

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ
кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

“До захисту допущено”
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА

“ ” _____ 2024 р.

Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою
“Цифрові технології в енергетиці”
зі спеціальності 122 **“Комп’ютерні науки”**

на тему:

Система захисту даних на базі алгоритму шифрування Ель-Гамалія

Виконав (-ла):

студент (-ка) IV курсу, групи ТР-02

Сухляк Дмитро Володимирович

(прізвище, ім’я, по батькові)

(підпис)

Керівник:

доцент, к.т.н., Крячок Олександр Степанович

(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові)

(підпис)

Рецензент:

(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім’я, по батькові)

(підпис)

Нормоконтролер:

(підпис)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів без відповідних
посилань.

Студент (-ка) _____
(підпис)

Київ - 2024

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти - перший (бакалаврський)

спеціальність 122 - “Комп’ютерні науки”

Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ

Завідувач кафедри ЦТЕ

Наталія АУШЕВА

«__» _____ 2024 р.

**ЗАВДАННЯ
на дипломну роботу студенту**

Сухляку Дмитру Володимировичу

(прізвище, ім’я, по батькові)

1. Тема роботи Система захисту даних на базі алгоритму
шифрування Ель-Гамала

керівник роботи **Крячок Олександр Степанович, к.т.н., доцент**

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «__» _____ 202__ р. № _____

2. Термін подання роботи студентом 07.06.2024 року

3. Вихідні дані до роботи мова програмування Java, фреймворк JavaFX, СКБД
MySQL, сервер Tomcat, середовище розробки IntelliJ
Idea

4. Перелік питань, які потрібно розробити _____

1) провести аналіз вимог до системи захисту даних, проаналізувати можливості
існуючих аналогів,

2) обрати стек технологій та алгоритмів для реалізації програмного забезпечення,
аргументувати вибір,

- 3) побудувати архітектури програмного забезпечення та баз даних,
 4) створити прикладний програмний веб-інтерфейс,
 5) провести тестування та порівняння створеної системи захисту даних з аналогом,
 6) представити процес застосування веб-інтерфейсу.

5. Орієнтований перелік ілюстративного матеріалу діаграми, що демонструють порівняння розробленої системи з аналогічними існуючими, архітектуру системи, бази даних; зображення, що демонструють взаємодію користувачів із системою, особливості програмного забезпечення та його встановлення; приклади роботи з програмним продуктом.

6. Дата видачі завдання «15» вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі	17-20.04.24	
3.	Розробка архітектури та загальної структури системи	21-25.04.24	
4.	Розробка частин окремих підсистем	25.04-02.05.24	
5.	Програмна реалізація системи	03-07.05.24	
6.	Оформлення пояснювальної записки	08-12.05.24	
7.	Захист програмного продукту	13-17.05.24	
8.	Передзахист	03-06.06.24	
9.	Подання готової роботи на кафедру	07.06.2024	
10.	Захист	17-21.06.2024	

Студент

(підпис)

Дмитро СУХЛЯК

(ім'я, ПРИЗВИЩЕ)

Керівник роботи

(підпис)

Олександр КРЯЧОК

(ім'я, ПРИЗВИЩЕ)

АНОТАЦІЯ

Метою роботи було створення системи захисту даних на основі алгоритму шифрування Ель-Гамала. Система повинна шифрувати та розшифровувати текст, який користувач відправляє на сервер або отримує. Для забезпечення захисту даних ключі повинні генеруватись індивідуально для кожного користувача та зберігатись в базі даних без можливості до неї доступу сторонніх осіб.

В серверній частині ПЗ відбувається взаємодія з базою даних, розшифрування тексту користувача та шифрування відповіді для нього.

Клієнтська частина надає доступ користувачеві до взаємодії з сервером за допомогою графічного інтерфейсу, відображаючи отримані від нього дані. Користувач може переглядати, оновлювати, додавати та редагувати дані, що будуть зашифровані перед надсиланням на сервер.

Дипломна робота виконана на 62 сторінках, містить 21 ілюстрацію, 5 формул, 9 таблиць, 1 додаток, 26 джерел в переліку посилань.

ABSTRACT

The aim of this work was to create a data protection system based on the ElGamal encryption algorithm. The system should encrypt and decrypt text that the user sends to or receives from the server. To ensure data protection, keys should be generated individually for each user and stored in a database without the possibility of access by unauthorized persons.

In the server-side software, interaction with the database, decryption of user text, and encryption of responses for the user takes place.

The client-side software provides the user with access to interact with the server through a graphical interface, displaying the data received from it. The user can view, update, add, and edit data, which will be encrypted before being sent to the server.

The thesis consists of 62 pages, includes 21 illustrations, 5 formulas, 9 tables, 1 appendix, and 26 references in the list of sources.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ.....	8
ВСТУП.....	9
1 АНАЛІЗ ЗАДАЧІ ЗАХИСТУ ДАНИХ.....	11
1.1 Мета програмного забезпечення.....	11
1.2 Вимоги до програмного забезпечення	12
1.3 Шифрування даних	12
1.3.1 Симетричне шифрування	13
1.3.2 Хеш-функції.....	14
1.3.3 Асиметричне шифрування	14
1.3.4 Гібридне шифрування.....	14
1.4 Інші системи захисту даних	15
1.5 Математична модель алгоритму шифрування Ель-Гамала	16
1.6 Порівняння алгоритмів Ель-Гамала та RSA.....	17
2 ЗАСОБИ РОЗРОБКИ	20
2.1 Вибір мови програмування та середовища розробки.....	20
2.2 Вибір системи керування базою даних	22
2.3 Інструменти розробки клієнтського застосунку	23
2.4 Інструменти розробки серверної частини ПЗ.....	24
2.5 Допоміжні інструменти розробки.....	25
3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ ЗАХИСТУ ДАНИХ.....	27
3.1 Архітектура ПЗ	27
3.1.1 Клієнтська сторона.....	29
3.1.2 Серверна сторона	31
3.3. Візуалізація функціональних вимог	32
3.4 Архітектура бази даних	33
3.5 Процес автентифікації користувача в системі	38

	7
3.6 Процес збірки програмного забезпечення	39
4 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ	40
4.1 Інсталяція та системні вимоги	40
4.2 Автентифікація користувача в системі	42
4.3 Головні функції застосунку	43
4.4 Візуалізація шифрування	47
ВИСНОВОК	49
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	50
ДОДАТОК А	52

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

СКБД - система керування базами даних.

ПЗ - програмне забезпечення.

API (англ. Application Programming Interface) - прикладний програмний інтерфейс.

БД - база даних.

CPU (англ. Central processing unit) - центральний процесор.

ОС - операційна система.

MVC (англ. Model-view-controller) — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

VPN (англ. virtual private network) — віртуальна приватна мережа.

Desktop-застосунок - це програмне забезпечення, яке встановлюється і виконується на комп'ютері.

Фреймворк (англ. framework) — це структура або набір інструментів і бібліотек, який надає основу для розробки ПЗ.

JDK (англ. Java Development Kit) — це комплект інструментів для розробки програмного забезпечення на мові програмування Java.

RAM (англ. Random Access Memory) — оперативна пам'ять комп'ютера.

Деплоймент (англ. deployment) — процес розгортання програмного забезпечення робоче середовище.

GUI (англ. Graphical User Interface) — це графічний інтерфейс користувача, який дозволяє взаємодіяти з комп'ютерною системою через графічні елементи.

ВСТУП

У сучасному світі, де обсяг інформації зростає з кожним днем, питання захисту даних набувають дедалі більшої актуальності. Інформаційна безпека є критично важливою як для приватних осіб, так і для організацій, які прагнуть зберегти конфіденційність, цілісність та доступність своїх даних. Одним із ключових методів забезпечення захисту інформації є криптографія - наука про шифрування даних.

Сучасна криптографія являє собою широкий спектр методів та технологій для забезпечення безпеки інформації. Вона базується на складних математичних алгоритмах і комп'ютерних технологіях, котрі використовуються для забезпечення безпеки та цілісності даних в багатьох важливих сферах.

Захист даних є важливим у багатьох сферах, проте деякі з них потребують особливої уваги через специфіку інформації, яку вони обробляють, та потенційні наслідки її компрометації. Захист даних має критичне значення в фінансовій, медичній сферах, державному секторі, Інтернет-сервісах тощо. Це важливо не лише з етичних та фінансових причин, а й законодавство більшості країн зобов'язує заклади зберігати дані клієнтів в безпеці та з обмеженням до них доступу сторонніх осіб. Тому для забезпечення цих потреб необхідно впроваджувати системи захисту даних.

Для реалізації рішення даної проблеми було запропоновано розробити desktop-застосунок для керування даними з використанням алгоритму шифрування для гарантії безпеки інформації.

Для шифрування даних пропонується використовувати алгоритм Ель-Гамала, який гарантуватиме обмежений доступ до інформації, котра зберігається в базі даних, та забезпечуватиме безпечне надсилання до бази даних без загрози компрометації, оскільки алгоритм використовує ймовірнісні ключі, що значно підвищує стійкість до різноманітних атак на шифр.

Мовою програмування пропонується обрати Java, оскільки вона має велику кількість бібліотек та фреймворків, що значно спрощує розробку програмного забезпечення. В якості системи керування базою даних запропоновано обрати

MySQL, котра надає доступ до сучасних інструментів побудови системи зберігання та доступу до даних.

Впровадження системи захисту даних дозволить клієнтам бути впевненими в захищеності своїх даних, а застосування алгоритму Ель-Гамалю гарантує працівникам надійне шифрування даних та оптимальну швидкість доступу до вже існуючої інформації, що не буде сповільнювати їхню роботу.

Записка складається з чотирьох розділів:

У першому розділі проведено аналіз проблеми захисту даних, описано існуючі рішення та обраний алгоритм. Описано порівняння вибраного алгоритму з головним аналогом.

У другому розділі описано перелік інструментів, що використовувались в ході розробки програмного забезпечення. Вибір кожного інструменту аргументовано.

У третьому розділі описано програмну реалізацію системи захисту даних. Описано архітектуру ПЗ та бази даних, написано про допоміжні інструменти та нюанси в підготовці ПЗ до використання.

У четвертому розділі описано процес роботи користувача з програмою. Кожен процес супроводжується рисунками, котрий демонструє хід його роботи.

1 АНАЛІЗ ЗАДАЧІ ЗАХИСТУ ДАНИХ

На сьогоднішній день існує не мало систем захисту даних та алгоритмів шифрування зокрема. Проте, для кожного окремого випадку доречно використовувати той чи інший засіб. Важливо враховувати специфіку даних, що шифруватимуться та необхідний рівень захисту.

Сучасні системи захисту даних пропонують комплексні рішення для забезпечення безпеки інформації. Вони включають різноманітні технології, такі як шифрування, управління доступом, системи виявлення та запобігання вторгнень, а також засоби захисту на рівні мережі та хмари. Кожен з цих підходів має свої переваги та недоліки, і правильний вибір системи або їх ефективне поєднання забезпечує високий рівень захисту даних у сучасних інформаційних системах.

1.1 Мета програмного забезпечення

Мета розробки програмного забезпечення полягає у створенні функціональної системи захисту даних, що базується на алгоритмі шифрування Ель-Гамала.

Для досягнення мети необхідно виконати перелік наступних кроків:

- провести аналіз вимог до системи захисту даних, враховуючи специфічні потреби окремо взятого замовника;
- розробити архітектуру системи захисту даних, яка буде відповідати вимогам щодо конфіденційності, цілісності, доступності та безпеки [1];
- впровадити механізми автентифікації та авторизації користувачів, а також контролю доступу до даних;
- провести порівняння швидкодії обраного алгоритму з найбільш поширеним у сфері аналогом;
- провести ретельне тестування програмного забезпечення системи, щоб виявити та усунути помилки, вразливості та інші проблеми.

1.2 Вимоги до програмного забезпечення

До функціональних вимог відноситься наявність в програмі певних модулів, зокрема:

- модуль шифрування та розшифрування;
- модуль генерування та керування криптографічними ключами;
- модуль автентифікації та авторизації користувачів;
- модуль ведення журналів подій.

Список нефункціональних вимог виглядає наступним чином:

- швидкість: система повинна забезпечувати високу швидкість шифрування та розшифрування даних, щоб мінімізувати час очікування користувачів;
- ефективність використання ресурсів: система повинна ефективно використовувати обчислювальні ресурси, щоб мінімізувати навантаження на апаратну частину;
- конфіденційність: система повинна забезпечувати конфіденційність даних пацієнтів, використовуючи стійкий до атак алгоритм шифрування Ель-Гамала;
- шифрування даних: система повинна шифрувати всі дані пацієнтів, що передаються та обробляються;
- портативність: система повинна функціонувати на комп'ютерах з різними версіями поширених операційних систем.

1.3 Шифрування даних

Шифрування є ключовою технологією в захисті даних, забезпечуючи конфіденційність, цілісність та автентичність інформації [2]. Основна ідея полягає в тому, щоб зробити дані незрозумілими для будь-якої сторонньої особи, яка не має відповідного ключа або пароля для розшифрування. Тобто, навіть якщо стороння особа здобуде доступ до зашифрованих даних, вона не зможе їх прочитати без

відповідного ключа. Існує кілька видів шифрування, кожен з яких має свої особливості.

1.3.1 Симетричне шифрування

Симетричне шифрування - це метод шифрування, де той самий ключ використовується як для шифрування, так і для розшифрування даних. Процес симетричного шифрування можна описати наступним чином: відправник використовує ключ для перетворення звичайного тексту в шифрований формат перед його передачею, після чого отримувач використовує той самий ключ для розшифрування отриманого шифротексту, щоб отримати оригінальні дані.

Симетричне шифрування справляється зі своїм завданням досить ефективно та швидко, але вимагає безпечного обміну ключами між відправником і отримувачем, оскільки будь-яка сторона, яка отримає ключ, може розшифрувати всі дані, зашифровані за його допомогою. Тому застосування симетричного шифрування частіше відбувається в закритих системах, де забезпечена безпека обміну ключами. Найбільш поширеними алгоритмами симетричного шифрування є DES (Data Encryption Standard), AES (Advanced Encryption Standard), IDEA (International Data Encryption Algorithm). Різновидами симетричного шифрування є блоковий та потоковий шифри.

Блокове шифрування - це метод шифрування, в якому вхідні дані розбиваються на блоки фіксованого розміру, які потім обробляються незалежно один від одного. Процес шифрування являє собою поділ вхідних даних на блоки, використання ключа для перетворення кожного блоку даних в шифротекст, а потім об'єднання зашифрованих блоків в кінцевий зашифрований текст. При розшифруванні кожен зашифрований блок розшифровується окремо з використанням того самого ключа, і результати об'єднуються для відновлення вихідних даних.

Потокове шифрування - це метод шифрування, де дані шифруються біт за бітом під час їх передачі або зберігання. Основна ідея полягає в тому, що кожен біт вхідного тексту обробляється окремо за допомогою спеціального ключа і алгоритму шифрування. У потоковому шифруванні кожен біт вхідного тексту комбінується зі

станом потоку, що змінюється в залежності від вхідних даних і ключа. Це забезпечує високу швидкість обробки даних, адже шифрування відбувається паралельно з передачею чи зберіганням даних. Процес розшифрування в потоковому шифруванні відбувається аналогічно до процесу шифрування. Кожен біт зашифрованого тексту комбінується зі станом потоку, використовуючи той самий ключ і алгоритм, що і для шифрування, та відновлює оригінальні дані.

1.3.2 Хеш-функції

Хеш-функція - це функція, яка приймає вхідні дані будь-якого розміру і перетворює їх в вихідну послідовність фіксованої довжини, яка є унікальним хешем вхідних даних. Одним з основних застосувань є збереження паролів: замість зберігання паролів у чистому вигляді, системи зазвичай зберігають хеш-значення паролів, що дозволяє перевіряти відповідність паролів без зберігання самих паролів. Крім того, хеш-функції використовуються для перевірки цілісності даних: хеш-значення файлу може бути порівняне з хеш-значенням оригіналу, щоб визначити, чи були дані змінені. Хеш-функції також використовуються в криптографії для створення підписів даних та цифрових підписів.

1.3.3 Асиметричне шифрування

Асиметричне шифрування використовує пару ключів - відкритий та закритий. Для шифрування та розшифрування використовуються різні ключі, тому немає необхідності для користувачів погоджувати єдиний ключ через захищені канали передачі даних. Цей метод забезпечує високу безпеку та часто використовується для цифрових підписів і автентифікації. Проте він є повільнішим порівняно із симетричним шифруванням. Найбільш поширеними алгоритмами асиметричного шифрування є RSA та алгоритм Ель-Гамала.

1.3.4 Гібридне шифрування

Гібридне шифрування поєднує переваги асиметричного та симетричного шифрування. Асиметричне шифрування використовується для обміну симетричними

ключами, які потім застосовуються для шифрування даних. Це забезпечує високу безпеку обміну ключами та швидкість шифрування даних. Гібридне шифрування широко використовується в протоколах SSL/TLS для захищеного з'єднання в Інтернеті.

1.4 Інші системи захисту даних

Окрім шифрування, існують інші важливі методи захисту даних, кожен з яких відіграє свою роль у забезпеченні безпеки інформації.

Системи управління доступом забезпечують контроль над тим, хто і як може отримувати доступ до інформаційних ресурсів. Існує два основних підходи до управління доступом: на основі ролей та на основі атрибутів. Контроль доступу на основі ролей призначає користувачам певні ролі, і доступ до ресурсів надається відповідно до цих ролей. Це дозволяє централізовано керувати доступом в організаціях з великою кількістю користувачів. Контроль доступу на основі атрибутів є більш гнучким і надає доступ на основі характеристик користувачів, ресурсів та умов доступу, що дозволяє налаштовувати більш детальні політики доступу [4].

Системи виявлення та запобігання вторгнень є ще одним ключовим компонентом захисту даних. Системи виявлення вторгнень моніторять мережевий трафік або активність у системах, щоб виявити підозрілу діяльність та можливі атаки. Вони повідомляють адміністраторів про виявлені інциденти, що дозволяє вчасно реагувати на загрози. У свою чергу, системи запобігання вторгнень не лише виявляють, але й активно блокують та запобігають атакам у режимі реального часу. Це дозволяє автоматично реагувати на загрози та запобігати потенційним збиткам [5].

Захист на рівні мережі включає різні технології та практики, спрямовані на забезпечення безпеки передачі даних через мережі. Віртуальні приватні мережі (VPN) створюють захищене з'єднання через публічні мережі, забезпечуючи конфіденційність і цілісність переданих даних. VPN використовуються для безпечного з'єднання віддалених користувачів з корпоративними мережами та

захисту конфіденційної інформації під час передачі через Інтернет. Брандмауери є ще одним важливим інструментом захисту на рівні мережі. Вони фільтрують вхідний та вихідний трафік на основі встановлених правил безпеки, захищаючи системи від несанкціонованого доступу та атак [4].

1.5 Математична модель алгоритму шифрування Ель-Гамала

Алгоритм шифрування Ель-Гамала є асиметричним, складається з двох ключів: відкритого та закритого.

Тому в першу чергу необхідно згенерувати ключі, це відбувається наступним чином [3]:

1. вибирається велике просте число p (чим довше - тим надійніший ключ);
2. вибирається примітивний корінь g модулю p ;
3. вибирається випадкове число x в діапазоні $1 < x \leq p - 2$;
4. обчислюється y :

$$y = gx \bmod p \quad (1.1)$$

Таким чином відкритий ключ складається з p , g та y , а закритий ключ - x . Після того як отримувач згенерував ключі він публікує відкритий ключ, тобто робить видимим для відправника повідомлення. Відправник з шифрує своє повідомлення з використанням відкритого ключа.

Шифрування складається з наступних кроків:

1. нехай потрібно зашифрувати повідомлення M (довжина повідомлення обов'язково повинна бути коротшою за довжину p , інакше шифрування пройде некоректно);

2. вибирається випадкове число r , таке $1 < r \leq p - 2$ (сесійний ключ);
3. обчислюється c_1 :

$$c_1 = g^r \bmod p \quad (1.2)$$

4. обчислюється c_2 :

$$c_2 = M \cdot y^r \bmod p \quad (1.3)$$

Після шифрування повідомлення відправник надсилає шифротекст, який складається з c_1 та c_2 отримувачеві, який в свою чергу розшифровує його за допомогою свого приватного ключа:

$$s = C_1^x \bmod p \quad (1.4)$$

$$m = c_2 \cdot s^{-1} \bmod p \quad (1.5)$$

В результаті отримувач бачить повідомлення в такому вигляді, в якому його написав відправник до шифрування. Реалізація шифрування та розшифрування подана в додатку А.

1.6 Порівняння алгоритмів Ель-Гамалю та RSA

Дані алгоритми можуть застосовуватись для однакових цілей, проте вони мають між собою певні відмінності, що робить більш чи менш доцільним використовувати певний алгоритм у відповідній сфері. В таблиці 1.1 наведено порівняння алгоритмів за основними параметрами [3].

Таблиця 1.1 - Загальне порівняння алгоритмів Ель-Гамалю та RSA

Критерій	Алгоритм Ель-Гамалю	Алгоритм RSA
Математична основа	Дискретний логарифм	Факторизація великих чисел
Шифрування	Використання випадкового числа	Детерміноване
Розшифрування	Залежить від випадкового числа	Детерміноване
Довжина ключа	Рекомендується використовувати довжину більше 2048 біт [3]	Рекомендується використовувати довжину більше 2048 біт [3]
Продуктивність	Швидший у розшифруванні	Швидший у шифруванні
Асиметричність	Асиметричний	Асиметричний

Оскільки швидкодія алгоритму може прямо впливати на досвід користування, тому варто порівняти ці два алгоритми в даній площині. Для перевірки швидкості

алгоритмів на практиці було реалізовано обидва алгоритми шифрування та проведено шифрування/дешифрування певних текстів за використанням ключів різної довжини.

Виконання роботи алгоритмів проводилось на ПК з наступними характеристиками:

- CPU: AMD Ryzen 5500U;
- об'єм RAM: 16 ГБ DDR4;
- ОС: Windows 10;
- версія JDK: 17.

Щоб результати були більш точними та менш залежали від можливих випадкових затримок викликаних апаратним забезпеченням було проведено шифрування та розшифрування 4000 тисяч текстів для кожної довжини ключа. На рисунку 1.1 зображено результати роботи, порівняння середнього часу виконання операцій з відповідною довжиною ключа кожним алгоритмом.

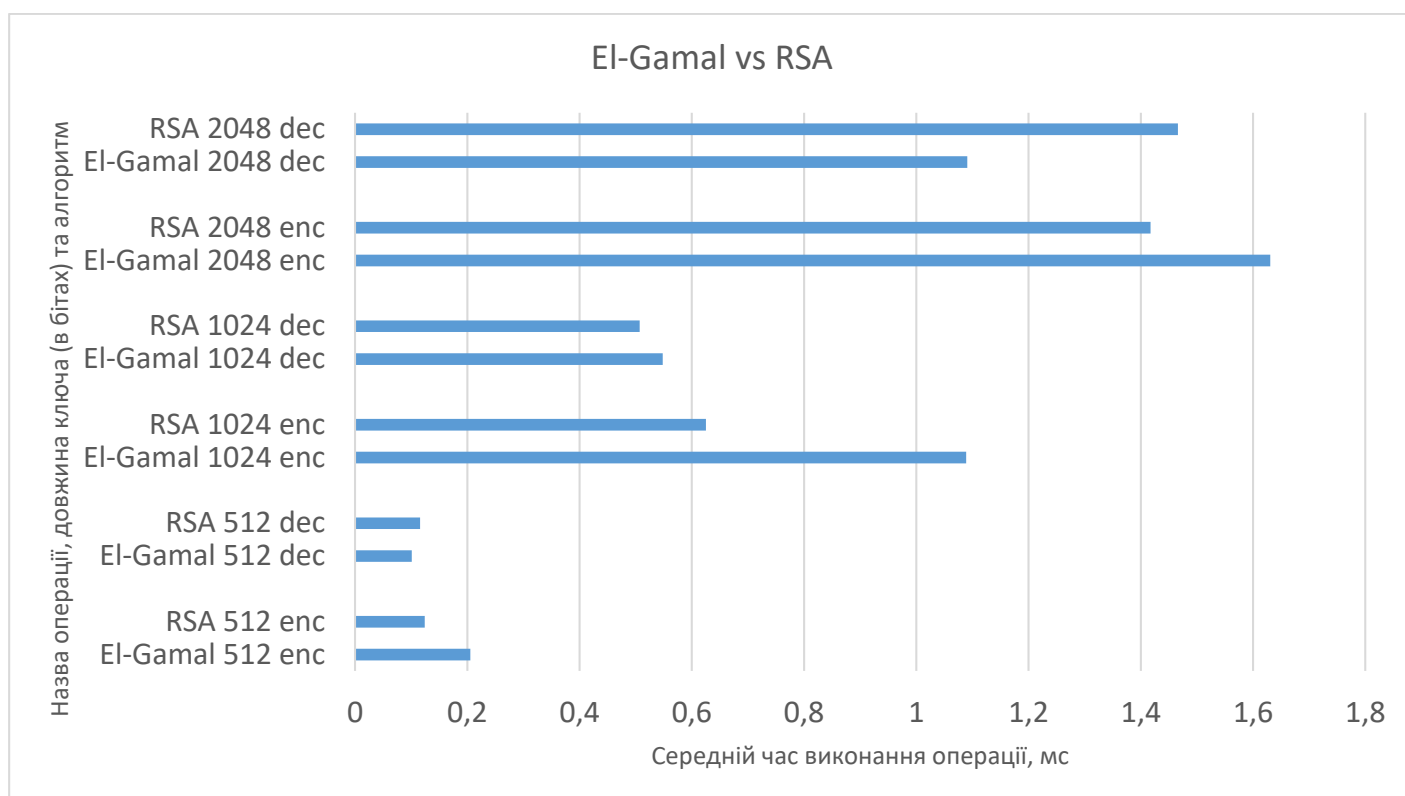


Рисунок 1.1 - Порівняння алгоритмів Ель-Гамала та RSA за часом виконання операцій

З даних, представлених на діаграмі, можна зробити висновок, що в більшості алгоритм Ель-Гамалі дійсно показує себе швидше в розшифруванні, але повільніше в шифруванні (що зумовлено використанням випадкового ключа для шифрування [6, 7]).

2 ЗАСОБИ РОЗРОБКИ

На сьогоднішній день існує багато засобів для зручної розробки програмного забезпечення. Проте, важливо обрати підходящі для задачі засоби, оскільки від них залежить не лише ефективність та продуктивність процесу розробки, а й якість та надійність кінцевого продукту.

2.1 Вибір мови програмування та середовища розробки

Оскільки виконується розробка desktop-застосунку з реалізацією графічного інтерфейсу, розробкою серверної частини та зв'язку з базою даних, тому мова програмування повинна мати необхідні інструменти для зручної реалізації всіх елементів програмного забезпечення.

Завдяки ретельному аналізу доступних мов програмування та рішень, Java було обрано як оптимальний інструмент для розробки проекту. Цей вибір ґрунтується на тому, що Java наразі є однією з найпоширеніших та затребуваних мов програмування. Її можливості виходять далеко за межі простої мови, адже Java перетворилася на потужну платформу та екосистему, що об'єднує різноманітні технології для розв'язання широкого спектру завдань - від створення десктопних програм до розробки масштабних веб-порталів та сервісів [9]. Окрім того, Java широко використовується для написання програмного забезпечення для різноманітних пристроїв: від звичайних ПК до планшетів, смартфонів, мобільних телефонів та навіть побутової техніки.

Переваги Java, що роблять її вигідним вибором:

- концепція ООП, завдяки якій код стає легко повторно використовувати в різних проектах, а також спрощує модернізацію вже існуючого;
- Java надає потужні можливості для роботи в багатопотоковому режимі, що робить її ідеальною для створення ефективних та масштабованих серверних додатків, які здатні одночасно обробляти великі обсяги даних від численних користувачів;

– принцип «написати один раз і використовувати скрізь» - програми, написані на Java, можуть без проблем функціонувати на різних операційних системах, таких як Windows, macOS та Linux. Це робить їх універсальними та доступними для ширшої аудиторії користувачів;

– для Java доступні численні інструменти та бібліотеки, які спрощують та пришвидшують процес розробки. Також Java має активну та залучену спільноту розробників, що полегшує пошук інформації та допомоги у вирішенні проблем;

– для запуску Java-додатків не потрібні складні налаштування. Все, що необхідно - це встановити JRE (Java Runtime Environment) - мінімальну реалізацію віртуальної машини Java, яка є невибагливою та легко інсталується.

При виборі IDE необхідно враховувати декілька факторів, які насправді можуть мати суб'єктивний характер. Перш за все, важливо враховувати потреби. Наприклад, для розробки невеликого проекту бути зручніше використовувати простіші середовища з меншою кількістю функцій, тоді як для більш масштабних проектів розробники можуть віддати перевагу потужним IDE з широкими можливостями.

Проте, на мою думку, функціональність IDE є ключовим фактором. Вона має підтримувати основні можливості, такі як зручне редагування коду, автоматичне завершення коду, рефакторинг, налагодження, інтеграцію з системами контролю версій, а також можливість роботи з різними фреймворками і бібліотеками. Також досить важливий і дизайн IDE, оскільки розробнику доводиться дивитись на вікно програми годинами.

Врахувавши всі вище перелічені фактори було вирішено зупинитися на IntelliJ IDEA від JetBrains. Це середовище розробки чудово підходить для реалізації поставлених задач, має більший функціонал та володіє значно більшою бібліотекою скриптів та плагінів. Приємний для ока та інтуїтивно-зрозумілий графічний інтерфейс (рисунок 2.1) є не меншою перевагою, тим більше, що існує можливість налаштовувати під себе елементи та тему інтерфейсу.

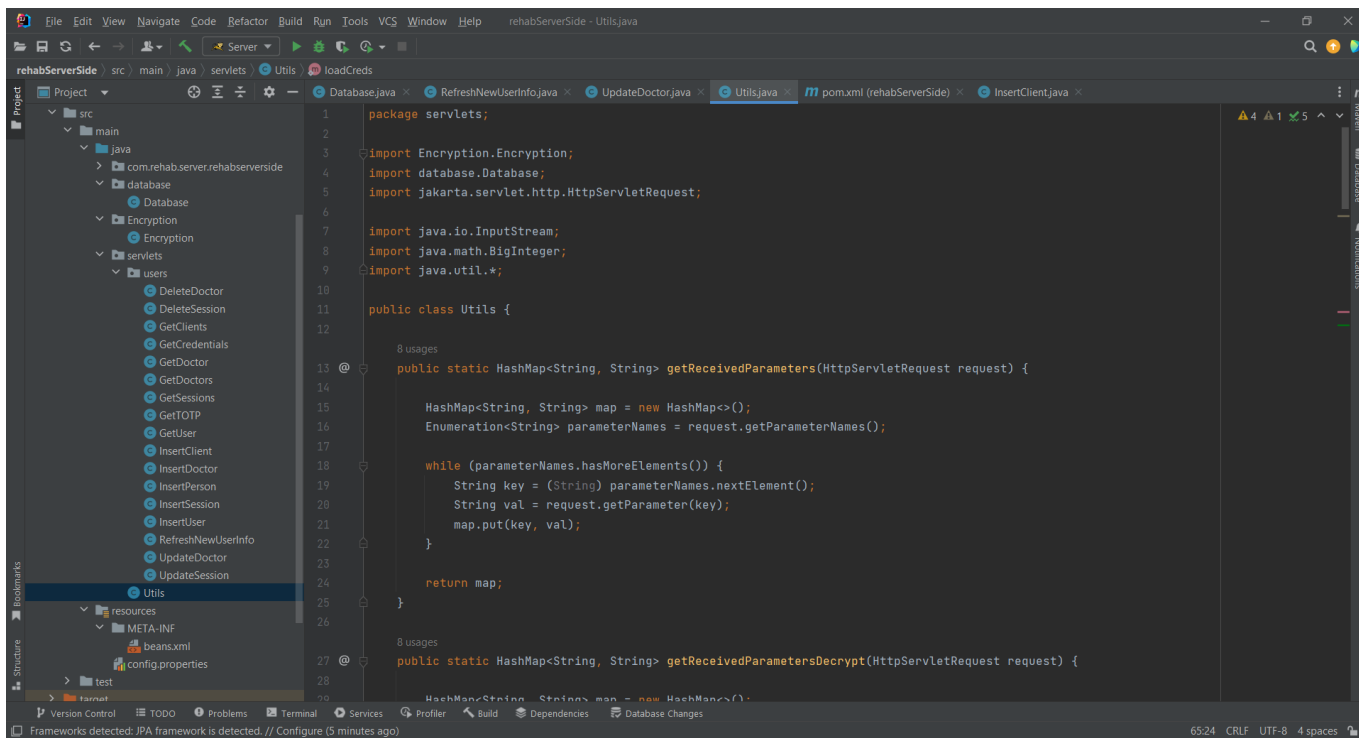


Рисунок 2.1 - Робочий простір IntelliJ IDEA

Також на вибір даного середовища розробки вплинуло те, що існує спеціальний тариф, завдяки якому студенти можуть безкоштовно отримати доступ до найпотужнішої версії ПЗ на період свого навчання.

2.2 Вибір системи керування базою даних

Система керування базами даних MySQL є однією з найпопулярніших у світі, поступаючись за популярністю лише Oracle протягом останнього десятиліття [10]. Така популярність MySQL обумовлена численними перевагами, які роблять її привабливим вибором для багатьох розробників і підприємств.

Однією з головних переваг MySQL є її простота у використанні. Ця система дозволяє швидко і легко налаштувати базу даних навіть новачкам завдяки зрозумілому інтерфейсу і доступній документації. Крім того, MySQL має велику кількість плагінів, які розширюють її функціональність, а процес інсталяції є нескладним і швидким [11].

Переважна більшість функцій, які відповідають за безпеку, автоматично встановлюються та працюють за замовчуванням. Це дозволяє знизити ризики і забезпечити високий рівень захисту даних без додаткових налаштувань. MySQL також вирізняється високою швидкістю роботи завдяки оптимізації і спрощенню деяких процесів. Це робить її швидшою за деяких конкурентів і дозволяє ефективніше обробляти запити до бази даних. Крім того, MySQL підтримує різні платформи і може працювати на різних операційних системах, що забезпечує гнучкість і зручність інтеграції з іншими системами [12, 13].

Хоча у MySQL є і недоліки, одним з яких є проблеми при роботі з дуже великими базами даних, для багатьох користувачів це не стане суттєвою проблемою. Для вирішення поставлених задач у цій роботі, наприклад, це не буде мати значного впливу на продуктивність і ефективність роботи.

2.3 Інструменти розробки клієнтського застосунку

Основним інструментом при розробці клієнтської частини ПЗ є JavaFX — сучасний фреймворк для створення графічних інтерфейсів користувача на Java. Він призначений для побудови додатків з використанням набору інструментів для розробки графічних елементів, що забезпечують високу продуктивність, що робить додатки плавними та швидкими навіть на пристроях з обмеженими ресурсами, і привабливий зовнішній вигляд. JavaFX має кілька важливих особливостей, котрі роблять його популярним вибором серед розробників.

Головне те, що цей фреймворк пропонує сучасні можливості для створення графічних інтерфейсів, такі як анімації, стилізація з використанням CSS, підтримка мультимедіа та 2D/3D-графіки. Це дозволяє розробникам створювати візуально привабливі та інтерактивні додатки.

CSS (Cascading Style Sheets) - це мова опису стилів, яка використовується для оформлення веб-сторінок і керування їх зовнішнім виглядом. Вона визначає, як елементи повинні відображатися у браузері, включаючи кольори, шрифти, розміри,

відступи, позиціонування та інші аспекти. Зокрема для дизайну застосунку була використана бібліотека JMetro [14], котра базується на CSS. Цей інструмент дозволяє швидко та просто зробити дизайн інтегрованим під Windows, при тому мати хороший вигляд і на інших ОС.

Крім того, JavaFX підтримує FXML - XML-формат для визначення інтерфейсу користувача, що дозволяє відокремити логіку програми від її представлення і робить код більш організованим і легким для підтримки. Для роботи з файлами такого розширення існує зручний додаток Scene Builder.

Scene Builder - це інструмент для візуального редагування інтерфейсів користувача для програм, що базуються на платформі JavaFX. Він дозволяє розробникам створювати та налаштовувати графічні інтерфейси за допомогою перетягування та розміщення елементів у вікні, не пишучи код вручну.

2.4 Інструменти розробки серверної частини ПЗ

Технологія сервлетів та сервер Tomcat задовольняють постевлені завданням потреби.

Сервлети - це Java-класи, які використовуються для розробки веб-додатків та додатків з подібним функціоналом. Вони виконуються на стороні сервера і відповідають за обробку запитів від клієнтів. Сервлети зазвичай використовуються для створення динамічних веб-сторінок, обробки форм, взаємодії з базами даних тощо. Сервлети запускаються в контейнері сервлетів, який надає середовище для їх виконання.

Apache Tomcat є одним з найпопулярніших контейнерів сервлетів та серверів веб-додатків для платформи Java. Tomcat є відкритим програмним забезпеченням і забезпечує різноманітні можливості для розгортання, керування та виконання веб-додатків, написаних на Java. Він включає в себе контейнер сервлетів, який відповідає за обробку та виконання сервлетів, а також підтримує JavaServer Pages (JSP) - ще одну технологію для створення динамічних веб-сторінок. Tomcat зазвичай

використовується як сервер додатків для веб-додатків Java EE (Java Enterprise Edition) та відкритих стандартів, таких як Servlet API та JavaServer Pages. Він може бути використаний як самостійний сервер або в поєднанні з іншими веб-серверами, такими як Apache HTTP Server, які використовуються для обробки статичного вмісту. Tomcat надає зручний і ефективний спосіб розгортання та управління веб-додатками на платформі Java. Запити до серверу надсилаються за допомогою HTTP протоколу.

Протокол HTTP (Hypertext Transfer Protocol) є стандартом для передачі гіпертекстових документів, таких як веб-сторінки, у мережі Інтернет [9]. Він використовується для взаємодії між клієнтом та сервером. Цей протокол визначає, як запити відправляються від клієнта до сервера, а також як сервери відповідають на ці запити, передаючи необхідні дані. За наявності та необхідності пересилати дані, вони відправляються на сервер або з сервера користувачеві у JSON-форматі.

JSON (JavaScript Object Notation) - це легкий формат обміну даними, який є зрозумілим для людей і легко обробляється машинами. Його структура базується на підмножині мови програмування JavaScript, але він незалежний від неї, тому може використовуватися в різних мовах програмування. Формат JSON складається з двох основних структур: об'єктів та масивів. Об'єкти представляються у вигляді пар "ключ-значення", де ключі - це рядки, а значення можуть бути різних типів, таких як рядки, числа, булеві значення, інші об'єкти або масиви. Масиви - це впорядковані списки значень, які можуть містити будь-які з перелічених типів даних.

2.5 Допоміжні інструменти розробки

Для розгортання програмного забезпечення на сервері використовується Maven - інструмент управління проєктами та система автоматизації збірки. Він значно спрощує процес деплоюменту, звільняючи розробників від рутинних задач, забезпечуючи централізоване керування залежностями, життєвим циклом проєкту та стандартизацією процесів збірки.

Основна мета Maven - зробити управління проєктами більш ефективним і організованим. Maven використовує файл конфігурації, відомий як POM (Project Object Model), у якому визначаються всі необхідні залежності, плагіни та конфігураційні параметри проєкту. Це дозволяє розробникам уникнути прямого керування бібліотеками та плагінами, оскільки всі залежності автоматично завантажуються з центральних репозиторіїв. Завдяки широкому обсягу існуючих плагінів за допомогою Maven можна зібрати і клієнтську частину.

Для контролю над версіями програми використовується система Git (зокрема сервіс GitHub). Git - це розподілена система керування версіями, яка використовується для відстеження змін у файловій системі. Вона дозволяє розробникам спільно працювати над проєктами, зберігаючи копії коду та історію змін.

Git забезпечує такі можливості:

- дозволяє зберігати кілька версій файлів;
- дозволяє створювати гілки, що дозволяють розробникам працювати над різними функціональностями паралельно;
- дозволяє синхронізувати робочий каталог з віддаленим репозиторієм;
- зберігає історію комітів;
- За допомогою файлу .gitignore можна вказати Git, які файли та каталоги необхідно ігнорувати при збереженні змін.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ ЗАХИСТУ ДАНИХ

Програмна реалізація проекту охоплює детальний опис архітектури, методів та алгоритмів, які використовувалися для створення системи. У цьому розділі буде розглянуто ключові компоненти програмного забезпечення, їхні функції та взаємодію між собою.

3.1 Архітектура ПЗ

Архітектура програмного забезпечення - це структура системи, яка визначає її основні компоненти та взаємодію між ними [15]. Вона існує для організації коду та забезпечення зрозумілості, гнучкості та супроводження системи. Архітектура допомагає розробникам зрозуміти, як різні частини програмного забезпечення працюють разом і як вони можуть бути змінені або розширені. Вона також сприяє кращому розподілу завдань в команді розробників при розробці масштабних проектів і полегшує інтеграцію нових функцій.

Оскільки поставлені завдання вимагають розробку окремо клієнтського інтерфейсу та окремо серверної частини, що функціонуватиме незалежно, необхідно обрати архітектуру, яка задовольняє цим вимогам. Можна побудувати свою унікальну модель архітектури, проте провівши огляд основних архітектурних шаблонів та стало зрозуміло що під поставленні завдання дуже добре підійде MVC-шаблон.

Модель - вид - контролер (MVC, англ. Model-view-controller) - архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення, що передбачає поділ системи на три взаємопов'язані частини [16]. В рамках архітектурного шаблону MVC, програма розбивається на три взаємозалежні, але окремі частини з різними функціями (рисунок 3.1).

Цими частинами є:

- модель є центральним елементом шаблону MVC і відображає поведінку програми, незалежно від інтерфейсу користувача. Вона відповідає за управління

даними, логікою та правилами програми. Існують два типи моделей: активна, яка відстежує зміни в моделі та реагує на них, і пасивна, яка оновлюється через контролер;

- вигляд може приймати різні форми відображення інформації, проте його функція - відображає ці дані для користувача, представляючи інтерфейс програми;

- контролер отримує вхідні дані від користувача та перетворює їх на команди для взаємодії з моделлю чи виглядом.

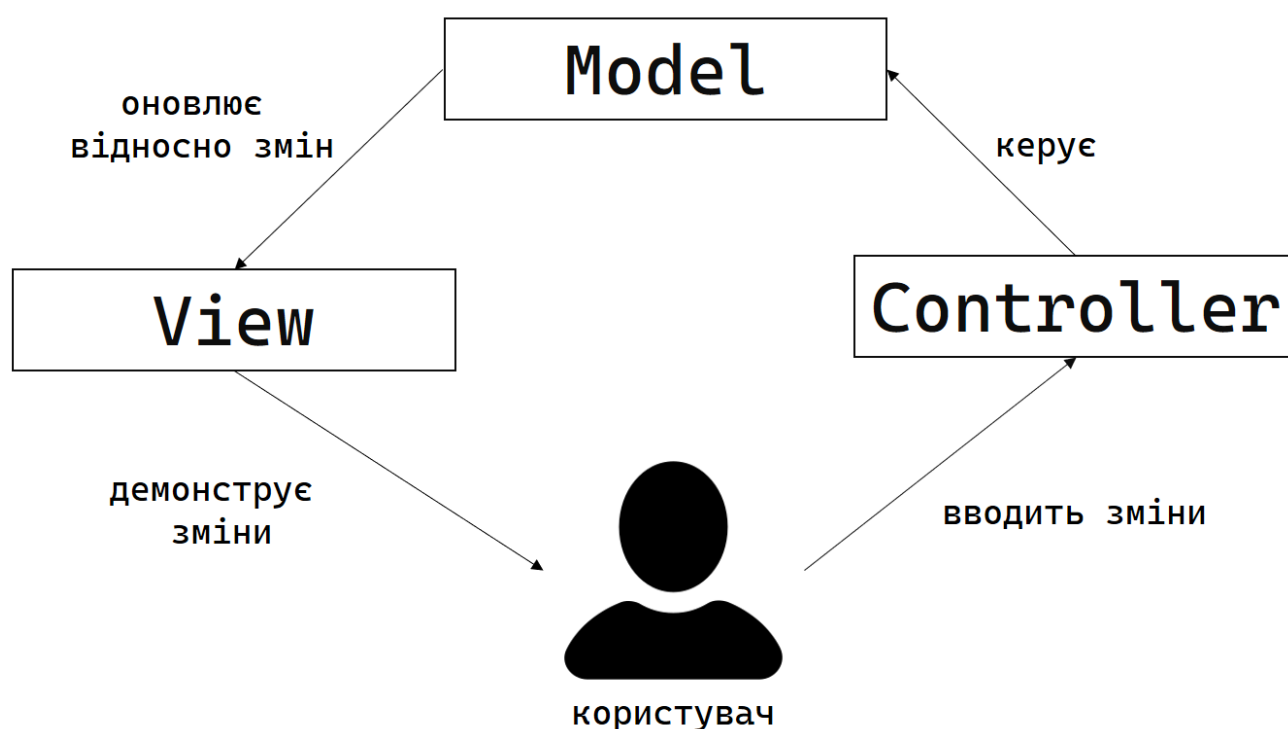


Рисунок 3.1 - Діаграма взаємодії компонентів шаблону MVC

Спершу використовувався для настільних графічних інтерфейсів користувача (GUI) для desktop-застосунків, але з часом цей шаблон став популярним і для розробки веб-додатків. Оскільки цей шаблон використовується розробниками досить часту, то зрозуміло що популярні мови програмування мають фреймворки MVC, які полегшують реалізацію шаблону (Ruby on Rails, Laravel, Spring MVC, ASP.Net Core MVC).

Створення запитів та обмін інформацією між клієнтом та сервером виконується за допомогою HTTP протоколу. Безпеки цього протоколу цілком достатньо, оскільки виконується передача зашифрованих даних та публічного ключа.

3.1.1 Клієнтська сторона

На клієнтській стороні відбувається вся логіка з взаємодією користувача з програмою. На рисунку 3.2 зображено перелік класів клієнтської сторони, що забезпечують роботу програми.

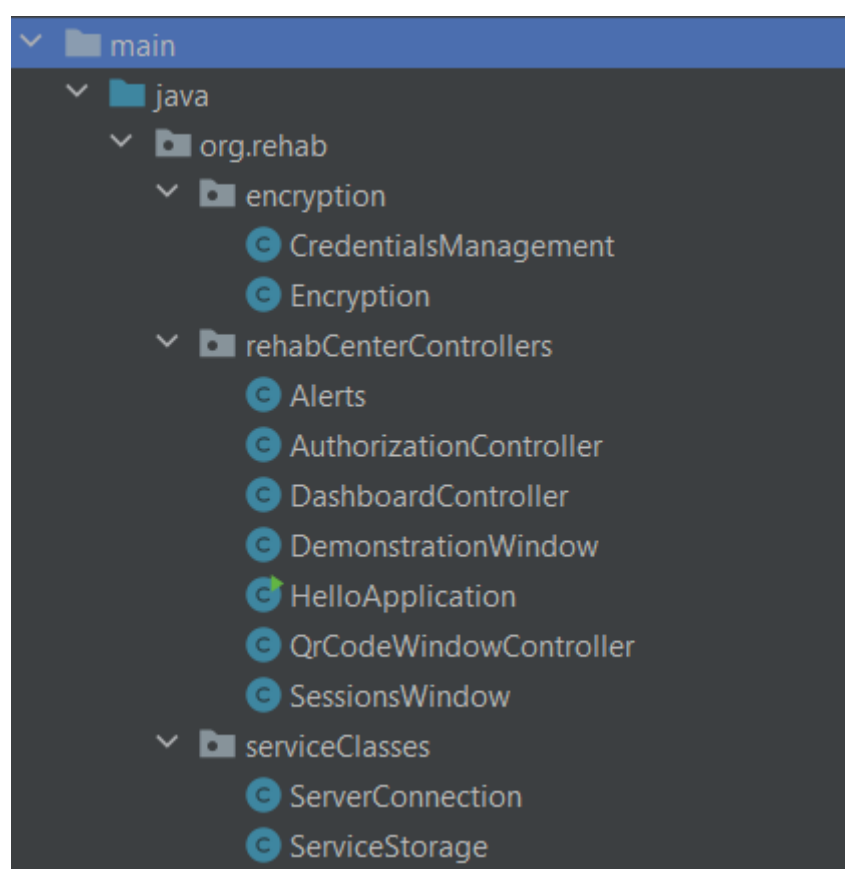


Рисунок 3.2 - Структура клієнтської сторони

В пакеті encryption містяться класи, що відповідають за шифрування та розшифрування інформації та контроль особистих даних користувача. В пакеті rehabCenterControllers містяться класи, в яких описується взаємодія графічного інтерфейсу з користувачем. В пакеті serviceClasses містяться класи, що відповідають

за створення та обробку запитів до сервера. Класи даного пакету загалом виконують функцію компоненти Controller шаблону MVC.

На рисунку 3.3 зображено перелік .fxml файлів, які служать для відмальовки користувацького інтерфейсу, прив'язки його компонентів до методів контролера.

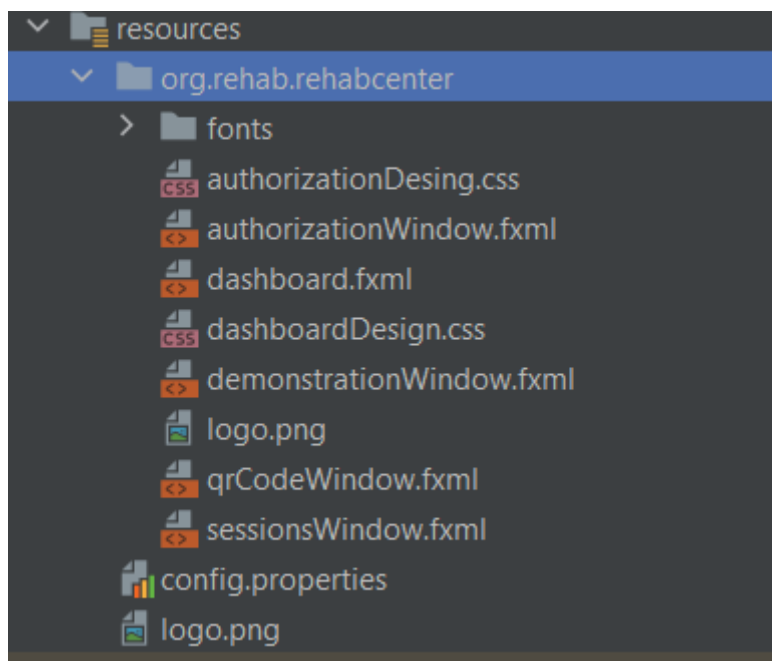


Рисунок 3.3 - Компоненти користувацького інтерфейсу

FXML є декларативною мовою, яка використовується в JavaFX для опису інтерфейсів користувача [17]. Вона дозволяє визначити графічний інтерфейс додатка за допомогою XML-подібного синтаксису. FXML використовується для розділення структури користувацького інтерфейсу від логіки програми, що спрощує розробку та підтримку програмного забезпечення[18].

Для можливості імпорту кастомних іконок та інших шрифтів у користувацький інтерфейс використовується fontawesomefx, який надає велику бібліотеку іконок та інструмент для роботи з шрифтами в GUI.

Загалом до переліку функцій клієнтської сторони можна віднести:

- візуалізація даних у вигляді графічного інтерфейсу;

- обробка дій користувача (введення даних, перемикання вікон, натискання клавіш);
- автентифікація користувача;
- створення запитів до сервера;
- обробка відповідей від сервера;
- шифрування даних перед створенням запиту та розшифрування відповіді від сервера.

3.1.2 Серверна сторона

На серверній стороні програмного забезпечення відбувається вся логіка обробки запитів користувача та взаємодії з базою даних. На рисунку 3.4 зображено перелік класів, що забезпечують роботу сервера.

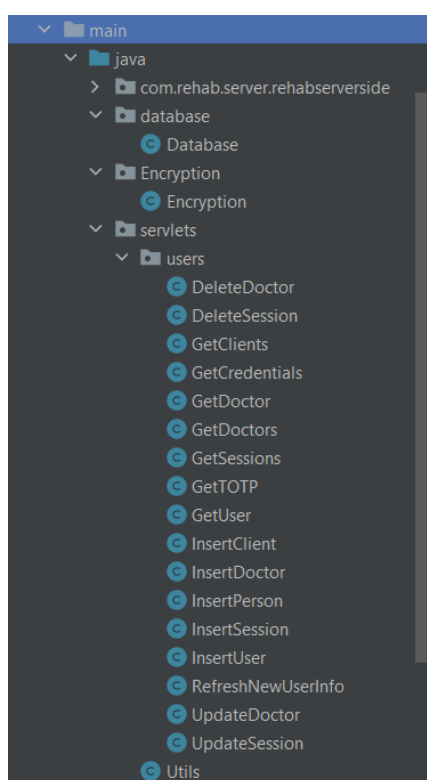


Рисунок 3.4 - Структура серверної частини

В пакеті encryption містяться класи, що відповідають за шифрування та розшифрування інформації. В пакеті database містяться класи, в яких відбувається взаємодія з базою даних. В пакеті servlets містяться сервлети, що відповідають за обробку запитів до сервера та клас Utils, який містить методи для форматування інформації перед шифруванням та відправкою.

Загалом до переліку функцій серверної сторони можна віднести [19]:

- обробка запитів клієнта;
- взаємодія з базою даних;
- надання відповідних прав користувачеві при автентифікації;
- шифрування даних перед відправкою та розшифрування при отриманні від користувача;
- створення відповідей на запити користувача.

3.3. Візуалізація функціональних вимог

На рисунку 3.5 зображена діаграма прецедентів, на якій продемонстровано як система взаємодіє з її користувачами та які операції вона виконує для задоволення їх потреб.

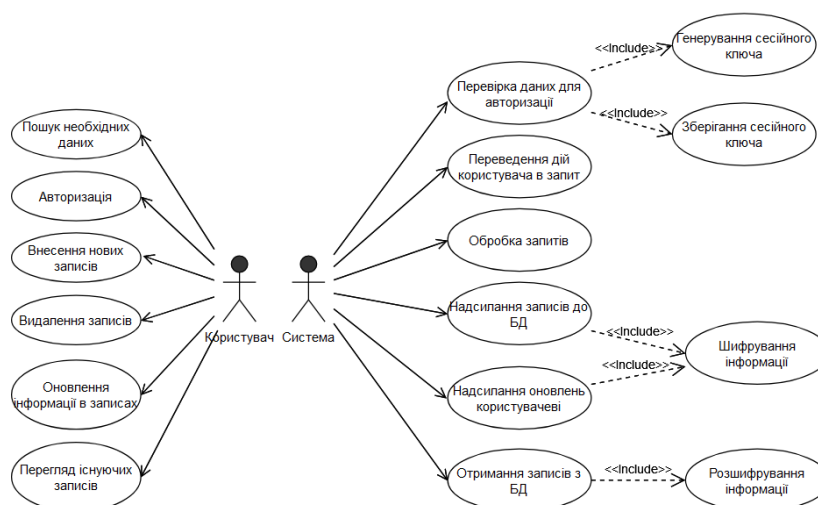


Рисунок 3.5 - Діаграма прецедентів

З діаграми можна зробити висновок, що головними функціями користувача є полягають у взаємодії з графічним інтерфейсом, а система обробляє ці дії, звертається до бази даних та демонструє користувачеві результат його запиту. Всі ці дії супроводжуються шифруванням/дешифруванням системою захисту даних. На рисунку 3.6 зображено тіло відповіді від сервера клієнту, в якому містяться дані про лікарів.

```
{
  "array": [
    {
      "address": "6257335531962685922248639130537402158199641112502789801740969415423368916649562422214442185386635420192001709828900205622746955422721117430355441339102510334817788881533054948731373473628462607891762641361512735202989419455953164317588548898825612225879846845852838714347523542149281771262195719545406201502638907145856239212415326159216202041528582792049800974020581602092422890885046176468580561809719641233996406480346475039614002662066857007573933191",
      "gender": "[252122341368968529497474105141089077930014260293060191663184764715195081359279589988440483536952691407896383540493474218624331772191647523057809",
      "date_of_birth": "[835159611844011346234709301501529528687770043761702524194963842608105049563371763224802454585634369319650928311467216286966514724618202675",
      "photo": "[152689521271416440475452772713752155647076247349817601170667750455769330712229882338422975451444980812999322217853388698700783618709455190185442",
      "person name": "[75891973887725635956039256833566850231204715974437207342219737442371169353976221289526855825989413736062994350745903760067257206972610393",
      "doctor id": "[13285963190743865101278737517548373902795008544985229456053449101548206623968472839830231792039209755413367772349620369662465418799192867425",
      "patronymic": "[752139277425805984143361145028876055122202718413956625699994157361560659709843824174631978520693518798022844435500580312317528590131609864",
      "surname": "[9869917733967795418200282250002895172037026360206000451217995945660533777957033023734490077983743511021884077359667809094377598967331674396003",
      "session key": "[9897690739593424912434026942268014154458965848427117364376240891271334730972252709755770021321228199669955815184166383126032588993528866610",
      "specialization": "[579126777072972581021350967152343139942450270678500364002325633898428938555353495581209920595904691824878979316586143222790299048255024",
      "phone number": "[14318172982293379690745878062019539719665386709805405629447715776656773341454308037654983510435882346310248239552999386162900010511350832",
      "email": "[328185801156144422883842894996030728639642337109136385013265277303435699448128454802182947313613784668253410339450175993208713528970302815164530",
      "person id": "[8511515001888869463458818649333608204995909270789062447773084223213914851870033421622773157196707251622859119507482861725147990574966516231",
    }
  ]
}
```

Рисунок 3.6 - Приклад відповіді сервера клієнту

Відповідь приходить в зашифрованому вигляді в JSON-форматі як масив JSON-об'єктів. Після отримання відповіді відбувається розшифрування даних та оновлення елементів користувацького інтерфейсу (заповнення таблиць, очищення полів вводу після успішного додавання запису, оновлення фотографії тощо). Код генерування запитів та відповідей, шифрування та дешифрування міститься в додатку А.

3.4 Архітектура бази даних

База даних є реляційною та поділяється на дві основні частини: перша частина містить всі основні дані складається з 4 таблиць, друга частина містить дані про ключі

для шифрування та автентифікації користувачів, містить одну таблицю. Поділ бази даних на дві частини зумовлений тим, щоб покращити захист до її вмісту. Для того, щоб зломисник отримав розшифровані дані йому необхідно отримати доступ до обох частин бази даних, інакше він або отримає просто ключі шифрування, або лише зашифровані дані. На рисунку 3.7 зображено ER-діаграму першої частини бази даних.

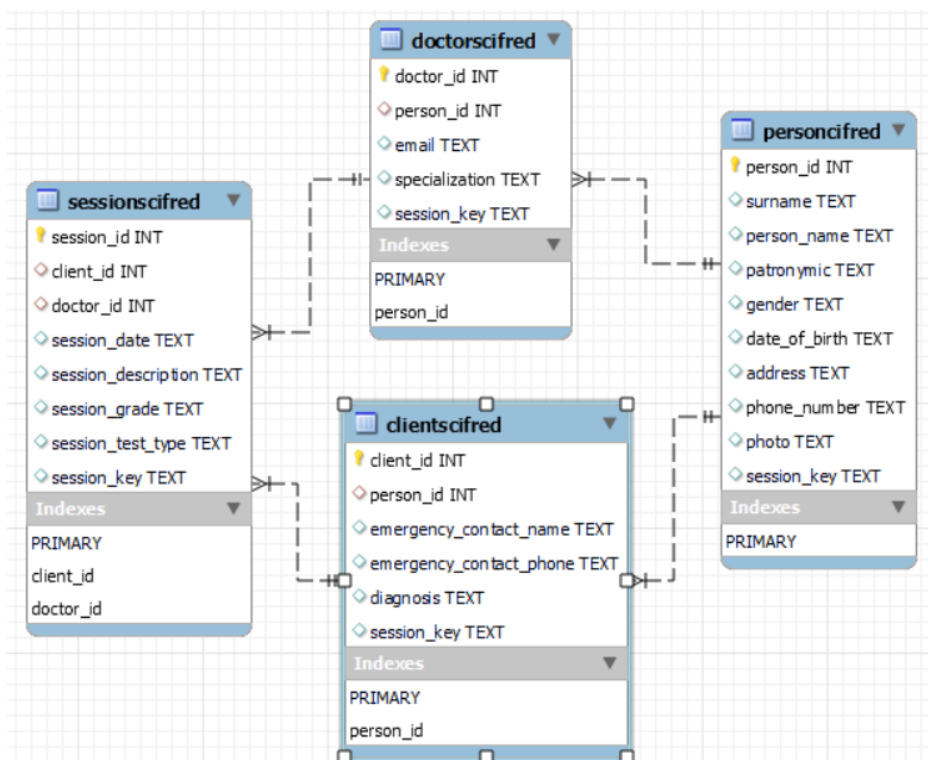


Рисунок 3.7 - ER-діаграма першої частини бази даних

Таблиця personCifred містить загальну інформацію про особу, використовується для створення записів про клієнтів та персонал. Поля таблиці детально описані в таблиці 3.1.

Таблиця 3.1 - Опис таблиці personCifred

Назва поля	Опис поля
person_id	унікальний ідентифікатор особи

Кінець таблиці 3.1

surname	прізвище особи
person_name	ім'я особи
patronymic	по-батькові особи
gender	стать особи
date_of_birth	дата народження особи
address	адреса особи
phone_number	номер телефону особи
photo	шлях до фотографії особи

Таблиця clientsCifred містить дані про клієнтів центру, її дані слугують основою для запису проведених психологічних сесій. Всі поля детально описані в таблиці 3.2.

Таблиця 3.2 - Опис таблиці clientsCifred

Назва поля	Опис поля
client_id	унікальний ідентифікатор пацієнта
person_id	зовнішній ключ, що посилається на personcifred.person_id
doctor_id	зовнішній ключ, котрий вказує на ідентифікатор лікаря, який займається лікуванням пацієнта, посилається на doctorscifred.doctor_id
emergency_contact_name	ім'я контактної особи для надзвичайних випадків
emergency_contact_phone	номер телефону контактної особи для надзвичайних випадків
diagnosis	діагноз пацієнта

Таблиця doctorsCifred містить дані про персонал закладу, через поле email надається доступ до відповідних функцій та запис ключів. Поле email також використовується для автентифікації користувача в системі та генерування QR-кодів

для входу, це необхідно для підвищення безпеки та унеможливорює доступ до програми небажаних осіб. Поля описані в таблиці 3.3.

Таблиця 3.3 - Опис таблиці doctorsCifred

Назва поля	Опис поля
doctor_id	унікальний ідентифікатор лікаря
person_id	зовнішній ключ, що посилається на personcifred.person_id
email	електронна адреса лікаря
specialization	спеціалізація лікаря

Таблиця sessionsCifred містить дані про проведені психологічні сесії та заходи. Поля описані в таблиці 3.4.

Таблиця 3.4 - Опис таблиці doctorsCifred

Назва поля	Опис поля
session_id	унікальний ідентифікатор сесії
person_id	зовнішній ключ, котрий вказує на ідентифікатор пацієнта, посилається на clientscifred.client_id
doctor_id	зовнішній ключ, котрий вказує на ідентифікатор пацієнта, що посилається на doctorscifred.doctor_id
session_date	дата сесії
session_description	опис сесії
session_grade	оцінка сесії
session_test_type	тип тесту сесії

Також необхідно зазначити, що в кожній таблиці є поле session_key, який зберігає сесійний ключ користувача, котрий вносить записи до бази даних. Це слугує для подальшого можливого відновлення даних.

Друга частина бази даних слугує для збереження ключів користувачів та проведення автентифікації при запуску програми. На рисунку 3.8 зображено вигляд таблиці users.

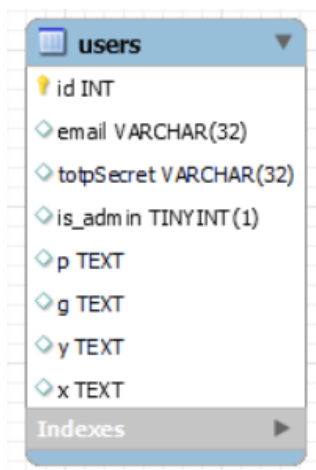


Рисунок 3.8 - Таблиця users

Всі поля таблиці описано в таблиці 3.5.

Таблиця 3.5 - Опис таблиці users

Назва поля	Опис поля
поля p, g, y	частини публічного ключа користувача
x	приватний ключ
email	електронна пошта користувача, є такою ж як і в таблиці doctorsCifred та слугує для генерування одноразових паролів для автентифікації
is_admin	чи має користувач права адміністратора
totpSecret	секретний ключ для автентифікації користувача

Використання поділу бази даних зумовлено тим, щоб зробити важчим задачу компрометації зашифрованого тексту для зловмисника, навіть якщо він отримав доступ до інформації в одній з таблиць.

3.5 Процес автентифікації користувача в системі

Для підвищеного захисту від несанкціонованого доступу до інформації, котра зберігається в базі даних було вирішено додати автентифікацію користувача. Для цього використовується TOTP алгоритм.

TOTP (Time-based One-Time Password) - це алгоритм, який генерує одноразові паролі, прив'язані до часу. Цей метод використовується для додаткової автентифікації користувачів, забезпечуючи більшу безпеку при вході в систему [20].

TOTP працює за принципом синхронізації з поточним часом. На основі секретного ключа і поточного часу обчислюється хеш, який потім перетворюється в одноразовий пароль [21]. Пароль генерується і є дійсним протягом 30 секунд від моменту створення, що унеможлиблює його використання в якийсь інший момент часу. В базі даних зберігаються секретні ключі кожного користувача (рисунок 3.9), який здійснив свій перший вхід до програми після створення адміністратором його облікового запису.

email	totpSecret
1	DSR6MZKXJKIFQWBETWDXDNIF6NN4U2NF
admin	XSWE4R7REYQTHE34YWYL3AY7CYZRXMFF
yulia0205k@gmail.com	NULL
mightynaz@protonmail.com	NULL
luzkoanalii@ukr.net	NULL
luzkoanalii@ukr.net	NULL
test@ukr.net	FDVYMCM6GD57W3CAO3IK77LRVUKITKML
111@ukr.net	MBXJJCZ4FDH3XDRTKIZDEW7Q6XBUSF3P

Рисунок 3.9 - Зберігання секретних ключів для автентифікації

За замовчуванням при створенні нового облікового запису значення ключа стає рівним null, зміниться воно після першого входу за вказаною поштою при скануванні QR-коду для генерації одноразових паролів на телефоні користувача (або ж іншого пристрою, на якому налаштована генерація паролів за цим алгоритмом).

3.6 Процес збірки програмного забезпечення

Для того щоб весь написаний програмний код зробити можливим для запуску на інших системах необхідно провести збірку ПО (build). Для Java існує два основних для цього інструменти - Maven та Gradle. Обидва інструменти мають схожий набір функцій та характеристик, в результаті було обрано Maven для виконання даного завдання.

Оскільки в програмі використовуються сторонні від JDK бібліотеки, то при збірці ПЗ необхідно включити до результуючого продукту перелік залежностей, котрі дозволять завантажити відповідну бібліотеку до продукту. Для того щоб додати залежність, необхідно прописати її в pom.xml файл, котрий надає Maven всю необхідну інформацію про проект. Збірка програмного забезпечення з використанням JavaFX потребує окремого скрипта, який збере програму з можливістю взаємодії з графічним інтерфейсом. Для того щоб додати залежність, необхідно прописати її в pom.xml файл, котрий надає Maven всю необхідну інформацію про проект. Згадані вище скрипти знаходяться в додатку А.

Використання Maven та існуючих для нього скриптів автоматизує процес збірки програмного забезпечення, значно полегшує життя розробників на фінальних етапах розробки та дозволяє в подальшому легко розгортати програмний продукт.

4 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

В даному розділі описано системні вимоги та інсталяцію програми, головні елементи взаємодії користувача з програмою.

4.1 Інсталяція та системні вимоги

Клієнтська частина програмного продукту має вигляд jar-файлу, який необхідно запустити для роботи з програмою. Після цього запуститься вікно програми яке повністю готове до роботи. Для безперебійної та комфортної роботи в програмі мінімальні вимоги до системи наведені в таблиці 4.1.

Таблиця 4.1 - Мінімальні апаратні вимоги для клієнтської частини

Назва	Характеристика
Центральний процесор	Intel® Core™ i3 або AMD® Athlon™ Caspian , тактова частота - не нижче 1.5 GHz
Об'єм оперативної пам'яті	2 ГБ
Мінімальний об'єм вільного простору на диску	200 МБ
Підтримка видів апаратної архітектури	Підтримуються 32- і 64-бітні системи
Швидкість Інтернет-з'єднання	128 КБ/с

Операційна система, повинна бути Windows 10+, MacOS 10+, встановлена версія Java 11+.

Для розгортання на сервері необхідно запустити jar файл, після чого програма створить необхідні директорії, таблиці в базі даних та завантажить необхідні залежності. Щоб це все успішно було виконано, програмі необхідно мати достатньо прав у системі. Для ОС Windows достатньо запустити ПЗ від імені адміністратора. Або ж для Unix-подібних систем необхідно використати команду, яка дозволяє

користувачеві виконувати команди з привілеями суперкористувача (наприклад “sudo”). Про успішне розгортання на сервері буде сповіщено в консолі (рисунок 4.1)

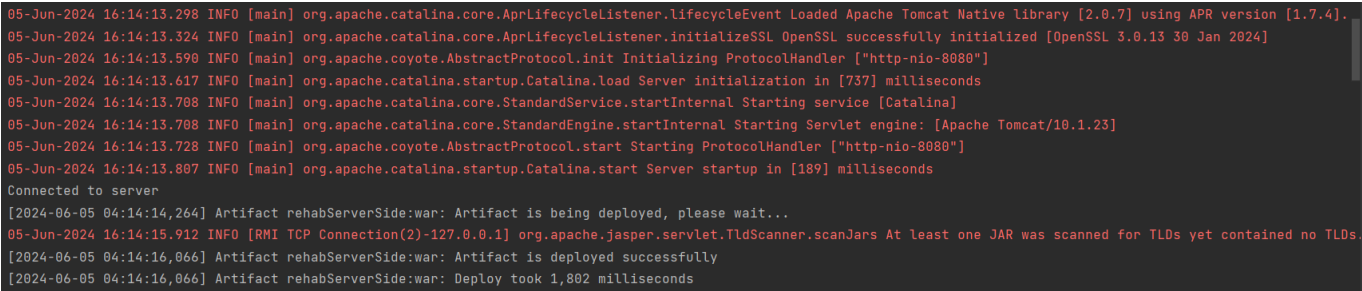


Рисунок 4.1 - Результат успішного розгортання серверної частини ПЗ

Звичайно, для роботи серверу він повинен мати доступ до Інтернету. Мінімальні вимоги до апаратного забезпечення наведено в таблиці 4.2. Вимоги є дуже “демократичними” та не змушують витрачати велику кількість коштів на потужний сервер, що робить можливим використовувати персональний компютер або ноутбук минулих років у якості сервера. Проте, для більш стабільної роботи та забезпечення стабільності рекомендується використовувати спеціалізовані для цього завдання конфігурації.

Таблиця 4.2 - Мінімальні апаратні вимоги для серверної частини

Назва	Характеристика
Центральний процесор	Intel® Core™ i3 або AMD® Athlon™ II, тактова частота - не нижче 2.5 GHz
Об’єм оперативної пам’яті	2 ГБ
Мінімальний об’єм вільного простору на диску	15 ГБ
Підтримка видів апаратної архітектури	Підтримуються 32- і 64-бітні системи
Швидкість Інтернет-з’єднання	1 МБ/с

Перелік необхідного програмного забезпечення сервера для коректної роботи розробленої програми наведено в таблиці 4.3.

Таблиця 4.3 - Мінімальні вимоги до ПЗ серверної частини

Назва	Характеристика
Операційна системи	Windows 10+, MacOS 10+, Ubuntu 18.04+, Debian Jessie
Необхідне установлене програмне забезпечення	СКБД MySQL 5.6+, Java 11+, Apache Tomcat 9+

Система не вимагає встановлювати дуже велику кількість стороннього ПЗ, що є безумовним плюсом та робить можливим встановлення та запуск роботи на сервері навіть для не дуже досвідчених користувачів.

4.2 Автентифікація користувача в системі

Після запуску програми користувача зустріне вікно входу, в якому необхідно вказати свою адресу електронної пошти та одноразовий пароль, який відображається в користувача в застосунку Google Authenticator (рисунок 4.2). В разі введення неправильних даних користувачеві буде запропоновано ввести їх заново.

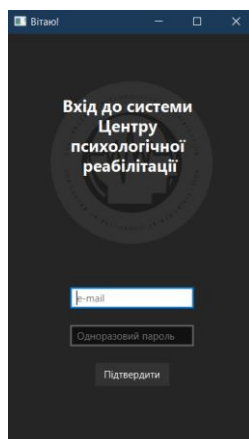


Рисунок 4.2 - Вхід до системи

Якщо обліковий запис користувача є в базі даних, але це його перший вхід - йому буде запропоновано відсканувати QR-код для отримання паролю (рисунок 4.3).



Рисунок 4.3 - QR-код для автентифікації

Після сканування коду в користувача на телефоні будуть генеруватися одноразові паролі для входу в систему. Кожен код дійсний лише 30 секунд, тому навіть якщо стороння людина його побачить - вона не зможе використати його повторно.

4.3 Головні функції застосунку

Головне меню складається з 4 головних функцій (рисунок 4.4):

- додавання медичної картки;
- перегляд списку пацієнтів;
- додавання запису про проведену сесію;
- додавання нового співробітника (доступно користувачам з правами адміністратора).

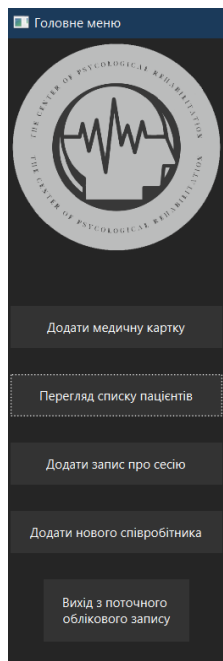


Рисунок 4.4 - Головне меню програми

Функція “Додати медичну картку” (рисунок 4.5) дозволяє користувачеві внести дані про нового пацієнта для подальшого ведення обліку його прогресу в лікуванні.

Рисунок 4.5 - Вікно “Додати медичну картку”

Функція “Перегляд списку пацієнтів” (рисунок 4.6) дозволяє користувачеві переглядати перелік поточних пацієнтів та їх дані. Також він може отримати список сесій проведених з пацієнтом. Для зручності реалізовано пошук запису по номеру телефону та елементу ПІБ.

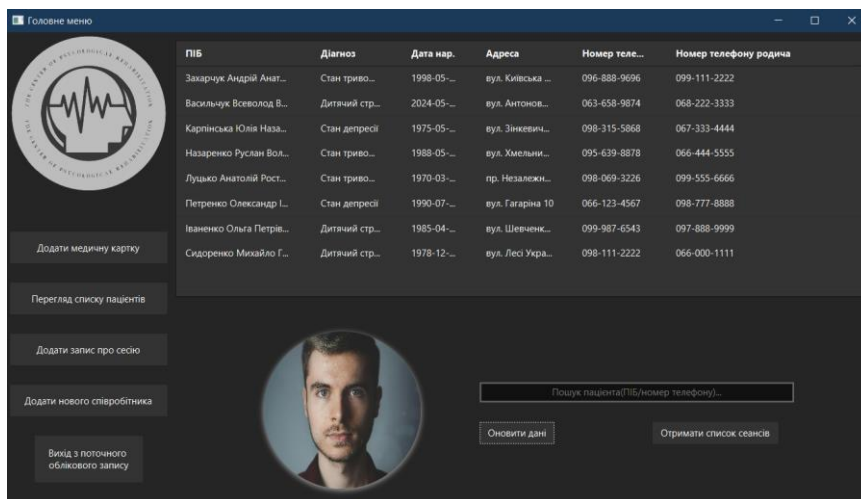


Рисунок 4.6 - Вікно “Перегляд списку пацієнтів”

Обравши необхідний запис та натиснувши на кнопку “Отримати список сеансів” відкриється нове вікно (рисунок 4.7), в якому користувач може переглянути деталі про всі раніше проведені з пацієнтом сеанси.

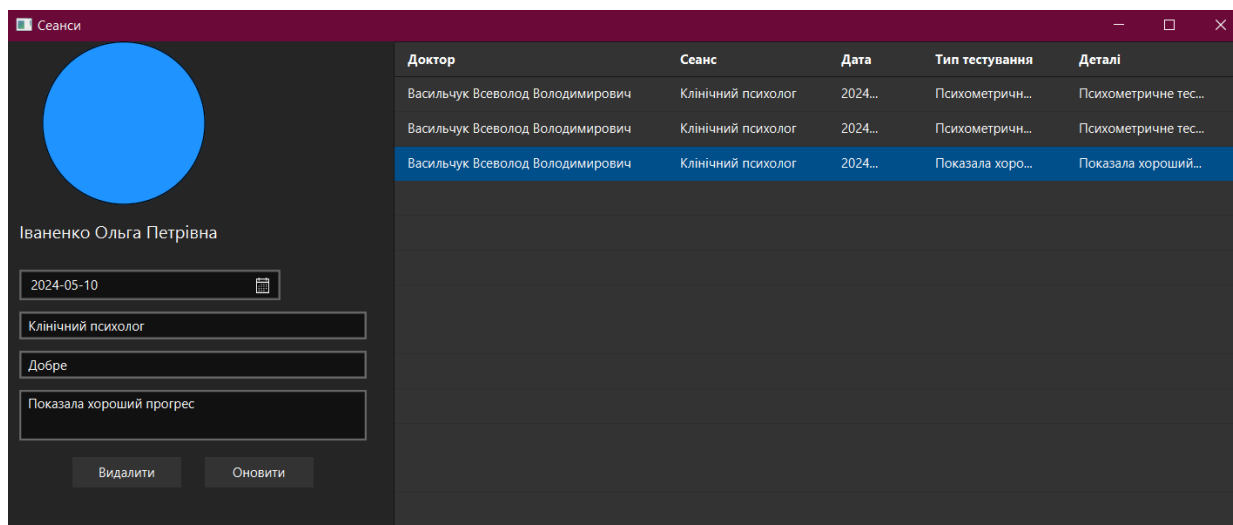


Рисунок 4.7 - Вікно “Перегляд списку сеансів пацієнта”

Користувач може вносити записи про проведені ним сесії у вікні “Додати запис про сесію” (рисунк 4.8). Відповідно, записи про сесії можуть створюватись відносно пацієнтів, яких лікує даний користувач.

Головне меню

Вибір пацієнта:

Вид тестування/опитування:

Оцінка роботи та прогресу пацієнта:

Дата проведення сесії:

Введіть деталі щодо сесії (замініть пустим, якщо немає необхідності заповнювати):

Додати медичну картку

Перегляд списку пацієнтів

Додати запис про сесію

Додати нового співробітника

Вихід з поточного облікового запису

Підтвердити

Рисунок 4.8 - Вікно “Додати запис про сесію”

Для користувачів з розширеними правами доступна функція “Додати нового співробітника” (рисунк 4.9). Після створення запису в цьому вікні буде створено новий запис у базі даних, за допомогою якого новий користувач при спробі заходу в застосунок за допомогою своєї електронної адреси зможе отримувати одноразові паролі.

Головне меню

ПІБ	Спеціаліза...	Дата нар.	Адреса	Номер теле...	email
Васильчук Всеволод В...	Клінічний п...	2024-05-...	вул. Антонов...	063-658-9874	admin
Карпінська Юлія Наза...	Дитячий пс...	1975-05-...	вул. Зінкевич...	098-315-5868	yulia0205k@gmail.com
Назаренко Руслан Вол...	Психолог-пс...	1988-05-...	вул. Хмельни...	095-639-8878	mightynaz@protonmail.com
Лущко Анатолій Рост...	Психотерап...	1970-03-...	пр. Незалежн...	098-069-3226	luzkonali@ukr.net
Артеменко Анатолій Р...	Психотерап...	1970-03-...	пр. Незалежн...	098-069-3226	test@ukr.net

Прізвище:

Ім'я:

По-батькові:

Дата народження:

Email:

Адреса:

Номер телефону:

Стать:

Спеціалізація:

☐ Права адміністратора?

Додати медичну картку

Перегляд списку пацієнтів

Додати запис про сесію

Додати нового співробітника

Вихід з поточного облікового запису

Обрати фото

Створити

Оновити

Видалити

Рисунок 4.9 - Вікно “Додати нового користувача”

Для завершення роботи в програмі користувачеві необхідно внести всі зміни та просто закрити програму. Також є змога одразу ввійти іншому користувачеві, за що відповідає кнопка на панелі меню.

4.4 Візуалізація шифрування

Для демонстрації шифрування введених даних було реалізовано вікно, яке за бажанням можна відкрити при запуску програми (рисунок 4.10). Весь алгоритм шифрування реалізований за формулами 1-5. Користувач може ввести необхідний текст та довжину ключа та покроково шифрувати цей текст. У відповідних полях показано елементи публічного ключа та приватний ключ. Всі дані можна скопіювати та зберегти для подальшого розшифрування, яке також можна інтерактивно зробити в даному вікні.

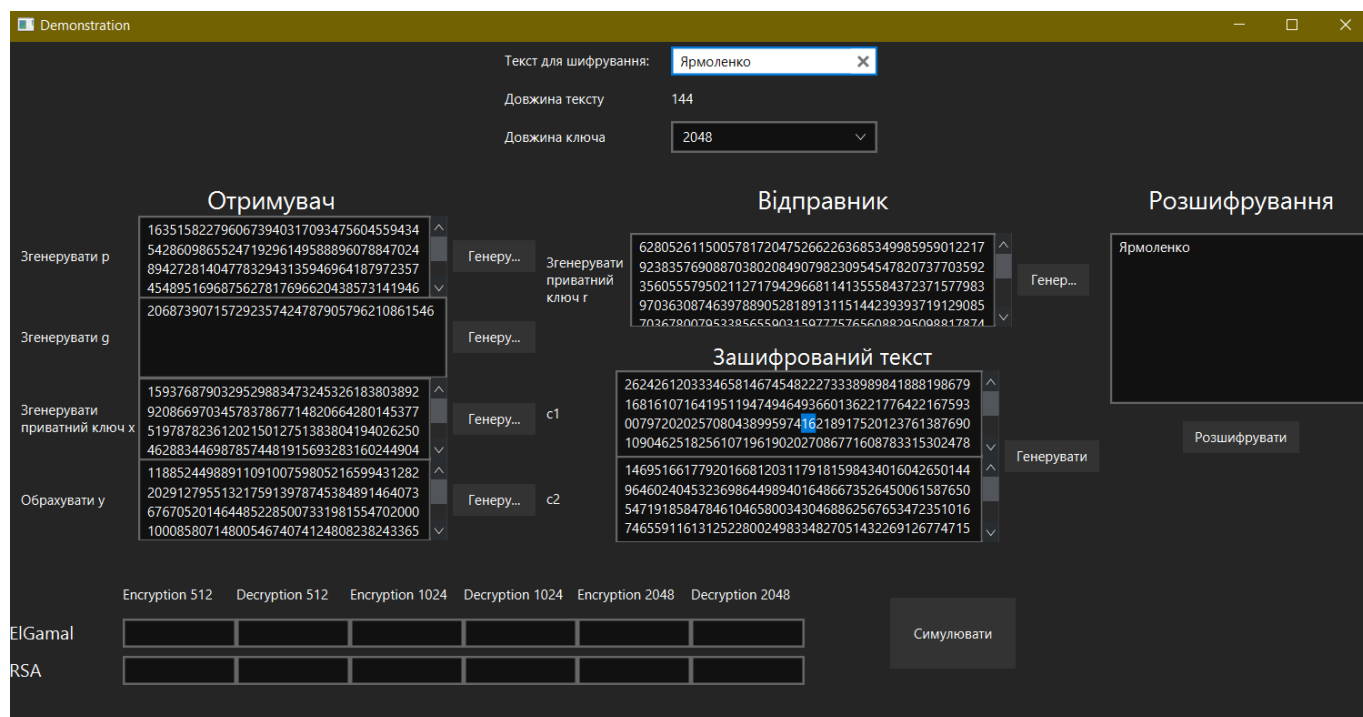


Рисунок 4.10 - Покрокова демонстрація шифрування

В цьому ж вікні, в нижній його частині, можна провести порівняння алгоритмів Ель-Гамалю та RSA в плані швидкодії. Проте, для репрезентативності даних необхідно провести значно більше, ніж кілька симуляцій, оскільки при виконанні симуляції апаратне забезпечення може спричиняти затримки. Також варто зазначити, що на різних системах результат з великою ймовірністю відрізнятиметься, тому що швидкість виконання алгоритмів напряду залежить від потужності комплектуючих машини, на які виконується програма.

ВИСНОВОК

В ході виконання дипломного проекту проведено роботу над виконанням низки завдань, спрямованих на розробку та впровадження системи захисту даних на базі алгоритму шифрування Ель-Гамалія. Перелік основних виконаних завдань включає в себе:

- досліджено вимоги до системи захисту даних з врахуванням специфічних потреб замовника;
- запропоновано використання архітектурного шаблону, на основі якого розроблено систему захисту даних, що відповідає вимогам щодо конфіденційності, цілісності, доступності та безпеки;
- впроваджено механізми автентифікації та авторизації користувачів, а також контролю доступу до даних;
- досліджено швидкодію обраного алгоритму та проведено порівняння з найбільш поширеним у сфері аналогом;
- проведено тестування програмного забезпечення системи, в результаті якого виправлено помилки та нестабільності в роботі системи.

Робота включала ознайомлення з спеціалізованим програмним забезпеченням, дослідження технологій відповідної сфери, та опрацювання літератури за темою дипломної роботи. В результаті виконання роботи розроблено систему обліку психологічно-реабілітаційного центру та впроваджено в неї систему захисту даних на основі алгоритму шифрування Ель-Гамалія.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про захист персональних даних. *Офіційний вебпортал парламенту України*. URL: <https://zakon.rada.gov.ua/laws/show/2297-17#Text> (дата звернення: 19.04.2024).
2. Schneier B. Applied cryptography: protocols, algorithms and source code in C. 20th ed. Wiley. John Wiley & Sons, LTD, 2015. 784 p.
3. ElGamal T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE trans. inf. theory*. 31st ed. 1985. P. 469-472.
4. Pérez A. Network security. Wiley & Sons, Incorporated, John, 2014. 308 p.
5. Cybersecurity essentials / C. J. Brooks et al. Wiley & Sons, Incorporated, John, 2018. 784 p.
6. Vanstone S. A., Menezes A. J., Oorschot P. C. v. Handbook of applied cryptography. Taylor & Francis Group, 2018. 810 p.
7. Aderemi E., Olugbara O. Computational complexity of rsa and elgamal cryptographic algorithms on video data. *Journal of theoretical and applied information technology*. 2020. Vol. 100, no. 15.
8. What is Java technology and why do I need it?. URL: https://www.java.com/en/download/help/whatis_java.html (дата звернення: 21.04.2024).
9. Gourley D., Totty B. HTTP: the definitive guide. O'Reilly Media, Inc., 2002. 656 p.
10. DB-Engines ranking. *DB-Engines*. URL: <https://db-engines.com/en/ranking> (дата звернення: 23.05.2024).
11. MySQL advantages and disadvantages - blue claw database developer resource. *Blue Claw Database Developer Resource*. URL: <https://blueclawdb.com/mysql/advantages-disadvantages-mysql/> (дата звернення: 25.05.2024).
12. JDBC tutorial | what is java database connectivity(jdbc) - javatpoint. *www.javatpoint.com*. URL: <https://www.javatpoint.com/java-jdbc> (дата звернення: 26.04.2024).
13. Wadi H. Step by step java GUI with JDBC and mysql: practical approach to build database desktop application with project based examples. Independently Published, 2020.

14. JavaFX theme jmetro. java theme inspired on fluent design | pixel duke. *Pixel Duke*. URL: <https://www.pixelduke.com/java-javafx-theme-jmetro/> (date of access: 30.04.2024).
15. Richards M., Ford N. Fundamentals of software architecture: an engineering approach. O'Reilly Media, 2020. 432 p.
16. Фрімен Е., Робсон Е. Патерни проєктування. Head First. Легкий для сприйняття довідник. Харків : Фабула, 2020. 688 с.
17. Getting started with javafx: using FXML to create a user interface | javafx 2 tutorials and documentation. *Oracle*. URL: https://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm (дата звернення: 01.05.2024).
18. Мартін Р. С. Чистий код. Створення і рефакторинг за допомогою Agile. Харків : Фабула, 2019. 416 с.
19. Manelli L., Zambon G. Beginning jakarta EE web development. Berkeley, CA : Apress, 2020.
20. RFC 6238: tOTP: Time-Based One-Time Password Algorithm. *IETF Datatracker*. URL: <https://datatracker.ietf.org/doc/html/rfc6238> (дата звернення: 03.05.2024).
21. GitHub - samdjstevens/java-totp: a java library for implementing time-based one time passwords for multi-factor authentication. *GitHub*. URL: <https://github.com/samdjstevens/java-totp> (дата звернення: 04.05.2024).
22. Loshin P. Simple steps to data encryption: a practical guide to secure computing. Elsevier Science & Technology Books, 2013. 86 p.
23. O'Reilly A., Trappe W., Shmatikov V. Re RSA and ElGamal secure for hardware?. *In proceedings of the 2019 on great lakes symposium on VLSI*. P. 201-221.
24. Stallings W. Cryptography and network security: principles and practice. Pearson Education, Limited, 2010. 744 p.
25. Stinson D. R., Paterson M. Cryptography: theory and practice. Taylor & Francis Group, 2018. 580 p.

ДОДАТОК А

ТЕКСТ ПРОГРАМНОГО МОДУЛЯ

«ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗАХИСТУ ДАНИХ НА ОСНОВІ АЛГОРИМУ ШИФРУВАННЯ ЕЛЬ-ГАМАЛЯ»

УКР.НТУУ “КПІ ім. Ігоря Сікорського”_ІАТЕ_ЦТЕ_ТР-02

Аркушів 11

Київ-2024

```
//Encryption class
```

```
package org.rehab.encryption;

import org.rehab.rehabcenter.AuthorizationController;
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.Random;

public class Encryption {

    private static final int BIT_LENGTH = 1024;
    private static final SecureRandom random = new SecureRandom();

    public static KeyPair generateKeys() {
        BigInteger p = generateP();//випадкове велике просте число
        BigInteger g = generateG(p); //первісний корінь
        BigInteger x = generateX(p); //приватний ключ  $0 < x < p-1$ 
        BigInteger y = g.modPow(x, p); //  $y = g^x \bmod p$ 
        PublicKey publicKey = new PublicKey(p, g, y);
        PrivateKey privateKey = new PrivateKey(x, publicKey);
        return new KeyPair(publicKey, privateKey);
    }

    public static BigInteger generateP(){
        return BigInteger.probablePrime(BIT_LENGTH, random);
    }

    public static BigInteger generateG(BigInteger p){
        return new BigInteger(128, random).mod(p.subtract(BigInteger.ONE));
    }
}
```

```

public static BigInteger generateX(BigInteger p){
    return new BigInteger(BIT_LENGTH,
random).mod(p.subtract(BigInteger.ONE));
}

public static BigInteger randNum(BigInteger N, Random prg) {
    return new BigInteger(N.bitLength() + 100, prg).mod(N);
}

public static BigInteger[] encrypt(String message, PublicKey publicKey) {
    BigInteger r = AuthorizationController.sessionKey;    // сесійний ключ
    BigInteger m = new BigInteger(message.getBytes());
    BigInteger c1 = publicKey.g.modPow(r, publicKey.p);
    BigInteger c2 = m.multiply(publicKey.y.modPow(r, publicKey.p));
    return new BigInteger[]{c1, c2};
}

public static String decrypt(BigInteger[] encrypted, PrivateKey privateKey) {
    BigInteger a = encrypted[0];
    BigInteger b = encrypted[1];
    BigInteger s = a.modPow(privateKey.x(), privateKey.publicKey().p()); //  $s = a^x$ 
    % p
    BigInteger plaintext =
b.multiply(s.modInverse(privateKey.publicKey().p())).mod(privateKey.publicKey().p()); //
plaintext =  $b * (s^{-1} \% p) \% p$ 
    byte[] plaintextBytes = plaintext.toByteArray();
    return new String(plaintextBytes);
}

record KeyPair(PublicKey publicKey, PrivateKey privateKey) {}

```

```

    public record PublicKey(BigInteger p, BigInteger g, BigInteger y) {}
    public record PrivateKey(BigInteger x, PublicKey publicKey) {}
}

```

```
//Servlets example
```

```

package servlets.users;
import database.Database;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.*;
import servlets.Utils;
import java.io.IOException;
import java.util.HashMap;

```

```
@WebServlet(name = "InsertClient", value = "/InsertClient")
```

```
public class InsertClient extends HttpServlet {
```

```
    @Override
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException {
```

```
    }
```

```
    @Override
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException {
```

```
        HashMap<String, String> map = Utils.getReceivedParametersDecrypt(request);
```

```
        System.out.println(map);
```

```
        Database.insertClientCifred(map);
```

```
    }
```

```
}
```

```
public static int insertClientCifred(HashMap<String, String> parameters){
```

```

Connection connection = null;
PreparedStatement prepare = null;
try {
    connection = customConnect("db.url");
    String query = "INSERT INTO clientsCifred (person_id,
emergency_contact_name, emergency_contact_phone, diagnosis, session_key) " +
        "VALUES (?, ?, ?, ?, ?);";
    prepare = connection.prepareStatement(query,
prepare.RETURN_GENERATED_KEYS);
    prepare.setInt(1, Integer.parseInt(parameters.getDefault("person_id", "")));
    prepare.setString(2, parameters.getDefault("emergency_contact_name",
    ""));
    prepare.setString(3, parameters.getDefault("emergency_contact_phone",
    ""));
    prepare.setString(4, parameters.getDefault("diagnosis", ""));
    prepare.setString(5, parameters.getDefault("session_key", ""));
    prepare.executeUpdate();
    connection.close();
} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        if (prepare != null)
            prepare.close();
        if (connection != null)
            connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}

```

```
return -1;}
```

```
//GUI Interface example
```

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.DatePicker?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.control.TextArea?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.shape.Circle?>
<?import javafx.scene.text.Font?>

<HBox fx:id="mainBox" alignment="CENTER" prefHeight="450.0" prefWidth="900.0"
xmlns="http://javafx.com/javafx/20.0.1"          xmlns:fx="http://javafx.com/fxml/1"
fx:controller="org.rehab.rehabcenter.SessionsWindow">
  <children>
    <VBox HBox.hgrow="ALWAYS">
      <children>
        <VBox prefHeight="200.0" prefWidth="100.0">
          <children>
            <VBox prefHeight="200.0" prefWidth="100.0">
              <children>
```



```

</DatePicker>
<TextField fx:id="sessionsTestTypeField" promptText="Тип тестування...">
  <font>
    <Font size="14.0" />
  </font>
  <VBox.margin>
    <Insets bottom="10.0" />
  </VBox.margin>
</TextField>
<TextField fx:id="sessionsGradeField" promptText="Оцінка тестування...">
  <font>
    <Font size="14.0" />
  </font>
  <VBox.margin>
    <Insets bottom="10.0" />
  </VBox.margin>
</TextField>
<TextArea fx:id="sessionsTextArea" prefHeight="90.0" prefWidth="300.0"
promptText="Детальніше...">
  <font>
    <Font size="14.0" />
  </font>
</TextArea>
<HBox alignment="CENTER">
  <children>
    <Button fx:id="sessionDeleteButton" mnemonicParsing="false"
onAction="#makeDelete" prefHeight="30.0" prefWidth="100.0" text="Видалити">
      <HBox.margin>
        <Insets right="10.0" />
      </HBox.margin>

```

```

        </Button>
        <Button      fx:id="sessionUpdateButton"      mnemonicParsing="false"
onAction="#makeUpdate" prefHeight="30.0" prefWidth="100.0" text="Оновити">
            <HBox.margin>
                <Insets left="10.0" />
            </HBox.margin>
        </Button>
    </children>
    <VBox.margin>
        <Insets top="15.0" />
    </VBox.margin>
</HBox>
</children>
</VBox>
</children>
<HBox.margin>
    <Insets left="10.0" />
</HBox.margin>
</VBox>
<TableView      fx:id="sessionsTable"      onMouseClicked="#selectedTableItem"
HBox.hgrow="ALWAYS">
    <columns>
        <TableColumn  fx:id="sessionsDoctorCol"  prefWidth="123.20001375675201"
text="Доктор" />
        <TableColumn fx:id="sessionsSpezializationCol" prefWidth="84.79998016357422"
text="Сеанс" />
        <TableColumn  fx:id="sessionsDateCol"    prefWidth="75.20004272460938"
text="Дата" />
        <TableColumn  fx:id="sessionsGradeCol"   prefWidth="67.199951171875"
text="Оцінка" />
    </columns>

```

```

        <TableColumn fx:id="sessionsTypeCol" minWidth="0.0"
prefWidth="131.2000732421875" text="Тип тестування" />
        <TableColumn fx:id="sessionsDescriptionCol" prefWidth="153.5999633789063"
text="Деталі" />
    </columns>
    <HBox.margin>
        <Insets left="25.0" />
    </HBox.margin>
</TableView>
</children>
</HBox>

//build plugin
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.10.1</version>
            <configuration>
                <source>17</source>
                <target>17</target>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-maven-plugin</artifactId>
            <version>0.0.8</version>
            <executions>
                <execution>

```

```
<id>default-cli</id>
```

```
<configuration>
```

```
<mainClass>org.rehab.rehabcenter/org.rehab.rehabcenter.HelloApplication</mainClass>
```

```
  <launcher>app</launcher>
```

```
  <jlinkZipName>app</jlinkZipName>
```

```
  <jlinkImageName>app</jlinkImageName>
```

```
  <noManPages>true</noManPages>
```

```
  <stripDebug>true</stripDebug>
```

```
  <noHeaderFiles>true</noHeaderFiles>
```

```
</configuration>
```

```
</execution>
```

```
</executions>
```

```
</plugin>
```

```
</plugins>
```

```
</build>
```