями, простий інтуїтивно-зрозумілий інтерфейс,

швидкодія, захищеність повідомлень користувачів подвійним наскрізним шифруванням.

Продаж розробленого продукту охоплює всю Україну, але не виключений продаж і за її межами. Максимальна кількість потенційних споживачів у розглянутому регіоні - 25000.

Прогноз обсягів продажів програмного комплексу приведений у табл. 4.4.

Таблиця 4.4 - Прогноз обсягів продажів ПП

Періоди	Кількість
Для першого року реалізації	
Січень	2
Лютий	1
Березень	5
Квітень	2
Травень	1
Червень	3
Липень	2
Август	1
Вересень	5
Жовтень	3
Листопад	2
Грудень	1
Усього	28
Для другого року реалізації	
I квартал	5
II квартал	6
III квартал	4
IV квартал	1
Усього	16
Для третього року реалізації	
Усього	14

Параметрична сегментація ринку:

Для проведення багатофакторної сегментації продукту оцінимо його характеристики, що відповідають обраним нами параметрам (по п'ятибальній шкалі). Багатофакторна сегментація приведена в табл. 4.5.

Таблиця 4.5 - Параметрична сегментація ринку

Фактори, що характеризують Товар	Категорія споживачів		Підсумкова оцінка	Відсоток до загального підсумку
	А	Б		
Ціна	3	5	8	18.6
Вимоги до ПЕОМ	3	5	8	18.6
Простота використання	4	5	9	20.9
Надійність роботи	5	4	9	20.9
Захищеність даних	5	4	9	20.9
Разом	20	24	43	100

Виходячи з даних табл. 4.5, можна зробити висновок про те, що такі фактори як простота використання захищеність даних і надійність роботи є найбільш важливими, а сегмент ринку Б пред'являє найбільше високі вимоги до сукупності якісних параметрів розроблювального виробу.

4.4 Оцінка конкурентоздатності

За наявними в розроблювачів відомостями розроблене програмне забезпечення має декілька аналогів, таких як Viber, Hangouts, WhatsApp.

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.

Вихідні дані для розрахунку приведені в табл. 4.6.

Таблиця 4.6 - Розрахунок узагальненого показника якості

Параметри	Одиниці виміру	Вагомість B_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 Мін	$B_i \times M_{iн}$	Відносний одиничний показник $M_{iг}$	$B_i \times M_{iг}$
1. Ціна ПП	грн.	0,2	45000	54216	0,83	0,166	1,0	0,2
2. Мінімально необхідна кількість ОЗУ	Мб.	0,2	512	700	0,73	0,14	1,0	0,2
3. Зручність використання		0,2	10	8	1,25	0,25	1,0	0,2
4. Надійність		0,4	10	8	1,25	0,5	1,0	0,4
Усього		1				1,056		1

Величина відносного показника якості обчислюється по формулі:

$$M_{i'} = \frac{P_{iд}}{P_i} \text{ чи} \quad (4.1)$$

$$M_{iН} = \frac{P_i}{P_{iГ}}, \quad (4.2)$$

при цьому $M_{iн} > 1.0$.

4.5 Стратегія маркетингу

Поширення товару буде вироблятися шляхом прямих продажів.

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

$$З_{ЗП} = \overline{З_м} \times T, \quad (4.3)$$

де Т - час розробки ПП;

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

$$\overline{З_м} = \frac{\sum_{i=1}^{n} З_{м\,i}}{n}, \quad (4.4)$$

де $З_{мі}$ - заробітна плата і-го розроблювача;

n - кількість розроблювачів.

Таблиця 4.7 - Основна заробітна плата розроблювачів

Посада	Заробітна плата, грн.	Кількість розроблювачів
Інженер 1-й категорії	8500	1

Використовуючи дані, приведені в табл. 4.7, і виходячи з того, що розробка ПП велася 4 місяців, одержимо:

$$З_{зп}=8500\times4=34000 \text{ грн.}$$

Експлуатаційні витрати:

$$Ер=Т_{мв}\times С_{мч}, \quad (4.5)$$

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

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

$$Т_{мв}=Т\times Ф\times Т_{ч}, \quad (4.6)$$

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

$Т_{ч}$ - кількість годин, прооблених на ЕОМ у день (5 ч).

$$Т_{мв}=4\times22\times5=440 \text{ ч.}$$

$$Ер=440\times2=880 \text{ грн.}$$

Потреби в матеріальних ресурсах і устаткуванні для виробництва програмного продукту приведені в табл. 4.8 і табл. 4.9.

Таблиця 4.8 - Витрати на обладнання

Обладнання	Призначення	Кількість	Вартість у грн.
Intel Core i3-8300i	Для написання програми і її налагодження, а також для підготовки документів	1	15000
Принтер Epson XP 117		1	2500
Усього Зоб			17500

Вартість основних виробничих фондів визначається по формулі:

$$C_{О\,П\,Ф} = \frac{З_{о\,б} \times Т}{12}, \tag{4.7}$$

де Зоб - витрати на обладнання (дані в табл. 4.8).

$$C_{О\,П\,Ф} = \frac{17500 \times 4}{12} = 5833 \text{ грн.}$$

Таблиця 4.9 - 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. 4.10.

Прибуток П обчислюється як 30% від витрат на розробку ПП (дані в табл. 4.10)

$$П=0.3\times З_р, \quad (4.8)$$

$$П=0,3\times 54461.59= 16338,47 \text{ грн.}$$

Таблиця 4.10 - Розрахунок витрат на розробку програмного продукту

№ п/п	Найменування статей витрат	Значення в грн.
1	Вартість основних фондів	5833
2	Вартість матеріалів	280
3	Основна заробітна плата розроблювачів	34000
4	Додаткова заробітна плата розроблювачів (10% від п3)	3400
5	Єдині соціальні відрахування (22% від п3+п4)	8228
6	Експлуатаційні витрати	880
7	Накладні витрати (до 70% від п1)	1341,59
8	Комунальний податок (10% від мін. зар. плати)	500
Разом $З_р$		54461,59

Максимальна ціна розроблювального ПП буде

$$Ц_{max}=1.2\times(З_р+1.3\times П), \quad (4.9)$$

$$Ц_{max}=1.2\times(54461.59+1.3\times 16338.47)= 90841.93 \text{ грн.}$$

Отримана ціна є максимальною. Однак ця ціна може бути зменшена і складатися з витрат на тиражування ($З_{тир}$) і адаптацію (Зад) 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 $Ц_{min}$ визначаємо по формулі:

$$Ц_{min}=1.2\times(З_{тир}+Зад+1.3\times П'), \quad (4.10)$$

де Зад - витрати на адаптацію (приймаємо 5% від $З_р$);

П' - прибуток з одного продажу, грн:

$$П'=0.3\times(З_{тир}+Зад), \quad (4.11)$$

$З_{тир}$ - витрати на тиражування ПП:

$$З_{тир}=С_{мч}\times Т_к+З_д+З_и, \quad (4.12)$$

де $Т_к$ - час копіювання системи, година (приймаємо 0.005 години);

$З_д$ - вартість диска, грн (приймаємо 5 грн);

$З_и$ - зарплата виконавця, грн/година:

$$З_и = \frac{8500}{22\times 5} = 77 \text{ грн/година},$$

$$З_{тир}=2\times 0.005+5+77=82.01 \text{ грн},$$

$$Зад=0.05\times 54461.59=2723.07 \text{ грн},$$

$$П'=0,3\times(82,01 +2723.07)= 841,52 \text{ грн}.$$

Мінімальна ціна буде:

$$Ц_{min}=1.2\times(82.01 +2723.07 +1.3\times841.52)= 4678.86 \text{ грн.}$$

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

$$4678.86 \text{ грн} \leq Ц_{прод} \leq 90841.93 \text{ грн.}$$

Приймаємо продажну ціну програмного продукту без ПДВ рівної 45000 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 6 соціальних групах.

Таким чином, ціна реклами буде складати:

$$6\times10000=60000 \text{ грн.}$$

5 ОХОРОНА ПРАЦІ ТА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

5.1 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.

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.

Закон України «Про охорону праці» визначає основні положення щодо реалізації конституційного права працівників на охорону їх життя і здоров’я у процесі трудової діяльності, на належні, безпечні і здорові умови праці, регулює за участю відповідних органів державної влади відносини між роботодавцем і працівником з питань безпеки, гігієни праці та виробничого середовища і встановлює єдиний порядок організації охорони праці в Україні [17].

Приміщення лабораторії, у якому виконувалася дана робота, розташовано на п’ятому поверсі семиповерхового будинку. Площа приміщення, у якому виконувався дипломний проект, складає 33 м^2, у ньому 3 робочі місця, тобто на робоче місце приходиться 11 м^2. Об’єм приміщення лабораторії складає 99 м^3, тобто на одне робоче місце приходиться 33 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 20м^3 можна зробити висновок, що розміри приміщення відповідають нормам проектування.

5.2 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. 5.1.

Таблиця 5.1 – 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 і її технічною периферією	Шкідливий
Електричний струм	Мережа перемінного струму	Небезпечний
ЕМВ, у тому числі рентгенівське	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, статичність і незручність пози	Шкідливий
Пожежна небезпека приміщенням	Наявність спаленних матеріалів і можливих джерел запалювання	Небезпечний, шкідливий

5.3 Виробнича санітарія

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

підтримуватися тільки оптимальні параметри мікроклімату, що наведені в табл. 5.2.

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 5.2 – Оптимальні параметри мікроклімату в приміщенні з ЕОМ

Період року	Категорія робіт з важкості	Температура, t, °С	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.

5.3.1 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. 5.3.

Таблиця 5.3 – 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, $D_н$, 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 500 лк. Штучне освітлення реалізується шляхом встановлення визначеної кількості ламп білого світла – ЛБ 80.

5.3.2 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 корпуса.

5.3.3 Електромагнітне випромінюваня

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.

5.4 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.

5.4.1 Індивідуальне завдання

Розрахунок захисного заземлення

Дані для виконання індивідуального завдання зображено табл. 5.4 та табл. 5.5

Таблиця 5.4 – Індивідуальне завдання, частина 1

№	Трансформатор на підстанція напругою *U*, кВт	Розміри будинку		Розрахунковий опір природного заземлювача, Re, Ом	Довжина лінії електропередач	
		Довжина *L*, м	Ширина *B*, м		$l_{к.л.}$, км	$L_{в.л.}$, км
1	10/ 6	24	12	18	45	70

Таблиця 5.5 – Індивідуальне завдання, частина 2

Параметри вертикального електрода		Параметри горизонтального електрода	Питомий опір землі ρ обмірюване, Ом·м	Кліматична зона
Довжина lВ, м	Діаметр d, мм2	Переріз полоси, мм2		
3	14	4 х 40	110	II

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

$$Iз = \frac{Uл}{350}(35lк + lв) \qquad (5.1)$$

$$I_з = \frac{(10/6)}{350}(35*45+70) = 47(A);$$

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

lв, lк- довжина електрично пов'язаних відповідно кабельних і повітряних ліній, км;

2. Визначення необхідного опору штучного заземлювача:

$$R_з = \frac{250}{I} \tag{5.2}$$

$$R_з = \frac{250}{47} = 5.31(Ом)$$

$$Ru = \frac{ReeR}{Re - Rз} \tag{5.3}$$

$$Ru = \frac{15 * 5.31}{15 - 5.31} = 8.31(Ом)$$

де, *Re* - опір розтікання струму природних заземлювачів, Ом;

Ru - необхідний опір штучного заземлювача, Ом;

Rз - розрахункове нормоване опір ЗУ, Ом;

$$Rз = Ru$$

3. Визначення розрахункового питомого опору землі за формулою:

$$\rho = \rho_{изм} \cdot \Psi' \tag{5.4}$$

$$p = 110 * 1.5 = 165 = (Ом * м)$$

4. Обчислення опору розтікання струму одиночного вертикального заземлювача Rв, Ом:

$$R_в = \frac{\rho_в}{2\pi l}\left(\ln\frac{2l}{d} + \frac{1}{2}\ln\frac{4t+l}{4t-l}\right) \quad (5.5)$$

$$R_в = \frac{165}{2\pi * 3}\left(\ln\frac{2*3}{0.014} + \frac{1}{2}\ln\frac{4*2.3+3}{4*2.3-3}\right) = 55.96(Ом)$$

де, $\rho_в$ – 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.

5. Розрахунок наближеної (мінімальної) кількості вертикальних стрижнів:

$$n' = \frac{Rв}{Ru} \quad (5.6)$$

$$n' = \frac{55.96}{8.31} = 6.73 = 7$$

$Rв$ – опір розтікання струму одиночного вертикального заземлювача, Ом;

Ru – необхідний опір штучного заземлювача, Ом

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

по контуру $l_Г = 1{,}05an$ =44.1 м

ряд $l_Г = 1{,}05a(n-1)$ = 37.8 м

$$a = k \cdot l_в \quad (5.7)$$

a=3*2=6;

де, k – коефіцієнт кратності, що дорівнює 2;

$l_в$ – довжина вертикального стрижня.

n – кількість вертикальних стрижнів.

7. Обчислення опору розтікання струму горизонтального стрижня Rr:

$$R_г = \frac{\rho}{2\pi l} \ln \frac{2l^2}{bt} \tag{5.8}$$

$$R_г = \frac{165}{2\pi * 3} \ln \frac{2 * 3^2}{12 * 2.3} = 3.67\,(Ом)$$

Де,

ρ – розрахунковий питомий опір ґрунту, Ом·м;

l – довжина горизонтальної смуги, м;

b – ширина полоси, м;

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

8. Вибір коефіцієнтів використання вертикальних стрижнів і горизонтальної смуги з урахуванням числа вертикальних стрижнів і відносини відстані між стрижнями до їх довжині:

$(\eta_в) = 0.85$

$(\eta_г) = 0.88$

9. Розрахувати еквівалентний опір розтікання струму групового заземлювача:

$$R_{гр} = \frac{R_в R_Г}{R_в \eta_Г + R_Г \eta_в \cdot n} \quad (5.9)$$

$$R_{гр} = \frac{55.96 * 3.67}{55.96 * 0.85 + 3.67 * 0.88 + 7} = 3.55 (Ом)$$

Результати розрахунків наведені в табл. 5.6

Таблиця 5.6 – результати розрахунків

$\rho_{гр}$ Ом·м	$l_в$, м	n, шт	$l_г$, *м*	$\eta_в$	$\eta_г$	$R_в$, *Ом*	$R_г$, *Ом*	$R_{гр}$, *Ом*	$R_и$, *Ом*
165	10.5	7	44.1	0.85	0.88	55.96	3.67	3.55	8.31

5.5 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 встановлено III рівень захисту від блискавок відповідно до 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 малої електропровідності. Застосування пінних вогнегасників виключено, тому що ЕОМ може перебувати під напругою. Робоче 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;
- організаційними заходами щодо пожежної безпеки.

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;
- видання необхідних інструкцій і плакатів;
- наявність плану евакуації.

5.6 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; э 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 мг/м3, рекомендовано 0,075 мг/м3; ГДК озону під час роботи лазерного принтеру − 0,02 мг/м3. Особливо жорсткі вимоги до повторно використовуваних матеріалів.

Апарати, тара і документація повинні допускати нетоксичну вторинну переробку після закінчення терміну експлуатації. В ЕПТ міститься багато біоактивних речовин, що треба ураховувати під час утилізації.

Міжнародні стандарти, починаючи з ТСО-92, включають вимоги зниженого енергоспоживання та обмеження припустимих рівнів потужності, що споживаються у неактивних режимах.

6 ЦИВІЛЬНИЙ ЗАХИСТ

6.1 Вступ

Цивільний захист – є державною системою органів управління, сил і засобів, що створюється для організації і забезпечення захисту населення від наслідків надзвичайних ситуацій техногенного, екологічного, природного та воєнного характеру.

У результаті виникнення й розвитку будь-якої надзвичайної ситуації можуть з’явитися постраждалі або людські жертви.

У даному розділі дипломної роботи розглядається питання: «Основи проведення рятівних та інших невідкладних робіт у надзвичайних ситуаціях». Поставлену задачу будемо розглядати з огляду на виникнення надзвичайної ситуації в офісі.

Внаслідок знаходження великої кількості людей в одному приміщенні та з обмеженими варіантами його покидання, актуальними є питання проведення рятівних та інших невідкладних робіт у надзвичайних ситуаціях.

6.2 Рятувальні роботи

Сутність рятувальних та інших невідкладних робіт – це усунення безпосередньої загрози життю та здоров’ю людей, відновлення життєзабезпечення населення, запобігання або значне зменшення матеріальних збитків. Рятувальні та інші невідкладні роботи включають також усунення пошкоджень, які заважають проведенню рятувальних робіт, створення умов для наступного проведення відновлювальних робіт. РІНР поділяють на рятувальні роботи і невідкладні роботи [29].

До рятувальних робіт відносяться:

- розвідка маршруту руху сил, визначення обсягу та ступеня руйнувань, розмірів зон зараження, швидкості і напрямку розповсюдження зараженої хмари чи пожежі;

- локалізація та гасіння пожеж на маршруті руху сил та ділянках робіт;

- визначення об'єктів і населених пунктів, яким безпосередньо загрожує небезпека;

- визначення потрібного угрупування сил і засобів запобігання і локалізації небезпеки;

- пошук уражених та звільнення їх з-під завалів, пошкоджених та палаючих будинків, із загазованих та задимлених приміщень;

- розкриття завалених захисних споруд та рятування з них людей;

- надання потерпілим першої допомоги та евакуація їх (при необхідності) у лікувальні заклади;

- вивіз або вивід населення із небезпечних місць у безпечні райони;

- організація комендантської служби, охорона матеріальних цінностей і громадського порядку;

- відновлення життєздатності населених пунктів і об'єктів; – пошук, розпізнавання і поховання загиблих;.

- санітарна обробка уражених;

- знезараження одягу, взуття, засобів індивідуального захисту, територій, споруд, а також техніки;

- соціально-психологічна реабілітація населення.

До невідкладних робіт відносяться:

- прокладання колонних шляхів та улаштування проїздів (проходів) у за валах та на зараженій території;.

- локалізація аварій на водопровідних, енергетичних, газових і технологічних мережах;

- ремонт та тимчасове відновлення роботи комунально-енергетичних систем і мереж зв’язку для забезпечення рятувальних робіт;

- зміцнення або руйнування конструкцій, які загрожують обвалом і безпечному веденню робіт;

Аварійно-рятувальні та інші невідкладні роботи здійснюються в три етапи [30]:

На першому етапі вирішуються завдання:

- щодо екстреного захисту населення;

- з запобігання розвитку чи зменшення впливу наслідків;

- з підготовки до виконання АРІНР.

Основними заходами щодо екстреного захисту населення е:

- оповіщення про небезпеку;

- використання засобів захисту;

- додержання режимів поведінки;

- евакуація з небезпечних у безпечні райони;

- локалізація аварій;

- зупинка чи зміна технологічного процесу виробництва;

- попередження «запобігання» і гасіння пожеж.

На другому етапі проводяться:

- пошук потерпілих;

- витягання потерпілих з-під завалів, з палаючих будинків, пошкоджених транспортних засобів;

- евакуація людей із зони лиха, аварії, осередку ураження;

- надання медичної допомоги;

- санітарна обробка людей;

- знезараження одягу, майна, техніки, території;

- проведення інших невідкладних робіт, що сприяють і забезпечують здійснення рятувальних робіт.

На третьому етапі вирішуються завдання щодо забезпечення життєдіяльності населення у районах, які потерпіти від наслідків НС:

- відновлення чи будівництво житла;

- відновлення енерго -,тепло водо -, газопостачання, ліній зв'язку;

- організація медичного обслуговування;

- забезпечення продовольством і предметами першої необхідності;

- знезараження харчів, води, фуражу, техніки, майна, території;

- соціально-психологічна реабілітація;

- відшкодування збитків.

Відновлювальні роботи ЦЗ не виконує, їх здійснюють спеціально створені підрозділи. Залежно від рівня НС для проведення АРІНР залучаються сили і засоби ЦЗ центрального, регіонального, або об'єктового підпорядкування [30].

6.3 Рятування людей при надзвичайних ситуаціях

Рятування людей при надзвичайних ситуаціях є найважливішим видом аварійно-рятувальних та інших невідкладних робіт і являє собою сукупність заходів щодо переміщення людей із зони впливу небезпечних факторів надзвичайної ситуації та їхніх вторинних проявів або захисту людей від впливу цих факторів, у тому числі з використанням засобів індивідуального захисту та захисних споруд (укриттів).

Порядок і способи рятування людей визначаються керівником робіт з ліквідації надзвичайної ситуації залежно від обстановки у зоні надзвичайної ситуації і стану людей. При проведенні аварійно-рятувальних та інших невідкладних робіт враховуються стан основних та запасних шляхів евакуації, технічна оснащеність зони надзвичайної ситуації системами оповіщення, аварійного освітлення, а також характерні риси небезпечних факторів надзвичайної ситуації [30].

Основними способами рятування людей і майна є:

- переміщення їх у безпечне місце, у тому числі з використанням спеціальних технічних засобів;

- захист від впливу небезпечних факторів надзвичайної ситуації.

Для рятування людей вибираються найбільш безпечні шляхи і способи. Переміщення постраждалих у безпечне місце здійснюється з урахуванням умов ліквідації надзвичайної ситуації та їх стану.

Захист людей від впливу небезпечних факторів надзвичайної ситуації у випадку неможливості їхнього переміщення у безпечне місце здійснюється з використанням засобів індивідуального захисту органів дихання та зору, а також за допомогою використання спеціальних речовин і матеріалів, що перешкоджають поширенню та знижують вплив небезпечних факторів надзвичайної ситуації.

Для рятування людей застосовуються такі засоби:

- аварійно-рятувальне устаткування та пристрої;
- рятувальні пристрої (рятувальні рукави, мотузки, трапи та індивідуальні рятувальні пристрої);
- апарати захисту органів дихання та зору;
- літальні апарати;
- плавальні засоби;
- стаціонарні та ручні пожежні драбини тощо;
- автодрабини та автопідіймачі;
- інші доступні засоби рятування.

У ході аварійно-рятувальних та інших невідкладних робіт потерпілим надається екстрена медична допомога.

Надання екстреної допомоги постраждалим здійснюється відповідно до протоколів, затверджених центральним органом виконавчої влади у сфері охорони здоров’я, що регламентують дії сил, призначених для проведення аварійно-рятувальних та інших невідкладних робіт. Із цією метою можуть застосовуватися засоби індивідуального захисту органів дихання і зору, засоби екстреної медичної допомоги, а також інші засоби.

До прибуття у зону надзвичайної ситуації медичного персоналу екстрену медичну допомогу постраждалим у встановленому порядку надає особовий склад підрозділів, що проводять аварійно-рятувальні та інші невідкладні роботи.

6.4 Аварійно-рятувальні роботи внаслідок вибуху

Дії підрозділу на пожежо- і вибухонебезпечному об’єкті включають, у першу чергу, проведення розвідки як на об’єкті, так і на прилеглій до нього території. При організації розвідки особлива увага звертається на наявність постраждалих при вибухах на об’єкті та у найближчих житлових будинках, ступінь руйнування будинків, споруд, місця виникнення завалів, наявність та справність зовнішнього протипожежного водопостачання, стаціонарних систем пожежогасіння тощо.

У ході проведення розвідки встановлюються:

- райони пожеж і їх характер, визначаються основні напрямки вводу сил та засобів для проведення рятувальних робіт та гасіння пожеж, напрямок і швидкість поширення вогню, зони загазованості і наявність загрози населенню;

- межі району локалізації та гасіння пожеж;

- місцезнаходження потерпілих;

- наявність ділянок сильного задимлення, характер руйнування резервуарів (сховищ) і трубопроводів;

На основі даних розвідки проводиться оцінка обстановки та визначаються: заходи з організації рятування людей, порядку надання допомоги постраждалим та залучення для цього необхідних засобів; основні тактичні прийоми з ліквідації надзвичайної ситуації; рубежі локалізації і гасіння пожеж; напрями і шляхи відходу особового складу у разі загрози вибуху або викиду нафтопродуктів; організація зовнішнього протипожежного водопостачання; засоби захисту особового складу від небезпечних факторів [31].

Найважливішим завданням є пошук і деблокування постраждалих із зруйнованих будівель. Роботи за технологічним принципом розділяються на три основні види:

- деблокування постраждалих, які знаходяться під уламками будівельних конструкцій;

- деблокування постраждалих із замкнутих приміщень;

- рятування людей з верхніх поверхів зруйнованих будівель.

Виконання робіт з деблокування постраждалих здійснюється такими способами:

- послідовне розбирання завалів;

- влаштування лазів;

- вироблення галереї в ґрунті під завалом;

- пробивання отворів у стінах та перекриттях.

Під час виконання робіт, пов’язаних з ліквідацією аварії внаслідок вибуху, проводяться заходи для захисту особового складу і техніки від ураження вибуховою хвилею, осколками і уламками конструкцій, що розлітаються, теплового впливу та ураження органів дихання продуктами горіння [31].

Одночасно здійснюються заходи щодо рятування людей з палаючих, зруйнованих будинків і зон задимлення, надання їм медичної допомоги і евакуації в лікарні та спеціалізовані лікувальні заклади охорони здоров’я.

ВИСНОВКИ

У дипломній роботі вирішена науково-технічна задача – дослідження та створення методів шифрування повідомлень користувача веб-сервісу для інтернет листування.

Детально розглянуті існуючі методи шифрування повідомлень, порівняні існуючі методи з метою виявлення недоліків і переваг кожного з них;

Проведено аналіз існуючих методів хешування паролів. Досліджені способи злому хешей. Проведено аналіз технологій та методів розробки. На основі аналізу був зроблений вибір на користь мови програмування Java як основної мови розробки, технології Swing як засіб створення графічного інтерфейсу і СУБД MySQL як засобу зберігання інформації.

У проектному розділі були виконані розробку бази даних, з описом усіх таблиць і їх призначення, розроблено інтерфейс, створений власний алгоритм шифрування, спроектовано серверну і клієнтську частину веб-сервісу.

Створений сервіс та компоненти системи відповідають заявленим функціональним і сучасним технічним вимогам і готові до широкого використання.

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

Розглянуті питання охорони праці та навколишнього середовища. Також було визначено оптимальні параметри мікро клімату і характеристику виробничого освітлення в приміщенні, дозволені рівні шуму та вібрації, статичної електрики та електромагнітних випромінювань. Розглянуто питання електробезпеки та пожежної безпеки.

Розглянуті питанні цивільного захисту населення. Описано сутність рятувальних та інших невідкладних робіт, рятування людей при надзвичайних ситуаціях, аварійно-рятувальних робіт внаслідок вибухів

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

1 Криптографія // Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Криптографія#Історія_криптографії, 30.11.2020.

2 Семенов Ю.А. Алгоритм DES // Режим доступу до ресурса: book.itep/6/des_641.htm, 20.11.2020.

3 Онацький А. В., Йона Л. Г. Асиметричні методи шифрування. – Модуль 2 Криптографічні методи захисту інформації в телекомунікаційних системах та мережах: навч. посібник, 2011

4 ProtonMail. Наскрізне шифрування: опис і принцип роботи методу // Режим доступу до ресурсу: https://www.comss.ru/page.php?id=2468, 20.11.2020

5 Timing Analysis of Keystr ok es and Timing Attacks on SSH. : // Режим доступу: https://people.eecs.berkeley.edu/~daw/papers/ssh-use01.pdf., 10.10.2020

6 NIST Comments on Cryptanalytic Attacks on SHA-1 - NIST Information Technology Laboratory / 2006.

7 CERT Vulnerability Notes Database. Software Engineering Institute. Original Release Date: 2008. – 21с.;

8 Введение в Java. Язык программирования Java. Доступ до матеріалу: https://metanit.com/java/tutorial/1.1.php.

9 Design Goals of the JavaTM Programming Language.

10 Кей С. Хорстманн (2014). Java SE 8. Вводный курс. «Вільямс».

11 History of Java in Java Application Servers. TechMetrix Research, 1999.

12 JavaFX: Getting Started with JavaFX Oracle. Доступ до матеріалу: https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfxverview. html.

13 Krasner, G.E. and S.T. Pope, A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80, Journal of OOP, 1(3), pp. 26-49, August/September 1988, SIGS Publications, New York, NY, USA, 1988.

14 The mysql Open Source Project on Open Hub: Languages Page — 2006.

15 HeidiSQL. Доступ до матеріалу: https://www.heidisql.com/forum.php?t=26392

16 Core J2EE Patterns - Data Access Objects. Sun Microsystems Inc.

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-XII у редакції Редакція від 18.12.2019, підстава - 139-IX

29 Цивільна оборона: навч. посіб. / О.П. Депутат, І.В. Коваленко, І.С. Мужик.; за ред. В. С. Франчука. – Львів: Афіша, 2000. – 336 с.

30 Кодекс цивільного захисту України – ВРУ №5403-VI, від 2.10.2012

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

ДОДАТОК А

Лістинг програмного коду

A.1 ServerFrame.java

```
package com.socket;

import java.awt.Color;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.io.File;
import javax.swing.JFileChooser;
import javax.swing.UIManager;

public class ServerFrame extends javax.swing.JFrame {

    public SocketServer server;
    public Thread serverThread;
    public String filePath = "D:/Data.xml";
    public JFileChooser fileChooser;

    public ServerFrame() {
        initComponents();
        jTextField3.setEditable(false);
        jTextField3.setBackground(Color.WHITE);

        fileChooser = new JFileChooser();
        jTextArea1.setEditable(false);
    }

    public boolean isWin32(){
        return System.getProperty("os.name").startsWith("Windows");
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents() {

        jButton1 = new javax.swing.JButton();
        jScrollPane1 = new javax.swing.JScrollPane();
        jTextArea1 = new javax.swing.JTextArea();
        jLabel3 = new javax.swing.JLabel();
        jTextField3 = new javax.swing.JTextField();
        jButton2 = new javax.swing.JButton();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("jServer");

        jButton1.setText("Start Server");
        jButton1.setEnabled(false);
        jButton1.addActionListener(new java.awt.event.ActionListener() {
```

```
                public void actionPerformed(java.awt.event.ActionEvent evt) {
                    jButton1ActionPerformed(evt);
                }
            });

            jTextArea1.setColumns(20);
            jTextArea1.setFont(new java.awt.Font("Consolas", 0, 12)); // NOI18N
            jTextArea1.setRows(5);
            jScrollPane1.setViewportView(jTextArea1);

            jLabel3.setText("Database File : ");

            jButton2.setText("Browse...");
            jButton2.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent evt) {
                    jButton2ActionPerformed(evt);
                }
            });

            javax.swing.GroupLayout           layout           =           new
javax.swing.GroupLayout(getContentPane());
            getContentPane().setLayout(layout);
            layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jScrollPane1)
                        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                            .addComponent(jLabel3)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                            .addComponent(jTextField3,
javax.swing.GroupLayout.DEFAULT_SIZE, 282, Short.MAX_VALUE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                            .addComponent(jButton2,
javax.swing.GroupLayout.PREFERRED_SIZE,                                    91,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                            .addComponent(jButton1)))
                    .addContainerGap())
            );
            layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(jTextField3,
javax.swing.GroupLayout.PREFERRED_SIZE,    javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(jLabel3)
```

```
                        .addComponent(jButton2)
                        .addComponent(jButton1))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                    .addComponent(jScrollPane1,
javax.swing.GroupLayout.DEFAULT_SIZE, 287, Short.MAX_VALUE)
                    .addContainerGap())
            );

            pack();
        }// </editor-fold>//GEN-END:initComponents

        private  void  jButton1ActionPerformed(java.awt.event.ActionEvent  evt)
{//GEN-FIRST:event_jButton1ActionPerformed
            server = new SocketServer(this);
            jButton1.setEnabled(false); jButton2.setEnabled(false);
        }//GEN-LAST:event_jButton1ActionPerformed

        public void RetryStart(int port){
            if(server != null){ server.stop(); }
            server = new SocketServer(this, port);
        }

        private  void  jButton2ActionPerformed(java.awt.event.ActionEvent  evt)
{//GEN-FIRST:event_jButton2ActionPerformed
            fileChooser.showDialog(this, "Select");
            File file = fileChooser.getSelectedFile();

            if(file != null){
                filePath = file.getPath();
                if(this.isWin32()){ filePath = filePath.replace("\\", "/"); }
                jTextField3.setText(filePath);
                jButton1.setEnabled(true);
            }
        }//GEN-LAST:event_jButton2ActionPerformed

        public static void main(String args[]) {

            try{

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
            }
            catch(Exception ex){
                System.out.println("Look & Feel Exception");
            }

            java.awt.EventQueue.invokeLater(new Runnable() {
                public void run() {
                    new ServerFrame().setVisible(true);
                }
            });
        }
        // Variables declaration - do not modify//GEN-BEGIN:variables
        private javax.swing.JButton jButton1;
        private javax.swing.JButton jButton2;
        private javax.swing.JLabel jLabel3;
        private javax.swing.JScrollPane jScrollPane1;
        public javax.swing.JTextArea jTextArea1;
```

```
        private javax.swing.JTextField jTextField3;
        // End of variables declaration//GEN-END:variables
    }
```

A.2 Database.java

```
    package com.socket;

import java.io.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.*;

public class Database {

    protected static char[] upper =
            {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
'N', 'O', 'P', 'Q',
                    'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'};

    protected static char[] lower = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
            'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u',
'v', 'w', 'x', 'y', 'z'};

    public static final int n = 26;
    public static final int k = 1;

    public static int findIndUpper(char c) {
        int rez = -1;

        for (int i = 0; i < upper.length; ++i) {
            if (c == upper[i]) rez = i;
        }
        return rez;
    }

    public static int findIndLower(char c) {
        int rez = -1;

        for (int i = 0; i < lower.length; ++i) {
            if (c == lower[i]) rez = i;
        }
        return rez;
    }


    public String filePath;

    public Database(String filePath) {
        this.filePath = filePath;
    }
```

```
    String input = "";
    StringBuilder output = new StringBuilder();

    {
        for (int i = 0; i < input.length(); ++i) {
            char c = input.charAt(i);

            if (findIndLower(c) == -1) {
                int y = (findIndUpper(c) + k) % n;
                output.append(upper[y]);
            }
            if (findIndUpper(c) == -1) {
                int y = (findIndLower(c) + k) % n;
                output.append(lower[y]);
            }
        }
    }

    public boolean userExists(String username) {

        try {
            File fXmlFile = new File(filePath);
            DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(fXmlFile);
            doc.getDocumentElement().normalize();

            NodeList nList = doc.getElementsByTagName("user");

            for (int temp = 0; temp < nList.getLength(); temp++) {
                Node nNode = nList.item(temp);
                if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element eElement = (Element) nNode;
                    if (getTagValue("username", eElement).equals(username)) {
                        return true;
                    }
                }
            }
            return false;
        } catch (Exception ex) {
            System.out.println("Database exception : userExists()");
            return false;
        }
    }

    public boolean checkLogin(String username, String password) {

        if (!userExists(username)) {
            return false;
        }

        try {
            File fXmlFile = new File(filePath);
            DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(fXmlFile);
```

```
            doc.getDocumentElement().normalize();

            NodeList nList = doc.getElementsByTagName("user");

            for (int temp = 0; temp < nList.getLength(); temp++) {
                Node nNode = nList.item(temp);
                if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element eElement = (Element) nNode;
                    if (getTagValue("username", eElement).equals(username) &&
getTagValue("password", eElement).equals(password)) {
                        return true;
                    }
                }
            }
            System.out.println("Hippie");
            return false;
        } catch (Exception ex) {
            System.out.println("Database exception : userExists()");
            return false;
        }
    }

    public void addUser(String username, String password) {

        try {
            DocumentBuilderFactory docFactory =
DocumentBuilderFactory.newInstance();
            DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
            Document doc = docBuilder.parse(filePath);

            Node data = doc.getFirstChild();

            Element newuser = doc.createElement("user");
            Element newusername = doc.createElement("username");
            newusername.setTextContent(username);
            Element newpassword = doc.createElement("password");
            newpassword.setTextContent(password);

            newuser.appendChild(newusername);
            newuser.appendChild(newpassword);
            data.appendChild(newuser);

            TransformerFactory transformerFactory =
TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.newTransformer();
            DOMSource source = new DOMSource(doc);
            StreamResult result = new StreamResult(new File(filePath));
            transformer.transform(source, result);

        } catch (Exception ex) {
            System.out.println("Exceptionmodify xml");
        }
    }

    public static String getTagValue(String sTag, Element eElement) {
        NodeList nlList =
eElement.getElementsByTagName(sTag).item(0).getChildNodes();
        Node nValue = (Node) nlList.item(0);
```

```
        return nValue.getNodeValue();
    }
}
```

A.3 Message.java

```
package com.socket;

import java.io.Serializable;

public class Message implements Serializable{

    private static final long serialVersionUID = 1L;
    public String type, sender, content, recipient;

    public Message(String type, String sender, String content, String
recipient){
        this.type = type; this.sender = sender; this.content = content;
this.recipient = recipient;
    }

    @Override
    public String toString(){
        return "{type='"+type+"', sender='"+sender+"',
content='"+content+"', recipient='"+recipient+"'}";
    }
}
```

A.4 ServerThread.java

```
package com.socket;

import java.io.*;
import java.net.*;

class ServerThread extends Thread {

public SocketServer server = null;
public Socket socket = null;
public int ID = -1;
public String username = "";
public ObjectInputStream streamIn  =  null;
public ObjectOutputStream streamOut = null;
public ServerFrame ui;

public ServerThread(SocketServer _server, Socket _socket){
  super();
    server = _server;
    socket = _socket;
    ID     = socket.getPort();
    ui = _server.ui;
}

public void send(Message msg){
    try {
        streamOut.writeObject(msg);
```

```
            streamOut.flush();
        }
        catch (IOException ex) {
            System.out.println("Exception [SocketClient : send(...)]");
        }
    }

    public int getID(){
          return ID;
    }

    @SuppressWarnings("deprecation")
      public void run(){
      ui.jTextArea1.append("\nServer Thread " + ID + " running.");
        while (true){
          try{
                Message msg = (Message) streamIn.readObject();
            server.handle(ID, msg);
            }
            catch(Exception ioe){
                 System.out.println(ID + " ERROR reading: " +
ioe.getMessage());
                server.remove(ID);
                stop();
            }
        }
    }

    public void open() throws IOException {
        streamOut = new ObjectOutputStream(socket.getOutputStream());
        streamOut.flush();
        streamIn = new ObjectInputStream(socket.getInputStream());
    }

    public void close() throws IOException {
      if (socket != null)    socket.close();
        if (streamIn != null)  streamIn.close();
        if (streamOut != null) streamOut.close();
    }
}
```

```
public class SocketServer implements Runnable {

    public ServerThread clients[];
    public ServerSocket server = null;
    public Thread       thread = null;
    public int clientCount = 0, port = 13000;
    public ServerFrame ui;
    public Database db;

    public SocketServer(ServerFrame frame){

        clients = new ServerThread[50];
        ui = frame;
```

```
        db = new Database(ui.filePath);

      try{
          server = new ServerSocket(port);
            port = server.getLocalPort();
          ui.jTextArea1.append("Server startet. IP : " +
InetAddress.getLocalHost() + ", Port : " + server.getLocalPort());
          start();
        }
      catch(IOException ioe){
            ui.jTextArea1.append("Can not bind to port : " + port +
"\nRetrying");
            ui.RetryStart(0);
      }
    }

    public SocketServer(ServerFrame frame, int Port){

        clients = new ServerThread[50];
        ui = frame;
        port = Port;
        db = new Database(ui.filePath);

      try{
          server = new ServerSocket(port);
            port = server.getLocalPort();
          ui.jTextArea1.append("Server startet. IP : " +
InetAddress.getLocalHost() + ", Port : " + server.getLocalPort());
          start();
        }
      catch(IOException ioe){
            ui.jTextArea1.append("\nCan not bind to port " + port + ": " +
ioe.getMessage());
      }
    }

    public void run(){
      while (thread != null){
            try{
            ui.jTextArea1.append("\nWaiting for a client ...");
              addThread(server.accept());
          }
          catch(Exception ioe){
                ui.jTextArea1.append("\nServer accept error: \n");
                ui.RetryStart(0);
          }
        }
    }

    public void start(){
      if (thread == null){
            thread = new Thread(this);
          thread.start();
      }
    }

    @SuppressWarnings("deprecation")
    public void stop(){
```

```
        if (thread != null){
            thread.stop();
          thread = null;
      }
    }

    private int findClient(int ID){
      for (int i = 0; i < clientCount; i++){
            if (clients[i].getID() == ID){
                    return i;
                }
      }
      return -1;
    }

    public synchronized void handle(int ID, Message msg){
      if (msg.content.equals(".bye")){
            Announce("signout", "SERVER", msg.sender);
            remove(ID);
      }
      else{
            if(msg.type.equals("login")){
                if(findUserThread(msg.sender) == null){
                    if(db.checkLogin(msg.sender, msg.content)){
                        clients[findClient(ID)].username = msg.sender;
                        clients[findClient(ID)].send(new Message("login",
"SERVER", "TRUE", msg.sender));
                        Announce("newuser", "SERVER", msg.sender);
                        SendUserList(msg.sender);
                    }
                    else{
                        clients[findClient(ID)].send(new Message("login",
"SERVER", "FALSE", msg.sender));
                    }
                }
                else{
                    clients[findClient(ID)].send(new Message("login", "SERVER",
"FALSE", msg.sender));
                }
            }
            else if(msg.type.equals("message")){
                if(msg.recipient.equals("All")){
                    Announce("message", msg.sender, msg.content);
                }
                else{
                    findUserThread(msg.recipient).send(new Message(msg.type,
msg.sender, msg.content, msg.recipient));
                    clients[findClient(ID)].send(new Message(msg.type,
msg.sender, msg.content, msg.recipient));
                }
            }
            else if(msg.type.equals("test")){
                clients[findClient(ID)].send(new Message("test", "SERVER", "OK",
msg.sender));
            }
            else if(msg.type.equals("signup")){
                if(findUserThread(msg.sender) == null){
                    if(!db.userExists(msg.sender)){
```

```
                        db.addUser(msg.sender, msg.content);
                        clients[findClient(ID)].username = msg.sender;
                        clients[findClient(ID)].send(new Message("signup",
"SERVER", "TRUE", msg.sender));
                        clients[findClient(ID)].send(new Message("login",
"SERVER", "TRUE", msg.sender));
                        Announce("newuser", "SERVER", msg.sender);
                        SendUserList(msg.sender);
                    }
                    else{
                        clients[findClient(ID)].send(new Message("signup",
"SERVER", "FALSE", msg.sender));
                    }
                }
                else{
                    clients[findClient(ID)].send(new Message("signup", "SERVER",
"FALSE", msg.sender));
                }
            }
            else if(msg.type.equals("upload_req")){
                if(msg.recipient.equals("All")){
                    clients[findClient(ID)].send(new Message("message",
"SERVER", "Uploading to 'All' forbidden", msg.sender));
                }
                else{
                    findUserThread(msg.recipient).send(new Message("upload_req",
msg.sender, msg.content, msg.recipient));
                }
            }
            else if(msg.type.equals("upload_res")){
                if(!msg.content.equals("NO")){
                    String IP =
findUserThread(msg.sender).socket.getInetAddress().getHostAddress();
                    findUserThread(msg.recipient).send(new Message("upload_res",
IP, msg.content, msg.recipient));
                }
                else{
                    findUserThread(msg.recipient).send(new Message("upload_res",
msg.sender, msg.content, msg.recipient));
                }
            }
      }
    }

    public void Announce(String type, String sender, String content){
        Message msg = new Message(type, sender, content, "All");
        for(int i = 0; i < clientCount; i++){
            clients[i].send(msg);
        }
    }

    public void SendUserList(String toWhom){
        for(int i = 0; i < clientCount; i++){
            findUserThread(toWhom).send(new Message("newuser", "SERVER",
clients[i].username, toWhom));
        }
    }
```

```
    public ServerThread findUserThread(String usr){
        for(int i = 0; i < clientCount; i++){
            if(clients[i].username.equals(usr)){
                return clients[i];
            }
        }
        return null;
    }

    @SuppressWarnings("deprecation")
    public synchronized void remove(int ID){
    int pos = findClient(ID);
        if (pos >= 0){
            ServerThread toTerminate = clients[pos];
            ui.jTextArea1.append("\nRemoving client thread " + ID + " at " +
pos);
          if (pos < clientCount-1){
                for (int i = pos+1; i < clientCount; i++){
                    clients[i-1] = clients[i];
               }
          }
          clientCount--;
          try{
                  toTerminate.close();
          }
          catch(IOException ioe){
                  ui.jTextArea1.append("\nError closing thread: " + ioe);
          }
          toTerminate.stop();
      }
    }

    private void addThread(Socket socket){
      if (clientCount < clients.length){
            ui.jTextArea1.append("\nClient accepted: " + socket);
          clients[clientCount] = new ServerThread(this, socket);
          try{
                  clients[clientCount].open();
              clients[clientCount].start();
              clientCount++;
          }
          catch(IOException ioe){
                  ui.jTextArea1.append("\nError opening thread: " + ioe);
          }
      }
      else{
            ui.jTextArea1.append("\nClient refused: maximum " + clients.length +
" reached.");
      }
    }
}
```