«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В.Коваль
(підпис) (ініціали, прізвище)
“___”_____2019 р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки
6.050103 “Програмна інженерія’’

на тему: Система попередньої обробки та шифрування даних
Виконав: студент 4 курсу, групи ТВ-51

Шарацький Олександ Сергійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник к.т.н., доц. Крячок Олександр Степанович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент к.т.н. Реуцький М.О. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050103 “Програмна інженерія’’

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019 р.

ЗАВДАННЯ
на дипломну роботу студенту

Шарацькому Олександру Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи “Система попередньої обробки та шифрування даних”

керівник роботи к.т.н., доц. Крячок Олександр Степанович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”___”_____ 201__ р. № _____

2. Строк подання студентом роботи ___ _____ 201__ р.

3. Вихідні дані до роботи бінарний файл бази даних, в якому зберігаються результати шифрування даних

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі алгоритми шифрування, розробити модуль шифрування на базі обраного алгоритму, спроектувати архітектуру системи та бази даних, розробити веб-сервіс шифрування-дешифрування, розробити інтерфейс користувача.

5. Перелік ілюстраційного матеріалу (з точним зазначенням обов’язкових креслень) 1. Мета та завдання роботи 2. Огляд існуючих рішень 3. Функції керуючого модуля системи шифрування-дешифрування 4. Функції веб-сервісу шифрування-дешифрування 5. Формули розрахунку 6. Використані програмні

засоби 7. Висновки

Дата видачі завдання ”___”_____ 201__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі		
2.	Розробка архітектури та загальної структури системи		
3.	Розробка структур окремих підсистем		
4.	Підготовка матеріалів		
5.	Програмна реалізація системи		
6.	Захист програмного продукту		
7.	Оформлення пояснювальної записки		
8.	Передзахист		
9.	Захист		

Студент _____ Шарацький О. С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Крячок О. С.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Метою роботи була розробка системи попередньої обробки та шифрування даних. Для кожного користувача система генерує ключі та зберігає в зашифрованому вигляді. Ключі, якими зашифровані ключі користувачів в системі не зберігаються, натомість вони зберігаються на носії користувача, і при вході в систему остання потребує вказати, де знаходиться файл з ключами.

Система шифрує текст, що передається, обраним методом шифрування, а інші дані, такі як зображення, розбиваються на частини, деякі з яких шифруються.

Користувацький додаток представляє собою систему зберігання записів пацієнтів медичного закладу та надає останнім зручний інтерфейс для перегляду, а працівникам простий інтерфейс для внесення та редагування записів.

Записка містить 54 сторінки, 13 рисунків, 4 формули та 25 посилань.

ABSTRACT

The purpose of the work was to develop a system of pre-processing and data encryption. For each user, the system generates keys and stores in encrypted form. Keys which were used to encrypt user keys are not stored in the system, instead they are stored on the user's carrier, and when logging in, the system requires specifying where the key file is located.

The system encrypts the transmitted text by the chosen encryption method, and other data, such as images, are split into parts, some of which being encrypted.

The user application is a system for storing medical records for patients and provides them user-friendly interface for viewing, and employees have a simple interface for entering and editing records.

The note contains 54 pages, 13 figures, 4 formulas and 25 references.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних;

ГВЧ – генератор випадкових чисел;

СКБД – система керування базами даних;

API (англ. Application Programming Interface) – прикладний програмний інтерфейс;

MVC (англ. model-view-controller) – схема поділу структури програмного продукту на три окремих компоненти: модель, представлення і контролер, які відповідають за користувацький інтерфейс, структуру даних і керуючу логіку

ВСТУП

Ми живемо в інформаційно-орієнтованому світі. Як суспільство, ми все більше покладаємося на створення та споживання даних, які повинні бути доступні там і тоді, коли це необхідно. Дані та пов'язані з ними інформаційні послуги включені або надаються через послуги інформаційних технологій, що об'єднують програми, засоби, мережі, сервери, апаратні засоби зберігання та програмні ресурси.

Внаслідок такої зростаючої залежності від інформації, як для домашнього, так і для особистого користування, а також для ділових та професійних потреб, генерується, обробляється, переміщується, зберігається та зберігається в декількох копіях більше даних протягом більшого періоду часу.

Так як кількість створених і збережених даних продовжує зростати з безпрецедентними показниками, виникає важливе питання захисту даних. Значна частина стратегії захисту даних забезпечує швидке відновлення даних після пошкодження або втрати. Захист даних від компрометації та забезпечення конфіденційності даних є іншими ключовими компонентами захисту даних.

Якщо персональні дані стають відомі третім особам, це може призвести до втрати компанією репутації, а також штрафи, і тому важливо дотримуватися правил захисту персональних даних.

У зв’язку з проведенням реорганізації медичних структур виникло завдання зберігання персональних даних пацієнтів в електронному вигляді. Такі дані потребують захисту. Було запропоновано систему шифруванням та обробки даних з наступною передачею в захищеному вигляді.

Для вирішення цієї проблеми розроблена система захисту даних, які передаються та зберігаються.

Рішення представляє собою Web-додаток та пропонує зручний інтерфейс користувача, який доступний у звичайному браузері.

Для даної системи пропонується використовувати асиметричний алгоритм RSA через його простоту реалізації та надійність.

Записка містить 4 розділи.

У першому розділі описується постановка задачі створення системи обробки та шифрування даних використовуючи алгоритм шифрування RSA.

У другому розділі розглянуто проблему розробки системи, проводиться огляд існуючих рішень шифрування-дешифрування, передачі та зберігання даних.

У третьому розділі описані засоби реалізації програмної системи.

У четвертому розділі описується програмна реалізація системи обробки та шифрування даних та описана методика роботи користувача з програмною системою.

ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ СИСТЕМИ ШИФРУВАННЯ

Через велике різноманіття користувачів і видів ПЗ існує безліч різних стилів користувальницьких інтерфейсів, і при їх розробці можуть використовуватися різні принципи і підходи. Однак наступних найважливіших принципів варто дотримуватись завжди:

- користувальницький інтерфейс повинен базуватися на термінах і поняттях, знайомих користувачу;
- користувальницький інтерфейс повинний бути однаковим;
- користувальницький інтерфейс повинен дозволяти користувачу виправляти власні помилки;
- користувальницький інтерфейс повинний дозволяти одержання користувачем довідкової інформації: як по запиту самого користувача, так і генерованої ПЗ.

Графічний користувальницький інтерфейс надає користувачу можливості:

- звертатися до ПЗ шляхом вибору на екрані бажаного графічного чи текстового об'єкта,
- одержувати від ПЗ велику кількість інформації на екрані у вигляді графічних і текстових об'єктів,
- здійснювати прямі маніпуляції з графічними і текстовими об'єктами, які представлені на екрані.

Графічний користувальницький інтерфейс дозволяє:

- розміщувати на екрані практично безліч різних вікон, у кожне з яких можна виводити інформацію незалежно;
- використовувати графічні об'єкти, названі піктограмами (чи іконами), для позначення різних інформаційних процесів чи об'єктів;

- використовувати екранний вказівник для вибору об'єктів (чи їх елементів), розміщених на екрані; екранний вказівник керується (переміщається) за допомогою клавіатури чи миші.

Перевагою будь-якого графічного користувальницького інтерфейсу є можливість створення зручної і зрозумілої користувачу моделі взаємодії з ПЗ без необхідності вивчення якої-небудь спеціальної мови. Однак його розробка вимагає великих трудовитрат, порівнянних із трудовитратами по створення самого ПЗ. Крім того, виникає серйозна проблема з переносом ПЗ на інші операційні системи, тому що графічний інтерфейс повністю залежить від можливостей графічної користувальницької платформи, наданих операційною системою для його створення [1].

Метою роботи є створення системи в якій працівники медичного закладу можуть вносити записи про пацієнтів до бази, а пацієнти можуть переглядати лише власні записи. Усі дані передаються зашифрованими та зберігаються в зашифрованому вигляді.

Під час розробки виникли задачі розробки графічного інтерфейсу користувача, проектування бази даних, вибір алгоритму шифрування та передачі ключів, спосіб збереження даних в базі.

Призначенням даної системи є надання зручного інтерфейсу для користувачів та адміністраторів системи для перегляду та редагування записів. Модуль містить ряд класів, кожен з яких відповідає за певний сервіс, наприклад взаємодія з віддаленою базою даних, аунтентифікація/регістрація користувачів системи.

1.1. Потенційні користувачі

Web-додаток орієнтовано в першу чергу на пацієнтів медичних закладів, які хочуть мати доступ до своїх даних будь де і будь коли у зручній формі, та частини працівників медичних закладів, які будуть вносити та редагувати дані в системі.

Передбачена можливість:

- аутентифікація вже існуючих користувачів;
- створення нових користувачів за допомогою електронної адреси та паролю;
- збереження у віддаленій бд інформації про дії користувача;
- можливість зміни вже існуючого та його видалення з бд;
- створення облікового запису пацієнта, його редагування;
- створення записів пацієнта та їх редагування

1.2 Клієнт-серверна архітектура

Клієнт та сервер взаємодіють за допомогою HTTP-запитів [2]. У випадках, коли дані для передачі є специфічними об’єктами, вони конвертуються у рядковий формат JSON. Використання зумовлено читаємістю розробником та легкістю серіалізації та десеріалізації складних структур.

Клієнт та сервер створені за допомогою різних технологій, але це не впливає на основні принципи їх взаємодії.

АНАЛІЗ ПРОБЛЕМИ РОЗРОБКИ СИСТЕМИ ШИФРУВАННЯ

Проблеми безпеки створюють серйозні загрози для будь-якої системи, тому важливо знати її вразливості. Бази даних є дуже привабливими цілями для хакерів, оскільки містять цінну та конфіденційну інформацію. Вона може варіюватися від фінансової або інтелектуальної власності до корпоративних і персональних даних користувачів. Кіберзлочинці можуть отримувати прибуток, проникаючи до серверів компаній і пошкоджуючи бази даних у процесі. Таким чином, тестування безпеки бази даних є обов'язковим.

Розглянемо найпоширеніші з проблем.

Вразливість до генерації підроблених даних

Кіберзлочинці можуть підробляти або генерувати дані і відправляти їх до системи, де вони будуть зберігатися разом з дійсними. Наприклад, якщо ваша виробнича компанія використовує дані датчиків для виявлення несправних виробничих процесів, зловмисники можуть проникнути у вашу систему і змусити датчики показувати підроблені результати, скажімо, неправильні температури. Таким чином, ви можете не помітити тривожні тенденції і впустити можливість вирішити проблеми до того, як буде завдано серйозної шкоди. Від таких вразливостей можна захиститись шляхом застосування підходу виявлення шахрайства.

Проблеми криптографічного захисту

Хоча шифрування є відомим засобом захисту конфіденційної інформації, воно є в списку проблем безпеки даних. Незважаючи на можливість та важливість шифрування інформації, цей захід безпеки часто ігнорується. Персональні дані, як правило, зберігаються в хмарі без будь-якого захисту шифруванням. Причина такої безтурботності проста: постійні шифрування та дешифрування навіть незначних фрагментів даних сповільнюють процес, що тягне за собою втрату вагомої переваги – швидкості.

Можливість добування важливої інформації

В системах зазвичай використовується захист на її межах. Це означає, що всі «пункти в'їзду та виїзду» захищені. Але те, що роблять IT-спеціалісти у вашій системі, залишається загадкою.

Така відсутність контролю у вашому рішенні з даних може дозволити вашим корумпованим IT-спеціалістам або злому бізнесу конкуренту добувати незахищені дані та продавати їх задля власної вигоди. Ваша компанія, у свою чергу, може зазнати величезних втрат, якщо така інформація пов'язана з запуском нового продукту / послуги, фінансовими операціями компанії або особистою інформацією користувачів.

Так, дані можна краще захистити, додавши додаткові периметри. Крім того, безпека вашої системи може покращитися з анонімізацією. Якщо хтось отримує персональні дані ваших користувачів з відсутніми іменами, адресами та телефонами, вони практично не можуть завдати шкоди.

Немає перевірки безпеки перед розгортанням

Однією з найпоширеніших причин слабкості баз даних є недбалість на стадії розгортання. Незважаючи на те, що функціональне тестування проводиться для забезпечення найвищої продуктивності, цей тип тесту не може показати вам, чи робить база даних те, що вона не повинна. Таким чином, важливо перевірити безпеку веб-сайтів різними типами тестів перед повним розгортанням.

Викрадені резервні копії бази даних

Існує два види загроз для ваших баз даних: зовнішній і внутрішній. Є випадки, коли компанії борються з внутрішніми загрозами навіть більше, ніж із зовнішніми. Власники бізнесу ніколи не можуть бути на 100% впевнені в лояльності своїх працівників, незалежно від того, яке програмне забезпечення для комп'ютерної безпеки вони використовують, і наскільки відповідальними вони є. Кожен, хто має доступ до конфіденційних даних, може вкрасти його та продати його третім сторонам для отримання прибутку. Однак існує можливість усунути ризик: шифрувати архіви баз даних, застосовувати суворі стандарти безпеки, застосовувати штрафи у разі порушень, використовувати програмне забезпечення кібербезпеки та

постійно підвищувати обізнаність ваших команд через корпоративні зустрічі та особисті консультації.

Безмежний доступ до адміністрування = поганий захист даних

Розумне поділ обов'язків між адміністратором і користувачем гарантує обмежений доступ тільки досвідченим командам. Таким чином, користувачі, які не беруть участь у процесі адміністрування баз даних, відчуватимуть більше труднощів, якщо вони намагатимуться вкрасти будь-які дані. Якщо ви можете обмежити кількість облікових записів користувачів, це ще краще, оскільки хакерам доведеться зіткнутися з більшими проблемами в отриманні контролю над базою даних. Цей випадок може бути застосований до будь-якого виду бізнесу, але зазвичай це відбувається у фінансовій галузі. Таким чином, добре не тільки дбати про те, хто має доступ до конфіденційних даних, але й виконувати тестування банківського програмного забезпечення, перш ніж випускати його.

Неадекватне управління ключами

Це добре, якщо ви шифруєте конфіденційні дані, але також важливо, щоб ви звернули увагу на того, хто саме має доступ до ключів. Оскільки ключі часто зберігаються на чиємусь жорсткому диску, це, очевидно, легка ціль для того, хто хоче їх вкрасти. Якщо ви залишите такі важливі засоби безпеки програмного забезпечення незахищеними, знайте, що це робить вашу систему вразливою до атак.

Шифрування даних – це метод захисту, в якому інформація кодується і може бути відновлена або розшифрована лише користувачем з правильним ключем дешифрування. Зашифровані дані, також відомі як шифротекст, виглядають зашифрованими або нечитабельними для особи або суб'єкта, які здійснюють доступ без дозволу.

Шифрування даних використовується для запобігання доступу до конфіденційних даних зловмисниками або небажаними особами. Важлива лінія захисту в архітектурі кібербезпеки – шифрування – робить використання перехоплених даних, настільки складним, настільки можливо. Воно може бути застосовано до всіх видів потреб у захисті даних, починаючи від секретних документів уряду до операцій з особистими кредитними картками. Програмне

забезпечення для шифрування даних, також відоме як алгоритм шифрування або шифр, використовується для розробки схеми шифрування, яке теоретично можна обійти тільки з великими обсягами обчислювальної потужності.

Шифрування застосовується для зберігання важливої інформації в ненадійних джерелах та її передачі по незахищеним каналам зв'язку. Така передача даних представляє із себе два взаємно зворотних процесу:

Перед відправленням даних по лінії зв'язку або перед приміщенням на зберігання вони піддаються зашифровуванню.

Хоча дані все ще можуть бути перехоплені, вони будуть незрозумілими і тому не корисними для шпигунів або хакерів.

Всілякі пристрої в різних мережах шифрують повідомлення, якими обмінюються. Шифрування використовується не тільки для передачі в Інтернеті, а й для транзакцій банкоматів або мобільних телефонів. Алгоритми шифрування захищають всі передані дані.

Для відновлення вихідних даних із зашифрованих до них можна застосувати процедуру розшифрування.

Шифром називається пара алгоритмів, що реалізують кожне із зазначених перетворень. Ці алгоритми застосовуються до даних з використанням ключа. Ключі для шифрування та розшифрування можуть як відрізнятися, так і бути однаковими. Секретність другого з них робить дані недоступними для несанкціонованого читання, а таємність першого не дозволяє вносити неправдиві дані. У перших методах шифрування використовувалися однакові ключі, однак в 1976 році були відкриті алгоритми із застосуванням різних ключів. Збереження цих ключів в секретності і правильне їх поділ між адресатами є дуже важливим завданням з точки зору збереження конфіденційності інформації, що передається. Це завдання досліджується в теорії управління ключами.

На даний момент запропоновано величезну кількість методів шифрування. Головним чином ці методи діляться на симетричні та асиметричні, в залежності від структури використовуваних ключів. Крім того, методи шифрування можуть мати різну криптостійкість і по-різному обробляти вхідні дані – блокові шифри і потокові

шифри. Всіма цими методами, їх створенням і аналізом займається наука криптографія.

Це майже безсумнівно, але ключ є фундаментальною частиною захисту конфіденційності інформації, повідомлення або частини даних. Процес шифрування та дешифрування може бути ініційований лише за допомогою ключа.

Ключ шифрування / дешифрування можна порівняти з звичайним паролем – наприклад, з тим, що ви використовуєте для своєї електронної пошти. Ключ є невід'ємною частиною процесу кодування і декодування даних.

Ключі шифрування розроблені таким чином, щоб вони були абсолютно унікальні, використовуючи набір різних алгоритмів. Ключ шифрування використовується для кодування або декодування даних. Це означає, що ключ шифрування здатний змішувати дані в нечитабельні символи, і він може повернути ці символи назад до відкритого тексту.

Як правило, ключ є випадковим двійковим або фактичним паролем. Ключ «розповідає» алгоритму про те, які шаблони необхідно дотримуватися, щоб перетворити відкритий текст на зашифрований текст (і навпаки).

У зв'язку з тим, що алгоритми є загальнодоступними і доступні для будь-кого, якщо хакер отримає ключ шифрування, зашифровані дані легко розшифруються до відкритого тексту.

Симетричний ключ або секретний ключ використовує один і той самий ключ для кодування і декодування інформації (рисунок 2.1). Його найкраще використовувати при обміні невеликих об'ємів даних один з одним. Хоча симетричне шифрування набагато швидше ніж асиметричне, відправник повинен обмінятися ключем шифрування з одержувачем, перш ніж останній зможе розшифрувати повідомлення.

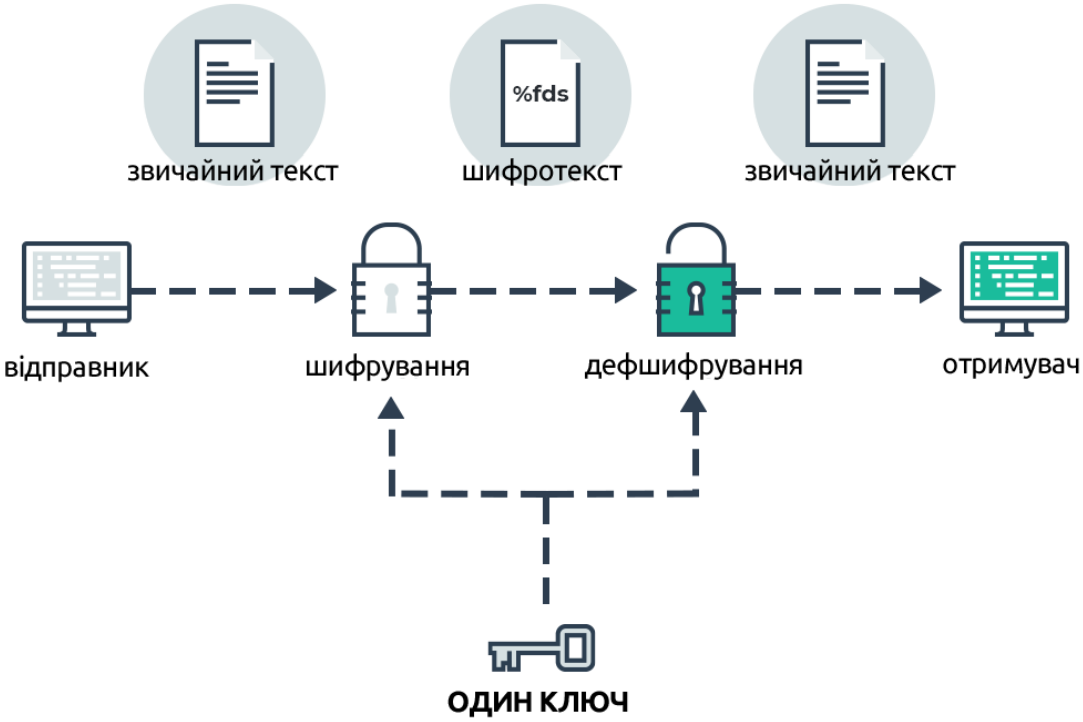

Рисунок 2.1 – Схема шифрування симетричним алгоритом

На відміну від симетричних алгоритмів, асиметричні алгоритми використовують пару двох ключів для виконання алгоритму – один приватний і один публічний (рисунок 2.2). Ключ шифрування є загальнодоступним і може використовуватися будь-ким для шифрування. Інший ключ зберігається в секреті і використовується для розшифрування.

Асиметричне шифрування встановлює аутентифікацію. Під час процесу аутентифікації функція відкритого ключа полягає в перевірці того, що повідомлення надіслано власником приватного ключа. Натомість, лише власник приватного ключа може розшифрувати повідомлення, зашифроване за допомогою публічного ключа.

Приватний ключ створюється на основі дуже складних математичних обчислень, які пов'язані з парою відкритих ключів. Простіше кажучи, якщо повідомлення або набір даних шифруються відкритим ключем, тільки його пара приватних ключів може розшифрувати його - і навпаки.

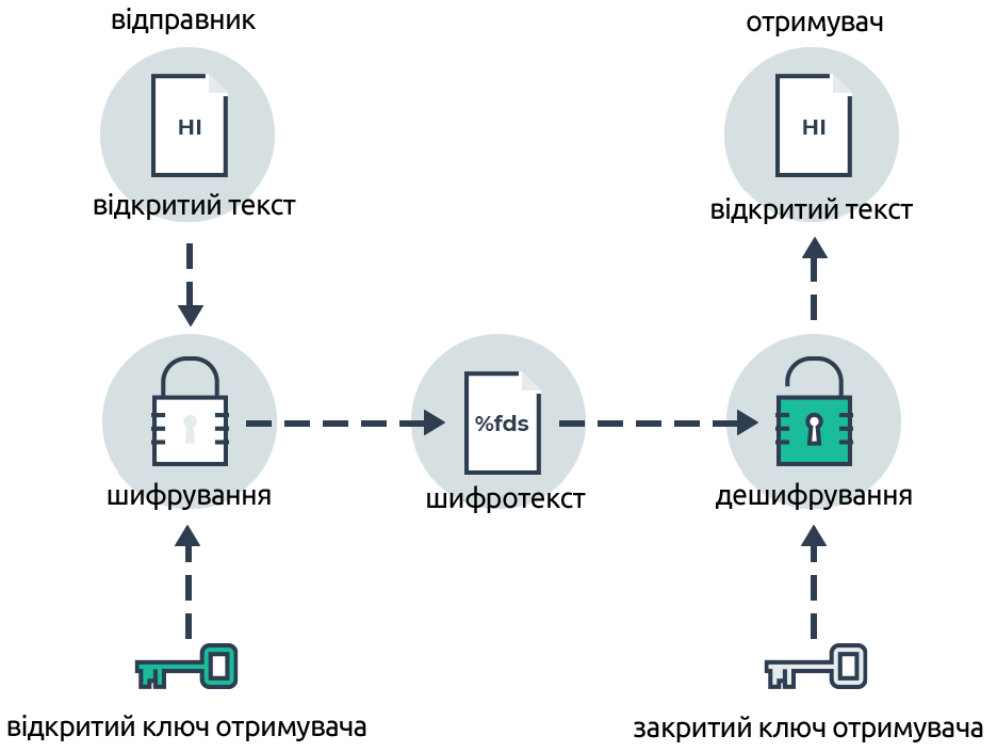

Рисунок 2.2 – Схема асиметричного алгоритму

Найбільша перевага використання асиметричних алгоритмів полягає в тому, що вам ніколи не потрібно надсилати або передавати ключ шифрування або пароль через небезпечний канал. Отже, це різко знижує можливість бути взламаним.

Відкритий ключ можна спільно використовувати з ким завгодно без шкоди для безпеки, оскільки будь-яка людина може шифрувати повідомлення за допомогою відкритого ключа отримувача. Але, зашифроване повідомлення може бути розшифровано у відкритий текст тільки власником приватного ключа.

Асиметричні алгоритми мають три недоліки:

- Потрібно завершити процес аутентифікації відкритим ключем кожного разу, коли надсилається повідомлення.
- Якщо ви втратите приватний ключ, неможливо розшифрувати зашифрований текст.
- Обробка асиметричного алгоритму набагато повільніше порівняно з симетричним алгоритмом через його математичну складність, і тому не підходить для обчислення великих обсягів даних.

Шифрування в основному використовувалося урядами і великими корпораціями до 1970-х років, але новаторське представлення книги «Нові напрямки в криптографії» Вітфілда Діффі та Мартіна Хеллмана змінило це в 1976 році.

Їх робота привела до впровадження алгоритму RSA на персональних комп'ютерах. Зрештою, шифрування стало широко впроваджуватися в веб-браузерах і серверах даних для захисту даних.

Алгоритм Rivest-Sharmir-Adleman (RSA) – це криптосистема шифрування з відкритим ключем, яка широко використовується для захисту конфіденційних даних, особливо коли вона надсилається через небезпечну мережу, подібну до Інтернету. Популярність алгоритму RSA походить від того, що публічні та приватні ключі можуть шифрувати повідомлення для забезпечення конфіденційності, цілісності, достовірності та безвідмовність електронних комунікацій та даних за допомогою цифрових підписів.

Найбільш простим методом атаки на шифрування є груба сила (анг. brute force), або перебір випадкових ключів, поки не знайдено потрібний. Звичайно, довжина ключа визначає можливу їх кількість і впливає на успішність цього типу атаки. Важливо мати на увазі, що сила шифрування прямо пропорційна розміру ключа, але, оскільки розмір ключа збільшується, зростає кількість ресурсів, необхідних для виконання обчислення.

Альтернативні методи взлому шифру включають атаки сторонніми каналами (англ. side channel attack) і криптоаналіз. Атаки сторонніми каналами каналу спрямовані на реалізацію шифра, а не на самий шифр. Ці атаки можуть бути успішними, якщо є помилка в проектуванні або виконанні системи. Аналогічно, криптоаналіз спрямований на знаходження слабкості в шифрі і його експлуатації. Криптоаналіз частіше виникає, коли в самому шифрі є недолік.

Найбільша слабкість алгоритмів шифрування полягає в тому, що деякі алгоритми не спроможні генерувати, здавалося б, випадкові рядки зашифрованого тексту, а замість цього створюють впізнавані структури.

Наприклад, коли хакер здатний ідентифікувати шаблон, це значно допомагає йому зламати зашифрований текст.

Це питання також стосується алгоритмів, які генерують шаблони, які є передбачуваними, в результаті певних повторюваних тестів введення даних.

Малоймовірно, щоб хакер зламав усі блоки шифру, але викриття лише декількох блоків вже може призвести до витоку важливих конфіденційних даних – і наслідки можуть бути катастрофічними.

Однак час, зусилля та обчислювальні витрати, необхідні для розкриття алгоритму, такого як, наприклад, AES, роблять це надзвичайно дорогим зусиллям, і ця спроба є досить безглуздою.

Гарною відправною точкою при пошуку недоліків у шифруванні є перегляд генераторів ключів шифрування, які в більшості випадків є лише деякою формою генератора випадкових чисел.

Якщо ви коли-небудь читали що-небудь про шифрування, ви, швидше за все, зіткнулися з тим, що хтось згадав про важливість генератора випадкових чисел. Причина цього полягає в тому, що якщо ви можете змусити генератор випадкових чисел відтворити те ж саме значення, яке було згенеровано під час попереднього шифрування, ви, ймовірно, зможете відтворити оригінальні ключі шифрування.

Приклад цього показаний на рисунку 2.3. Системний час використовується як зерно для генератора слабких випадкових чисел.

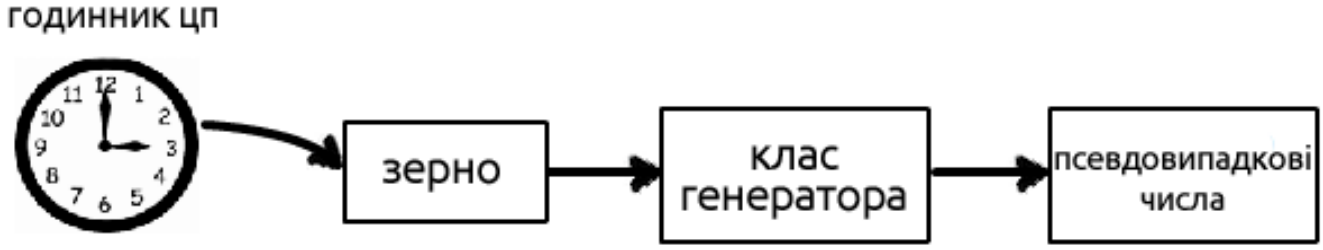

Рисунок 2.3 – Генерація випадкових чисел

Здебільшого будь-який компютерний алгоритм може виконувати лише кінцевий набір операцій. Якщо входи до функції однакові, результат також повинен бути однаковим. Це цілком логічно. У випадку випадкових генераторів, винахідливість полягає в тому, щоб брати достатню кількість входів для отримання

випадкового значення, щоб вихід було легко відтворити. Наприклад, деякі слабкі генератори приймають час доби як вхідні дані. Хоча це, в певному сенсі, захист, але умови можуть бути легко відтворені. Необхідно використовувати достатньо напіввипадкових входів, щоб отримати достатню ентропію.

Як можна бачити на рисунку 2.4, більш ускладнений генератор випадкових даних може прослуховувати аудіодані, в додаток до часу доби, і використовувати вхідні дані миші і ряд інших елементів, щоб спробувати зробити вхідні дані випадковими. Це вимагає невиправданої кількості операцій для перебору або відтворення.

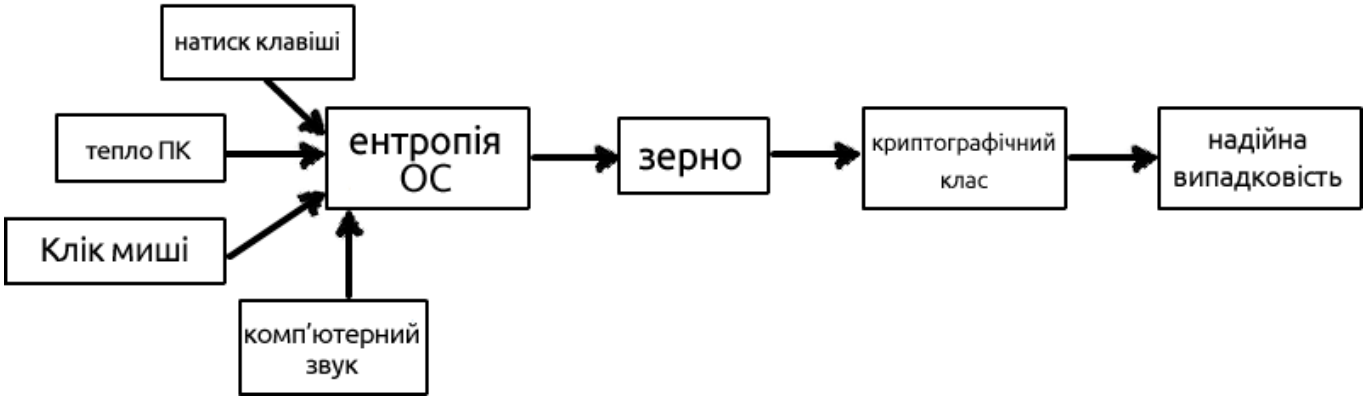

Рисунок 2.4 – Генератор криптографічно стійких псевдовипадкових чисел

Алгоритм DES був розроблений у 1970-х роках і широко використовувався для шифрування. Тепер він вважається слабким алгоритмом через розмір його ключів. Кількість бітів, що генеруються як ключ для алгоритму шифрування, є одним з міркувань щодо міцності алгоритму. Наприклад, був конкурс навзлом 40-бітового шифр, який виграв студент, використовуючи кілька сотень обчислювальних машин у своєму університеті. Це зайняло всього три з половиною години. Чим більше розмір ключа, тим важче буде зламати шифрування, тобто не знаючи нічого про нього.

Найбільші загрози безпеці зашифрованих даних в основному виходять за межі технології. Подумайте про клавіатурних шпигунів, які записують, який ключ або пароль ви вводите, а також бекдори та інші форми шкідливих програм, які використовуються для отримання ключів шифрування.

Для даної системи був обраний асиметричний алгоритм RSA через його простоту реалізації та надійність.

2.1 Генерація ключів

Для того, щоб згенерувати пари ключів виконуються такі дії:

- Вибираються два великі прості числа p та q приблизно 512 біт завдовжки кожне
- Обчислюється їх добуток $n = pq$
- Обчислюється функція Ейлера $\varphi(n) = (p-1)(q-1)$ (2.1)
- Вибирається ціле число e, таке, що $1 < e < \varphi(n)$ та e, взаємно просте з $\varphi(n)$
- За допомогою розширеного алгоритму Евкліда знаходиться число d, таке, що $ed \equiv 1(mod\ \varphi(n))$ (2.2)

Число n називається модулем, а числа e і d – відкритою і секретною експонентами відповідно. Пари чисел (n, e) називаються відкритою частиною ключа, а (n, d) – секретною. Числа p і q після генерації пари ключів не використовуються і можуть бути знищені, але в жодному разі не повинні бути розкриті.

2.2 Алгоритм шифрування

- Взяти відкритий ключ (e, n) та повідомлення m
- Зашифрувати повідомлення $c = E(m) = m^e\ mod\ n$ (2.3)

2.3 Алгоритм дешифрування

- Взяти закритий ключ (d, n) та повідомлення m

- Розшифрувати повідомлення $m = D(c) = c^d \ mod \ n$ (2.4)

Щоб адміністратори системи могли редагувати дані, про користувачів, не турбуючись про ключі, було прийнято рішення зберігати ключі користувачів в базі в зашифрованому вигляді. Ключі для шифрування та дешифрування цих ключів зберігатися на сервері не будуть, також вони не будуть відправлятися на сервер, а шифрування-дешифрування буде відбуватися безпосередню на комп'ютері адміністратора. При авторизації він має вказати файл ключів на комп'ютері, які завантажаться в пам'ять.

ЗАСОБИ РОЗРОБКИ ПРОГРАМНОЇ СИСТЕМИ

Програмна система написана на мовах PHP, JavaScript, HTML та CSS. Серверна частина виконується на веб-сервері Apache, який працює разом з MySQL-сервером.

3.1. Середовище розробки

Розробка та налагодження програмного продукту відбувалося в IDE PhpStorm.

PhpStorm це інтелектуальний редактор для PHP, HTML і JavaScript, який надає можливості аналізу коду на льоту, запобігання помилок у сирцевому коді і автоматизованими засобами рефакторинга для PHP і JavaScript. Автодоповнення коду в PhpStorm підтримує специфікацію PHP з 5-ї по 7-у версії, включаючи генератори, співпрограми, простори імен, замикання, типи даних і синтаксис масивів. Присутній повноцінний SQL-редактор, в якому можна редагувати отримані результати запитів.

PhpStorm надає зручний і інтелектуальний редактор декількох мов з підсвічуванням коду, простою конфігурацією форматування коду, перевіркою помилок ще при написанні коду і розумним автодоповненням. PhpStorm може використовуватися як при розробці на останніх версіях PHP, так і для традиційних проектів.

3.1.1 Редактор коду PHP

- Підтримка PHP 5.3, 5.4 та 5.5, включаючи генератори, простори імен, замикання, типажі, доступ до члена класу при інстанціюванні, синтаксис коротких масивів, розіменування масиву при виклику функції, бінарні літерали, вираження в статичних викликах тощо.

- Автодоповнення коду підказує імена класів, методів, змінних, ключові слова PHP, а також часто вживані імена властивостей та змінних залежно від їх типу.
- Підтримка стандартів оформлення коду (PSR1/PSR2, Drupal, Symfony2, Zend).
- Підтримка PHPDoc. PhpStorm надає відповідне автодоповнення коду, засноване на анотаціях в @property, @method і @var.
- Детектор дубльованого коду.
- PHP Code Sniffer (phpcs), який перевіряє код на льоту
- Рефакторинги (перейменування, введення змінної/константи/поля, вбудовування змінної).
- Підтримка редагування Smarty шаблонів (підсвічування синтаксичних помилок, автодоповнення до функцій і атрибутів Smarty, автоматична вставка парних дужок, лапок та закриваючих тегів тощо)
- MVC подання для фреймворків Symfony2 і Yii
- Розпізнавання коду, запакованого в PHAR-архіви.

3.1.2 Середовище розробки

- Підтримка SQL і баз даних (Рефакторинг схеми бази даних на льоту, генерація міграційних скриптів, можна експортувати результат виконання запиту у файл або буфер обміну, редагування збережених процедур і т.д.).
- Віддалене розгортання додатків використовуючи протоколи FTP, SFTP, FTPS та інші з автоматичною синхронізацією.
- Інтеграція з системами управління версіями, що дозволяє робити дії, такі як commit, merge, diff та інші, прямо з редактору.
- Локальна історія (Local History) (відстежує всі зміни в коді та дозволяє повернути їх частини).

- PHP UML (Діаграми класів UML з рефакторингом, що викликаються прямо з діаграми).
- Підтримка Phing (автодоповнення, перевірка стандартних тегів, властивостей, імена цілей, значень атрибутів.
- Інтеграція з системами відстеження помилок
- Підтримка Vagrant, SSH консолі і віддалених інструментів
- Підтримка Google App Engine For PHP
- PhpStorm також дозволяє різні поєднання клавіш для підвищення ефективності.

3.1.3 Налагодження і тестування

- Візуальний налагоджувач, який легко налаштовувати, для перевірки відповідних контексту локальних змінних і заданих користувачем об'єктів, у тому числі масиви та складні об'єкти, а також редагування значень на льоту.
- Інтеграція з налагоджувачем: скрипти можна профілювати прямо з PhpStorm за допомогою Xdebug або Zend Debugger. Доступний звіт; користувач може перейти від статистики виконання прямо до функції в коді.
- Інтеграція з фреймворком модульного тестування PHPUnit (тести можна розробляти в PhpStorm і відразу запускати з певної директорії, файлу або класу за допомогою контекстного меню) з покриттям коду

3.1.4 Робота з JavaScript, CSS і HTML

- Вся функціональність, доступна в WebStorm, включена в PhpStorm
- Автодоповнення коду для мов JavaScript, HTML і CSS (для тегів, міток, ключових слів, змінних, аргументів і функцій).
- Підтримка HTML5

- Live Edit: тобто зміни в коді можна миттєво переглянути у вбудованому браузері без перезавантаження сторінки
- Підтримка CSS/SASS/SCSS/LESS (автодоповнення коду, підсвічування помилок, валідація тощо)
- Zen Coding
- Зручна навігація по коду, пошук використань змінних, властивостей, функцій, перейти до оголошення
- Підтримка ECMAScript Harmony
- Рефакторинг для JavaScript (перейменування, використання змінної / функції, переміщення / копіювання скрипта в окремий файл)
- Налагоджувач JavaScript, а також інтеграція з фреймворками модульного тестування JavaScript

3.2. Технології, використані при розробці модулю

Серверна частина написана на мові PHP, для її роботи необхідний веб-сервер Apache, до якого звертаються користувачі та сервер MySQL, де зберігаються всі дані. Клієнтська частина окрім звичайного HTML використовує JavaScript для реалізації логіки та CSS, щоб отримати привабливий дизайн.

3.2.1 Веб-сервер Apache

Web-сервер Apache є самостійним, некомерційним, вільно розповсюджуваним продуктом. Продукт підтримує велику кількість можливостей, багато з яких реалізовані як окремі модулі, які розширюють основні функціональні можливості. Починаючи від підтримки мов програмування серверної частини до схем аутентифікації. Існують модулі для підтримки таких мов програмування як Perl, Python, Tcl і PHP.

Функції віртуального хостингу дозволяють одній запущеній програмі Apache обслуговувати різні веб-сайти.

Перш за все Apache використовується для надсилання статичних та динамічних веб-сторінок використовуючи протокол HTTP по Інтернету. Багато веб-додатків розроблені на основі середовища та можливостей веб-сервера.

Продукт може виступати в ролі проксі-сервера, який може значно підвищити продуктивність користувачів в локальній мережі при роботі з документами в Інтернеті. Можна вказати такі параметри та налаштування для проксі-сервера:

- типи файлів та директорії, які необхідно кешувати або навпаки, не включати в кеш;
- максимальний обсяг дискового простору, відведений під кеш;
- періодично переглядати та індексувати базу даних кешу, щоб звільнити місце на диску, видаливши застарілі об'єкти.

Apache зіграв ключову роль у розвитку всесвітньої павутини, і продовжує бути найпопулярнішим веб-сервером, а по факту платформою, на яку орієнтуються інші веб-сервери.

Ядро Apache містить основні функції, такі як налаштування файлів конфігурації, протоколу HTTP і завантаження модулів. Ядро (на відміну від модуля) було повністю розроблено Apache Software Foundation без участі третьої сторони. Теоретично ядро Apache може працювати в чистому вигляді без використання модуля. Однак функціональність цього рішення дуже обмежена. Ядро Apache повністю написано на мові програмування C.

Система конфігурації Apache заснована на текстових конфігураційних файлах. Вони мають три умовно налаштованих рівня:

- Конфігурація сервера (httpd.conf).
- Конфігурація віртуального хоста (httpd.conf з версії 2.2 extra/httpd-vhosts.conf).
- Конфігурація рівня директорії (.htaccess).

Він має власну спеціальну мову конфігураційного файлу на основі блоків директив. Майже всі параметри ядра можуть бути змінені за допомогою конфігураційних файлів. Більшість модулів мають свої власні параметри. Деякі модулі використовують файли конфігурації операційної системи (такі як /etc/passwd

та /etc/hosts). Крім того, параметри можна вказати за допомогою ключів командного рядку.

3.2.2 Мова PHP

Веб-сервер інтерпретує PHP у HTML-код, і потім переданє його клієнту. На відміну від скриптової мови JavaScript, користувач не може бачити код PHP, оскільки браузер отримує готовий HTML-код. Це перевага зі сторони безпеки, але інтерактивність сторінок погіршується.

PHP – мова, у код якої можна вставити безпосередньо HTML код, який, у свою чергу, буде коректно оброблений PHP-інтерпретатором. Велика кількість вбудованих функцій PHP дозволяє уникнути написання об’ємних функцій, призначених для програміста, як це відбувається в C або Pascal.

PHP має вбудовані бібліотеки для роботи з MySQL, Oracle, dbm, Hyperware, Informix, PostgreSQL, mSQL, InterBase, Sybase, а завдяки стандарту відкритого інтерфейсу зв'язку з базами даних (англ. Open Database Connectivity Standard, ODBC) можна підключатися до всіх баз даних, до яких існує драйвер [2-4].

Мова PHP буде знайомою розробникам, які працюють в різних областях. Багато мовних конструкцій запозичені з мов C та Perl. Код PHP дуже схожий на той, який можна знайти в типових програмах на мовах C або Pascal. Це значно зменшує початкові зусилля в навчанні мові. PHP – мова, що поєднує в собі переваги Perl та C і спеціально розроблена для роботи в Інтернеті, мова з універсальним і чітким синтаксисом. Хоча PHP є відносно молодою мовою, вона здобула таку популярність серед веб-розробників, що в наш час є найпопулярнішою мовою для створення веб-додатків (скриптів) [5].

Стратегія з відкритим кодом, і розповсюдження вихідного коду в масах, безумовно, позитивно вплинули на багато проектів, насамперед – Linux хоча і успіх проекту Apache сильно підкріпив позиції прихильників Open Source. Це також стосується історії створення PHP, оскільки підтримка користувачів у всьому світі виявилася дуже важливим фактором у розробці проекту PHP. Прийняття стратегії з

відкритим вихідним кодом і безкоштовне розповсюдження вихідних текстів PHP надало користувачам безцінні послуги. Крім того, користувачі PHP по всьому світу – це якась спільна підтримка, і навіть найскладніші питання можна знайти в популярних електронних конференціях.

Ефективність є дуже важливим фактором у програмуванні для багатокористувацьких середовищ, які включають і веб. Важливою перевагою PHP є те, що вона належить до інтерпретованих мов. Це дозволяє обробляти скрипти на відносно високих швидкостях. За деякими оцінками, більшість скриптів PHP (особливо не надто великих) обробляються швидше ніж аналогічні програми, написані на Perl. Але незважаючи на то, що роблять розробники PHP, викинуючі файли, отримані за допомогою компіляції, працюватимуть набагато швидше – в десятки або сотні разів. Однак продуктивність PHP достатня для створення досить серйозних веб-додатків.

Мова явно підтримує Netscape HTTP cookies. Це дозволяє встановлювати і читати невеликі сегменти даних на стороні клієнта. PHP надає можливість організувати роботу з користувачем під час сеансу (в мові це названо сесією). В сесії можна зберігати різні дані, включаючи об'єкти [6].

3.2.3 СКБД MySQL

База даних веб-сайтів є механізмом для зберігання, доступу, трохи обробки та отримання інформації. Всі динамічні сторінки, які містять структуровані дані, потребують бази даних. Дизайн бази даних для веб-сайту є одним з ключових моментів раннього розвитку. Цей процес повинен здійснюватися під контролем висококваліфікованого фахівця. Саме на цьому етапі закладаються основи, які надалі впливають на швидкість функціонування й складність розробки всього проекту в майбутньому [7].

MySQL – компактний багатопотоковий сервер баз даних. Він відрізняється високою швидкістю, стійкістю і простотою використання. Це одна з найпоширеніших систем управління базами даних. Вона використовується, в першу

чергу, для створення динамічних веб-сторінок, оскільки має відмінну підтримку зі сторони різноманітних мов програмування.

MySQL вважається гарним рішенням для малих і середніх програм. Вихідні коди сервера скомпільовані на багатьох платформах. Найбільш повно функції сервера виявляються в UNIX-системах, де є підтримка багатопоточності, що підвищує загальну продуктивність системи.

Як правило, сервери, які працюють на PHP, автоматично включають підтримку цієї СКБД.

Можливості сервера MySQL:

- простота установки та використання;
- підтримує необмежену кількість користувачів, що працюють одночасно з базою даних;
- кількість рядків у таблицях може досягати 50 мільйонів;
- висока швидкість виконання команд;
- наявність простої та ефективної системи безпеки.

До основних переваг MySQL належать:

- Масштабованість. MySQL може підтримувати роботу бази даних значних розмірів, що підтверджують її реалізації в Google, HP, Yahoo!, Associated Press. Згідно документації, що поставляється з MySQL, деякі бази даних, що використовуються компанією MySQL AB (розробником MySQL), зберігають до 50 мільйонів записів.
- Переносимість. MySQL працює на різних платформах, включаючи Unix, Linux, Windows, OS/2, Solaris, Mac OC. Крім того, MySQL працює на різних платформах.
- Зв’язаність. MySQL має мережеву структуру. Доступ до MySQL можна отримати із будь-якого місця в Інтернеті кільком користувачам одночасно. MySQL має цілий ряд програмних інтерфейсів додатків (API), які дозволяють під’єднуватися з MySQL із додатків, написаних на таких мовах як C, C++, Perl, PHP, Java, Python.

- Безпека. MySQL має систему контролю доступу до даних та забезпечує шифрування даних при передачі.
- Швидкість функціонування.
- Простота у використанні. MySQL дуже проста в установці і розгортанні, легка в управлінні.
- Відкритий код.

Переваги використання MySQL:

- Це одна з найшвидших систем управління базами даних;
- Підтримується з коробки на серверах PHP;
- Установка дуже проста, а вивчення займе значно менше часу, ніж знайомство з альтернативними продуктами;
- Містить стандартний набір засобів адміністрування, але вони можуть бути розширені при покупці платної ліцензії.

Зв'язка технологій PHP + MySQL – один з найбільш зручних і прикладних інструментів у сучасному програмуванні [8,9].

3.2.4 Мова JavaScript

JavaScript є мовою програмування, що дозволяє реалізувати ряд комплексних рішень у веб-документах. Вона допомагає зробити сторінки сайту більш інтерактивними та обробляє дії користувачів. Це об'єктно-орієнтована клієнтська мова, яку використовують додатки, що працюють з дизайном сайту.

JavaScript класифікується як прототипна (підмножина об'єктно-орієнтованої), скриптову мову програмування сценаріїв з динамічною типізацією. Разом з прототипною, в JavaScript частково реалізовано інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості, серед яких: динамічна та слабка типізація, автоматичне керування пам'яттю, прототипне наслідування [10], функції як об'єкти першого класу.

JavaScript використовується для:

- написання сценаріїв веб-сторінок для надання їм інтерактивності;

- створення односторінкових веб-застосунків (React, AngularJS, Vue.js);
- серверне програмування (Node.js);
- стаціонарних застосунків (Electron, NW.js);
- мобільні додатки (React Native, Cordova);
- сценаріїв в прикладному ПЗ (наприклад, як в програмах зі складу Adobe Creative Suite чи Apache JMeter);
- у документах PDF тощо.

JavaScript має багато особливостей об'єктно-орієнтованої мови, але завдяки концепції прототипування підтримка об'єктів в ній відрізняється від традиційних мов ООП. Крім того, JavaScript має ряд властивостей, притаманних всім функціональним мовам, – функції є об’єктами першого класу, списки є об’єктами, анонімні функції, замикання (closures) [11] – що надає мові гнучкості.

JavaScript має синтаксис, подібний до C, але принципові відмінності щодо C:

- об'єкти, з можливістю самоаналізу і динамічної зміни типу через механізм прототипів
- функції як об'єкти першого класу
- обробка винятків
- автоматичне приведення типів
- автоматичне збирання сміття
- анонімні функції

В JavaScript є вбудовані об’єкти (типи): Global, Object, Error, Function, Array, String, Boolean, Number, Math, Date, RegExp. Крім того, JavaScript включає в себе набір вбудованих операцій, які, грубо кажучи, не обов'язково є функціями або методами, а також набором вбудованих операторів, які керують логікою виконання програми. По суті, синтаксис JavaScript відповідає синтаксису Java (який, зрештою, успадковується від C), але спрощується порівняно з Java для легкого вивчення. Наприклад, декларація змінної не містить свого типу, властивості також не мають типу, а оголошення функції може бути в тексті програми після її використання.

Оскільки JavaScript є інтерпретатором, без строгої типізації, і може працювати в різних середовищах, кожне зі своїми власними особливостями сумісності,

розробник повинен бути дуже обережним і повинен переконатися, що його код виконується так, як очікується в широкому діапазоні можливих конфігурацій [12].

Кожен блок скриптів інтерпретатор виконує окремо. На веб-сторінках, коли справа доходить до об’єднання блоків JavaScript та HTML, легше знайти синтаксичні помилки, якщо ви зберігаєте функції скрипта в окремому кодовому блоці або (навіть краще) використовуєте багато невеликих пов'язаних файлів і підключаєте їх [13]. Таким чином, синтаксична помилка не викликає «падіння» цілої сторінки, і можна надати допомогу, елегантно вийшовши зі сторінки.

3.2.5 Мова HTML

Веб-браузери приймають документи HTML з веб-сервера або з локального файлу та передають документи на веб-сайти. HTML описує структуру веб-сторінки і деякі правила для зовнішнього вигляду документа [14].

Елементи HTML є складовою частиною HTML сторінок. За допомогою конструкцій мови можна вставляти зображення та інші об'єкти, такі як інтерактивні форми, на відображувану сторінку. HTML надає інструменти для створення структурованих документів, позначаючи структуру тексту, наприклад заголовки, абзаци, списки, посилання, цитати та інші елементи. Елементи HTML визначаються тегами, написаними з використанням кутових дужок . Одні теги безпосередньо вводять вміст на сторінку. Інші теги оточують і надають інформацію про текст документа і можуть містити інші теги як під-елементи. Браузери не відображають теги HTML, а використовують їх для інтерпретації вмісту сторінки.

У HTML можна вставляти скрипти, написані на мовах як JavaScript, що впливає на поведінку та вміст веб-сайту. За домопогою CSS можна змінювати вигляд і компонування тексту сторінки. World Wide Web Consortium (W3C), який контролює стандарти HTML і CSS, заохочує використання CSS разом зі звичайним HTML з 1997 року.

HTML реалізує інструменти для:

- створення структурованого документа шляхом введення текстової структури: заголовки, абзаци, списки, таблиці, цитати та інше;
- отримання інформації з Інтернету через гіперпосилання;
- створення інтерактивних форм;
- включення в текст зображень, звуку, відео, та інших об'єктів.

Розмітка в мові HTML складається з чотирьох головних компонентів: елементів (та їхніх атрибутів), основних типів даних, символічних мнемонік та декларації типу документа [15-18].

Документ HTML 5.2 складається з трьох частин:

- Декларація типу документа, на початку документа, в якій визначається тип документа (DTD).
- Шапка документа, в якій записано загальні технічні відомості або додаткова інформація про документ, яка не відображається безпосередньо в браузері;
- Тіло документа, в якому міститься основна інформація документа.

Елементи є основними компонентами розмітки HTML. Кожен елемент має дві основні особливості: атрибути і зміст (контент). Для кожного атрибуту та елемента контенту є деякі правила, які повинні бути виконані для того, щоб документ HTML був визнаний валідним [14].

Більшість елементів має початковий тег, який має вигляд `<element-name>`, та кінцевий тег, який має вигляд `</element-name>`. Атрибути елемента записуються в початковому тегу відразу після імені елемента, вміст елемента записується між його двома тегами. Наприклад: `<element-name element-attribute="attribute-value">контент елемента</element-name>`. Більшість з атрибутів елемента – це пара «назва-значення», розділених між собою знаком рівності, та записаних у початковому тегу одразу після імені елемента. Значення атрибуту може бути обмежено лапками (подвійними або одиничними), також, якщо значення атрибуту цілком складається з певних символів, його можна не брати в лапки. Проте ігнорування цього правила вважається небезпечним кодом. На відміну від атрибутів типу «назва-значення», є певні атрибути, що впливають на елемент, і які складаються лише з імені.

Більшість елементів можуть мати будь-який із загальних атрибутів: наприклад, атрибут id впроваджує унікальний ідентифікатор елемента по всьому документу. Доданий до URL документа, браузер перейде до елемента на сторінці.

Це може використовуватися:

- таблицями стилів для впровадження презентаційних властивостей;
- браузерами для фокусування уваги на певному елементі;
- скриптами для виконання дій над елементом.

Атрибут title використовується для додавання пояснювального тексту до елемента. У більшості браузерів значення цього атрибуту можна побачити як підказку, що виникає при наведенні курсору на елемент [14,16].

Атрибут class впроваджує інструмент об'єднання схожих елементів у класи [14].

Будь-яка комп'ютерна мова має свою граматику, словник слів і синтаксис. Будь-який документ, написаний на цій мові, повинен відповідати цим правилам. Мова HTML використовує таку машинно-зчитуючу граматику, яка називається DTD, це поганий механізм, успадкований від SGML.

Проте, так само як і тексти природних мов можуть містити граматичні помилки, документи, які використовують мови розмітки можуть не відповідати певній граматиці. Процес перегляду документу на відповідність визначених мовою правил називають валідацією, а інструмент, який здійснює перевірку – валідатором. Документ, що пройшов цей процес без помилок, називають валідним.

Згідно з цією концепцією, «перевірка розмітки HTML на валідність» визначається як перевірка веб-сторінки на відповідність правилам граматики (визначеними в DTD), на які він посилається із елемента doctype.

Один із вагомих принципів програмування: «Будьте трохи консервативні в тому, що ви робите; будьте трохи ліберальним в тому, що ви приймаєте» [18].

Браузери слідують другій частини цього принципу: вони приймають веб-документи такими, які вони є, та намагаються показати їх на екрані, навіть якщо вони не використовують стандартний HTML. Зазвичай це означає, що браузер спробує «вгадати», що мав на увазі автор документа. Проблема тут в тому, що різні

браузери (або навіть різні версії одного й того самого браузера) будуть робити різні припущення щодо одних і тих же нестандартних конструкцій, а ще гірше, якщо HTML-код дуже відрізняється від стандарту, браузер не зрозуміє його і безладно відтворить сторінку на екрані.

Отже, дотримання першої частини принципу потрібно авторам документа, шляхом перевірки їх відповідності стандарту. Найкращим інструментом є валідатор HTML розмітки.

3.2.6 Мова CSS

Каскадні таблиці стилів (CSS) – це спеціальна мова, яка використовується для опису макетів сторінок, написаних на мові розмітки.

CSS часто використовується для візуального представлення сторінок, написаних HTML та XHTML, але CSS також може використовуватися до інших типів XML-документів.

CSS використовується авторами та відвідувачами веб-сайтів для визначення кольорів, шрифтів, верстки та інших аспектів макета сторінки. Однєю з основних переваг є можливість розділити зміст сторінки (або контент, зазвичай HTML, XML або подібна мова розмітки) від вигляду документу (описаного в CSS) [14,19].

Такий поділ може поліпшити сприйняття і доступність контенту, забезпечити більшу гнучкість і контроль за відображенням контенту в різних середовищах, зробити контент більш структурованим та простим, прибрати повтори тощо. CSS дозволяє використовувати різні правила для контенту в різних умовах відображення (на екрані звичайного монітора, якогось мобільного пристрою, навіть у роздрукованому вигляді, на екрані телевізора, голосових браузерах , пристроях з підтримкою шрифту Брайля та ін.).

Один і той же HTML або XML документ можна переглядати по-різному залежно від використаного CSS. Стиль для відображення сторінки може бути:

- Стилі автора (інформація надана автором сайту):
    - зовнішня таблиця стилів, найчастіше окремий файл або файли .css

- внутрішні таблиці стилів, що входять до складу документу або блоку
- стилі для певного елемента

- Стилі користувача – локальний файл .css, який користувач вибрав для використання на сторінках і вказав в налаштуваннях браузера.
- Стилі браузера – стандартний стиль веб-переглядача, наприклад стандартні стилі для елементів, визначені браузером, використовуються якщо інформації про стиль елемента відсутня або неповна.

Стандарт CSS визначає порядок і діапазон застосування стилів, тобто, в якій послідовності і для яких елементів застосовуються стилі. В результаті, використовується принцип каскаду, коли для елементів вказується лише ті правила, що змінилися або не визначені більш загальними стилями.

Переваги [20]:

- Інформація про стиль для усього сайту або його частин можуть увійти до одного файлу .css, що дозволяє швидко робити зміни в дизайні сторінок;
- Різна інформація про стилі для різних видів користувачів: наприклад збільшений розмір шрифту для користувачів з послабленим зором, окремі стилі для виводу сторінки на принтер, стиль для мобільних пристроїв;
- Сторінки зменшуються в об’ємі та стають більш структурованими, оскільки інформація про стилі винесена в окремий файл;
- Прискорення завантаження сторінок за рахунок зменшення обсягів інформації, що передається, навантаження на сервер та канал передачі. Досягається це за рахунок того, що всі сучасні браузери здатні кешувати в пам’ять файли зі стилями і використовувати для всіх сторінок, а не завантажувати для кожної.

CSS має відносно простий синтаксис і використовує кілька англійських слів для найменування різних елементів стилю [21].

Стилі складаються зі списку правил. Кожне правило має один або більше селекторів та блок визначення. Блок визначення складається зі списку властивостей, оточеного фігурними дужками [22,23].

Властивості в списку оформлюються у такому вигляді: назва властивості, двокрапка (:), значення, крапка з комою (;).

3.2.7 Інструменти Bootstrap

Bootstrap – це веб-фреймворк, який створено для спрощення розробки інформаційних веб-сторінок. Основною метою додавання до веб-проекту є застосування кольору, розміру, шрифту та макета до цього проекту. Тобто, головним чинником є те, що розробники можуть обрати стилі за своїм смаком. Після додавання до проекту Bootstrap надає основні визначення стилів для всіх елементів HTML. Кінцевим результатом є єдиний вигляд для тексту, таблиць і елементів форми в веб-браузерах. Крім того, розробники можуть скористатися перевагами класів CSS, визначених у Bootstrap, для подальшого налаштування зовнішнього вигляду їхнього вмісту [24]. Наприклад, Bootstrap надав для таблиць світлих і темних кольорів, заголовків сторінок, більш виразних лапок і тексту з виділенням.

Bootstrap має модульну структуру і складається в основному з наборів таблиць стилів LESS, кожен з яких реалізує певні компоненти цього набору інструментів. Розробники можуть самостійно включати певні файли Bootstrap, обираючи компоненти для свого проекту.

Основні інструменти Bootstrap [25]:

- Сітки (grid) – наперед задані, готові до використання колонки
- Шаблони (template) – фіксовані чи адаптивні шаблони сторінок
- Типографіка (typography) – опис та визначення класів для шрифтів, серед яких шрифти для коду, цитат тощо
- Мультимедіа (media) – засоби управління зображеннями та відео
- Таблиці (table) – засоби оформлення таблиць, які зокрема забезпечують сортування
- Форми (form) – класи для оформлення привабливих форм, та деяких подій

- Навігація (nav, navbar) – класи для оформлення панелей навігації, сторінок, вкладок, меню і т.д.
- Сповіщення (alert) – класи для оформлення модальних вікон, підказок і спливаючих вікон для комунікації з користувачем
- Іконочний шрифт (icon font) – набір іконок у вигляді шрифту, складається майже з 500 компонентів.

ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ОБРОБКИ ТА ШИФРУВАННЯ ДАНИХ

Аналізуючи поставлену задачу та деякі методи її вирішення, було вирішено розробити програмну систему як веб-додаток. Головною перевагою перед іншими варіантами є його універсальність і можливість використання на будь-яких пристроях без встановлення на цільову операційну систему (потрібен лише сучасний браузер)

4.1 Вибір архітектури програмного комплексу

Для реалізації поставленої задачі було вирішено використовувати дволанкову архітектуру, яка складається з клієнтської та серверної частини. Схема даної архітектури зображена на рисунку 4.1.

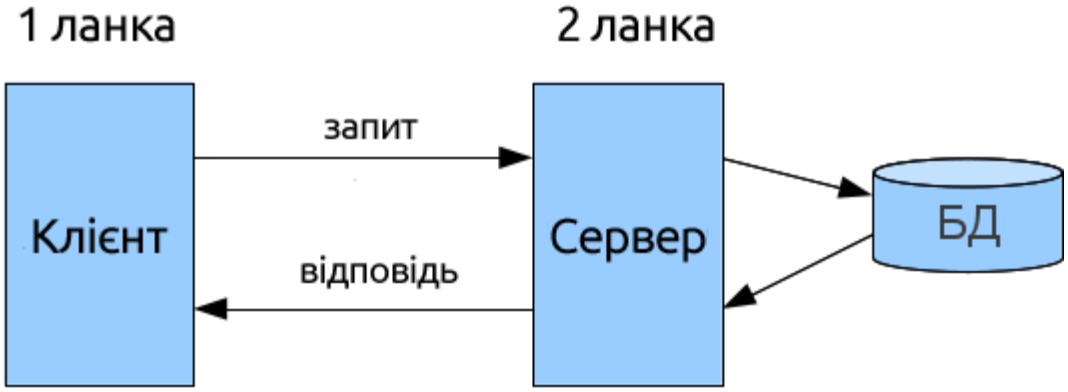

Рисунок 4.1 – Архітектура програмної системи

Дволанкова архітектура використовується в простих клієнт-серверних

системах, де сервер відповідає на запити клієнта безпосередньо і в повному обсязі, при цьому сервер використовує лише власні ресурси. Тобто не викликає сторонні мережеві програми, та не звертається до сторонніх ресурсів для виконання будь-якої частини запиту. Така архітектура є найпростішою, її легко організувати та підтримувати.

До сервера може звертатися відразу кілька клієнтів і їх максимальна кількість залежить від потужності сервера і від того, що хоче отримати клієнт від сервера.

Візуальна частина була спроектована за допомогою HTML5, CSS3 та JavaScript. Для прискорення та спрощення проектування використовувалася бібліотека Bootstrap, що розроблена на основі CSS та JavaScript.

Головним центром програмного комплексу є сервер. У ньому зосереджена основна бізнес-логіка та логіка доступу до бази даних. За допомогою серверу відбувається ідентифікація користувача для надання індивідуального доступу до програмного застосунку. Сервер є єдиним зв’язком між користувачем та базою даних, щоб унеможливити пошкодження даних та їх використання не за призначенням. Для того, щоб користуватися програмою, потрібно бути авторизованим у цій системі, тому дана логіка реалізується на рівні серверу, тому що на рівні користувача можлива підміна прав доступу та інші мтоди неконтрольованого доступу до даних.

Під час користування програмою, користувач взаємодіє з клієнтським додатком, яким є веб-сайт в даниму випадку. На рівні користувача реалізовано програмний інтерфейс, за допомогою якого відбувається налаштування програми та перегляд результатів роботи. Також на користувацькому рівні відбувається попередня обробка даних та шифрування перед відправленням на сервер і також дешифрування відповіді від сервера.

4.2 Опис архітектури серверу

Шаблон проектування – це архітектура, рішення яке описує яким чином вирішуються задачі, які часто зустрічаються при розробці програмних систем чи додатків.

Для проектування серверної частини використовувався шаблон проектування MVC (Model View Controller) (рисунок 4.2).

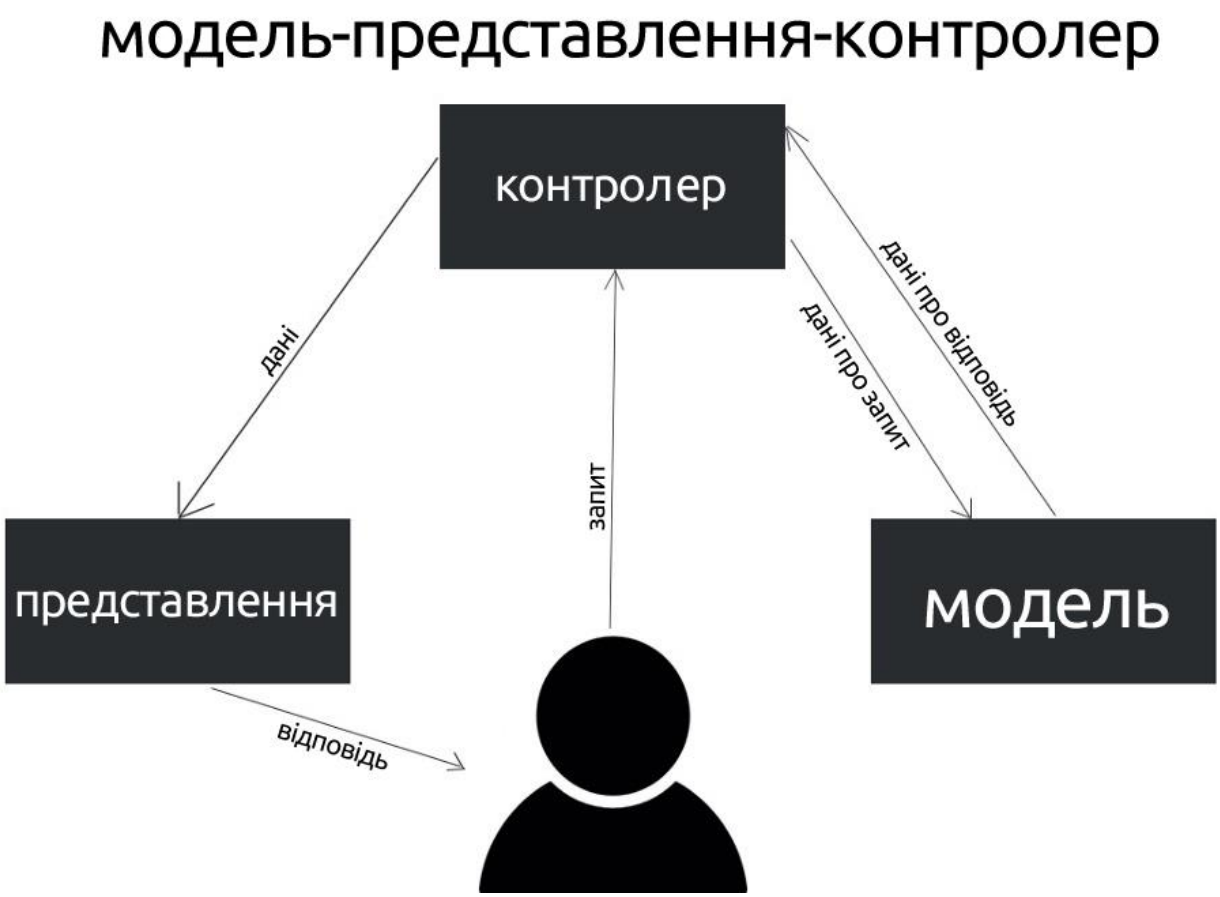

Рисунок 4.2 – Паттерн проектування MVC

Принципом MVC є розділення програмної архітектури системи на три головні компоненти: Model (Модель), View (Представлення), Controller (Контролер), таким чином, що редагування будь-якого компонента може відбуватися незалежно.

Модель – містить знання про предметну область, дані та правила доступу до цих даних, але нічого не знає про контролери та представлення. У моделі відбувається бізнес-логіка роботи застосунку. Модель повертає контролеру дані які запрошує користувач, або інша система.

Представлення – надає можливість по-різному відображати дані отримані будь-яким способом від моделі, може містити в собі логіку. Представлення - це кінцевий інтерфейс, з яким взаємодіє користувач. Користувач може передавати дані через представлення.

Контролер – реалізує взаємодію між моделлю і представленням. У функції контролера входить відстеження певних визначених подій, які виникають в результаті дій користувача. Контролер дозволяє структурувати код шляхом групування пов’язаних дій в окремий клас.

Основна мета застосування цієї концепції полягає в відділенні логіки (та моделі) від її візуалізації (представлення) . За рахунок такого поділу підвищується можливість повторного використання та гнучкість системи. Також для великих проектів, застосування MVC або похідних цього шаблону дозволяє застосувати тестування різноманітних компонентів в цій програмній системі.

4.3 Опис таблиць бази даних

Розглянемо більш детально структури кожної із таблиць бази даних системи.

Детальна інформація про їх структури (ім’я, тип і розмір поля, опис поля) приведена у таблицях 4.1 – 4.3.

Таблиця 4.1. Структура таблиці «Користувач»

Ім’я поля	**Тип і розмір поля**	**Опис поля**
client_id	int	Первинний ключ
e	int	Частина RSA ключа
d	int	Частина RSA ключа
n	int	Частина RSA ключа
fio	text	Зашифроване ПІБ користувача

Таблиця 4.2. Структура таблиці «Записи»

Ім’я поля	Тип і розмір поля	Опис поля
client_id	int	Id користувача
record_id	int	Первинний ключ
data	text	Текст запису
attachment	int	Прикріплене зображення

Таблиця 4.3. Структура таблиці «Прикріплення»

Ім’я поля	Тип і розмір поля	Опис поля
Id	int	Первинний ключ
body	mediumblob	Частина зображення у бінарному вигляді
header	text	Зашифрована частина зображення

4.4 Апаратно-програмні вимоги до програмної системи

Для коректного функціонування програмної системи необхідно забезпечити наступну апаратно-програмну конфігурацію:

- операційна система *Windows 7, 8, 10, MacOS 10+, Linux Ubuntu 10+*;
- *2 Гб* оперативної пам’яті;
- *200 Мб* вільного місця на жорсткому диску;
- веб-сервер *Apache* з модулем *PHP*;
- сервер *MySQL*;
- для користувачів веб-браузер з доступом до серверу даних та підтримкою *JavaScript*.

4.5 Встановлення та налаштування

Для встановлення програми потрібно з носія з дистрибутивними файлами перенести в папку веб-сервера (зазвичай – це *htdocs*) виконуючі файли програми (рисунок 4.1).

На наступному кроці необхідно імпортувати файл бази даних до *MySQL* сервера через командний рядок (рисунок 4.2) або утиліту *phpMyAdmin*.

Для імпорту бази даних з використанням командного рядка потрібно виконати пошук та відкрити кореневу папку носія та ввести команду:

mysql –u ім'я користувача –p < db.sql

Після цього програма запитає пароль (рисунок 4.2). При розробці використовується користувач по замовчуванню *root* без паролю.

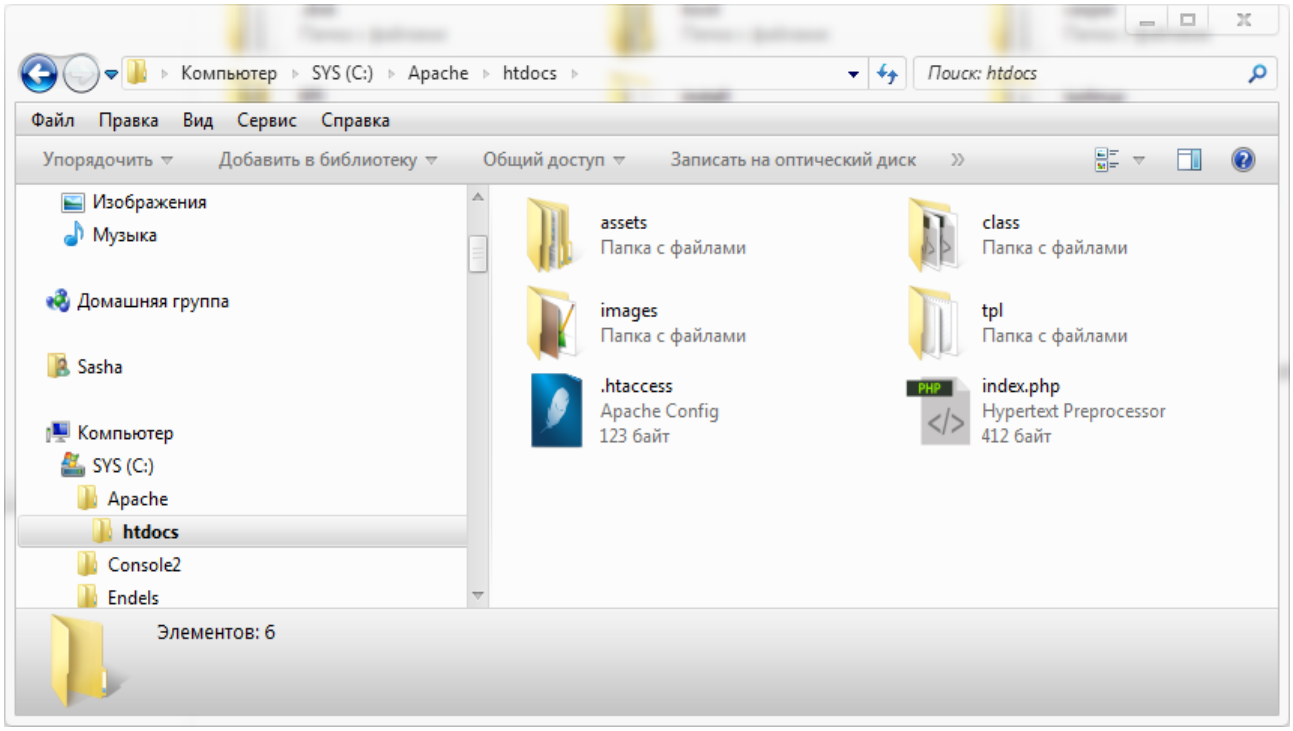

Рисунок 4.1 – Кореневий каталог веб-сервера

При успішному імпорті програма завершиться без помилок.

Після імпорту завантажити веб-сервер *Apache* та сервер *MySQL*.

Після запуску проект буде доступний розробнику на локальному сервері за посиланням *http://localhost/*

Користувачі повинні відкриті в браузері локальну або інтернет-адресу веб-сервера.

Рисунок 4.2 – Імпорт бази даних

4.6 Користування програмною системою

При відкриванні кореневої сторінки клієнт побачить форму вводу даних за якими від може переглянути персональні записи (рисунок 4.3)

Рисунок 4.3 – Форма вводу секретних даних

В перше поле треба ввести унікальний ідентифікатор пацієнта, а в друге закритий ключ RSA, який пацієнт має отримати в медичному закладі та надійно зберігати. Після вводу та натискання кнопки відправки система завантажить з

сервера дані, розшифрує їх наданим ключем та відобразить у вигляді таблиці (рисунок 4.4). Серед них ПІБ, безпосередньо текст запиту та зображення.

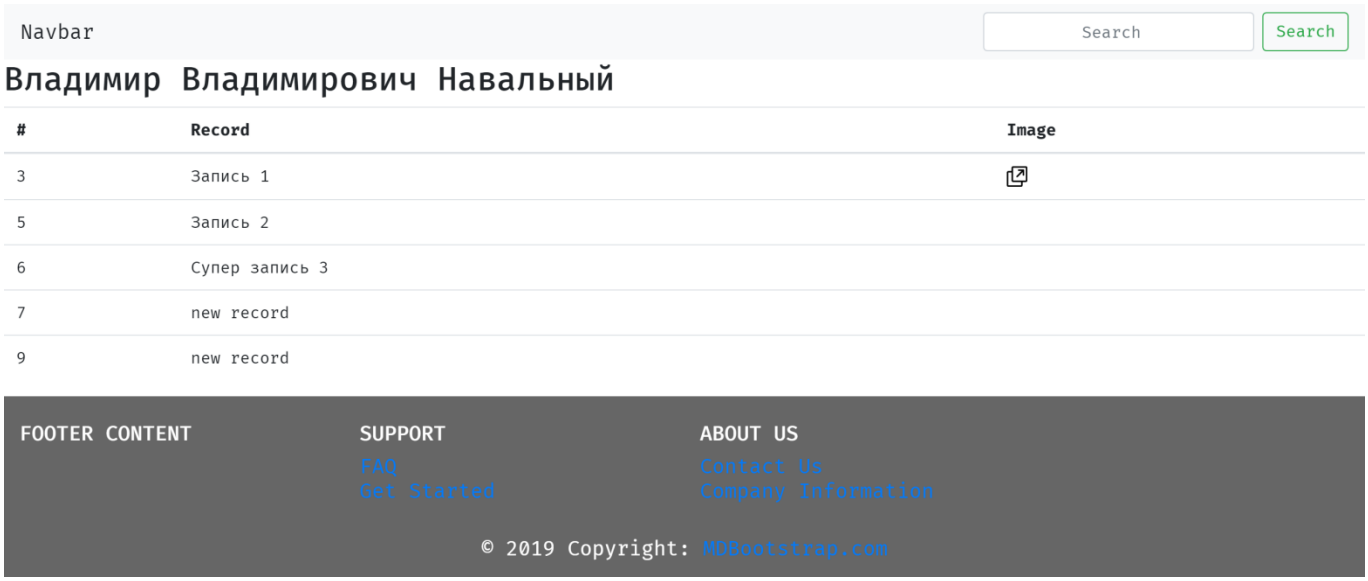

Рисунок 4.4 – Сторінка з даними

Для перегляду зображення потрібно натиснути по гіперпосиланню у вигляді нової сторінки.

Якщо в браузері перейти до директорії `admin/` то користувач побачить форму входу до адміністраторського розділу (рисунок 4.5). Тут потрібно ввести логін та вибрати файл з ключами на комп'ютері користувача.

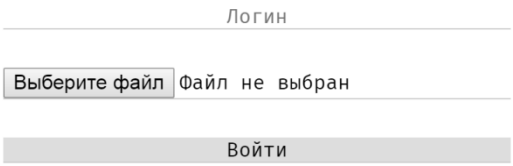

Рисунок 4.5 – Форма входу до адміністраторського розділу

Після входу система завантажить з сервера список користувачів та відобразить їх у вигляді списку (рисунок 4.6). При натисканні лівою кнопкою миші по рядку користувач перейде до сторінки обраного пацієнта (рисунок 4.7), при натисканні правою видалить обліковий запис з бази. По посиланню «новий» можна створити

новий обліковий запис користувача, який з’явиться в списку зі стандартним ім’ям і його можна буде редагувати.

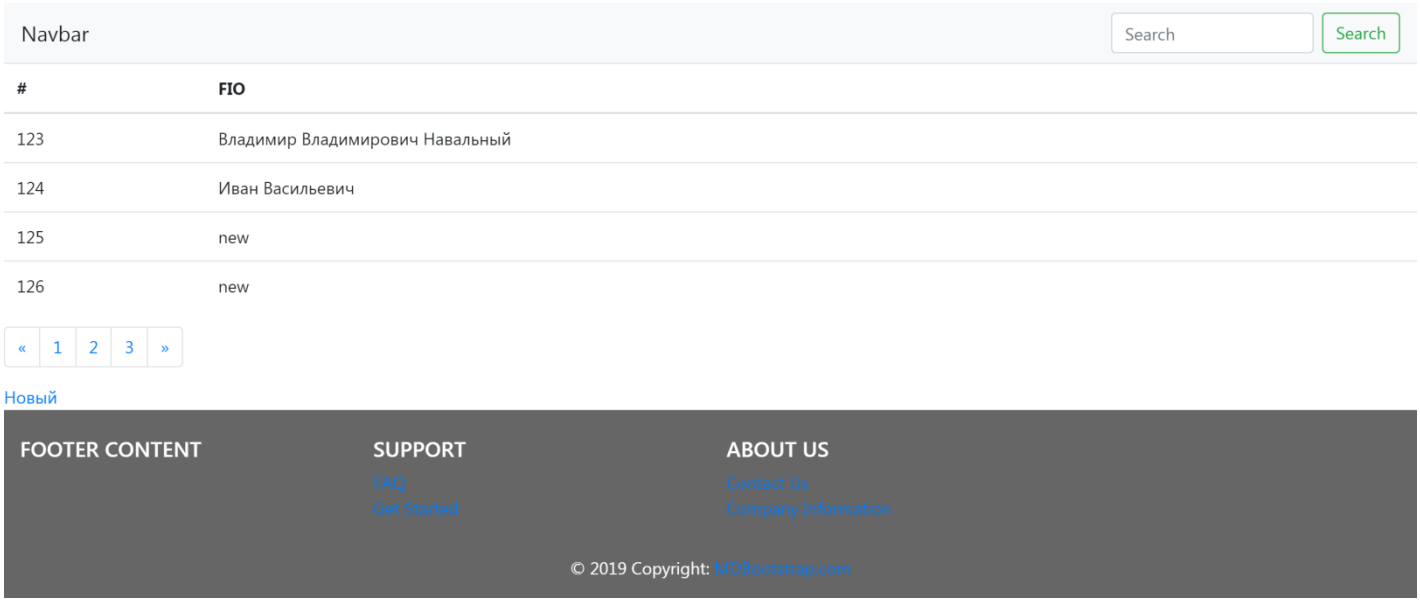

Рисунок 4.6 – Список пацієнтів

На сторінці пацієнта (рисунок 4.7) аналогічно клік лівою кнопкою миші дозволяю редагувати текст запису, а правою видалить його. Посилання «новий» створить новий запис. Клік по імені користувача дозволяє редагувати його. В стовпчику з зображеннями можна переглянути існуюче зображення або завантажити, якщо його немає.

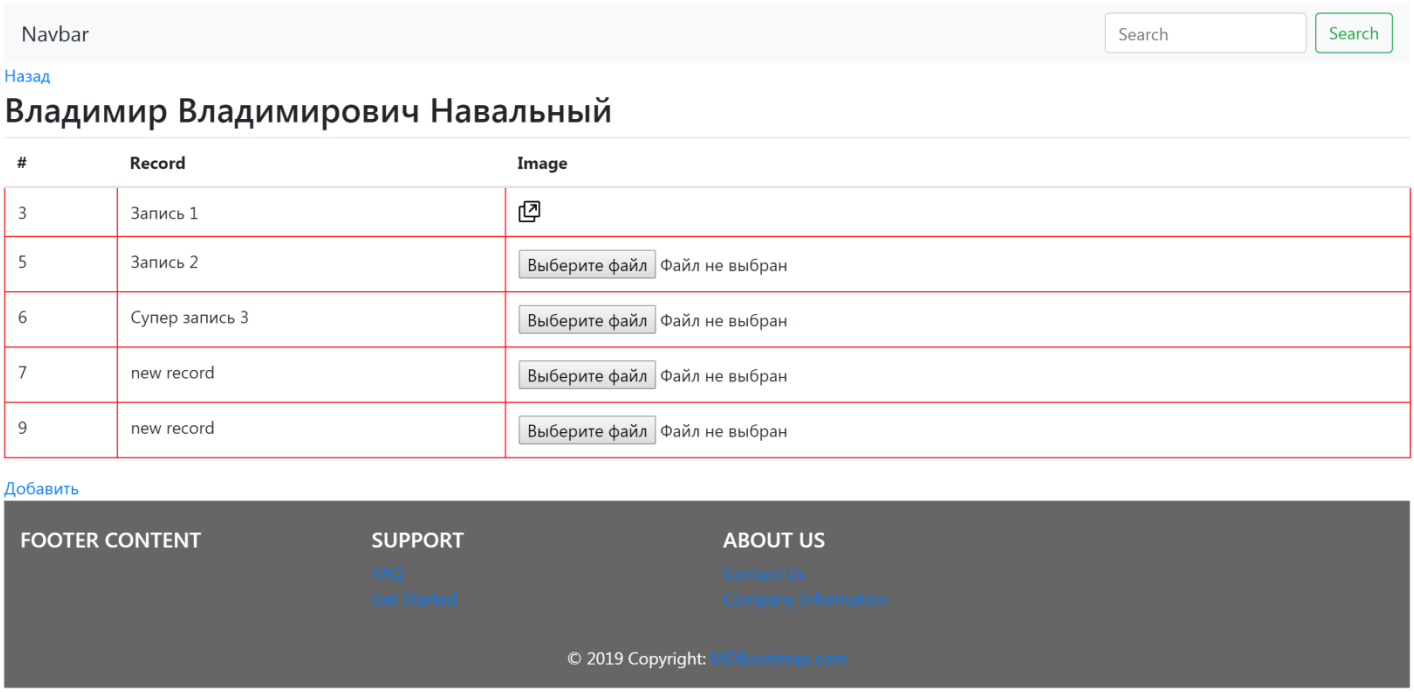

Рисунок 4.7 – Сторінка редагування пацієнта

ВИСНОВКИ

Під час розробки програмної системи досліджено основні принципи клієнт-серверної взаємодії, вирішено проблеми передачі та збереження складних структур даних в базі. У ході аналізу існуючих алгоритмів шифрування досліджено головні переваги та недоліки кожного з них, і обґрунтовано використання асиметричного алгоритму RSA. Досліджено різноманітні техніки та алгоритми захисту даних, було виявлено різноманітні сфери, де можна застосовувати дані методи. Також створено декілька прототипів програмного забезпечення, які вирішували різноманітні аспекти поставленої задачі, а також які лягли в основу розробленого програмного продукту.

При створенні продукту було запропоновано простий алгоритм шифрування та зберігання даних в захищеному вигляді. Для зручності користувачів система зберігає ключі, якими шифруються персональні дані, а для безпеки ключі шифрування зберігаються не в системі, а користувачами на носіях.

В результаті була розроблена система попередньої обробки, шифрування, передачі та зберігання даних в захищеному вигляді. Продукт написаний у вигляді веб-додатку, тому є універсальним і може бути використаним на будь-якій операційній системі, на якій встановлено браузер, який підтримує останні веб-стандарти, а також яка має постійний доступ до Інтернету. Користувачами системи можуть бути звичайні пацієнти та працівники медичних закладів.

Матеріали роботи доповідалися на конференції «Сучасні проблеми сталого розвитку енергетики», що відбулася в Києві 23 квітня 2019 року по темі «Програмна система шифрування даних на базі RSA методу».

На окремі елементи алгоритмічної моделі отримано свідоцтво про реєстрацію авторського права №88797 (див. Додаток Г).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Martin Fowler — GUI Architectures. Часть 2 [Електронний ресурс]. — 2009 — Режим доступу: https://habr.com/post/53536/.
2. Котеров Д. В. PHP 7 / Д. В. Котеров, И. В. Симдянов. – СПб.: БХВ-Петербург, 2016. – 1066с.
3. Скляр Д. PHP. Рецепты программирования / Д. Скляр, А. Трахтенберг – СПб.: Питер, 2015 – 784с.
4. Уэнц К. PHP. Карманный справочник. Необходимый код и команды – М: Вильямс, 2007 – 384с.
5. Куссуль Н. Использование PHP. Самоучитель / Н. Куссуль, А. Шелестов – М: Вильямс, 2006 – 272с.
6. Аткинсон Л. PHP 5. Библиотека профессионала / Л. Аткинсон, З. Сураски – М: Вильямс, 2006 – 944с.
7. Болье А. Изучаем SQL – М: Символ-Плюс, 2007 – 309с.
8. Веллинг Л. Разработка веб-приложений с помощью PHP и MySQL / Л. Веллинг, Л. Томсон – М: Вильямс, 2016 – 768с.
9. Колисниченко Д. PHP и MySQL. Разработка Web-приложений – СПб.: БХВ-Петербург, 2016. – 640с.
10. Флэнаган Д. JavaScript. Подробное руководство – М.:Символ-Плюс, 2012 – 1080с.
11. Маккоу А. Веб-приложения на JavaScript– СПб.: Питер, 2012 – 288с.
12. Вайк А. JavaScript. Энциклопедия пользователя / А. Вайк, Р. Вагнер – М: ДиаСофт, 2001 – 480с.
13. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 - СПб.: Питер, 2016. – 816с.
14. W3C Standards HTML & CSS [Электронный ресурс]. — Режим доступу: https://www.w3.org/standards/webdesign/htmlcss .

15. Сергеев А. HTML 4.0 / А. Сергеев, А. Матросов, М. Чаунин – СПб.: БХВ-Петербург, 2008. – 672с.
16. Хольцшлаг М. Использование HTML и XHTML. Специальное издание – М: Вильямс, 2004 – 736с.
17. В. Дунаев HTML, скрипты и стили – СПб.: БХВ-Петербург, 2015 – 816с.
18. Вейл Э. HTML5. Разработка приложений для мобильных устройств – СПб.: Питер, 2015 – 480с.
19. Макфарланд Д. Новая большая книга CSS – СПб.: Питер, 2016. – 720с.
20. Мейер Э. CSS. Карманный справочник – М: Вильямс, 2016 – 288с.
21. Грант К. CSS для профи – СПб.: Питер, 2016. – 496с.
22. Соколов С. CSS3 в примерах. Профессиональная работа – М: Вильямс, 2008 – 352с.
23. Бадд Э. CSS. Профессиональное применение Web-стандартов / Э. Бадд, К. Молл, С. Коллизон – М: Вильямс, 2009 – 272с.
24. Кириченко А. Динамические сайты на HTML, CSS, JavaScript и Bootstrap. Практика, практика и только практика / А. Кириченко, Е. Дубовик – СПб.: Наука и техника, 2016 – 272с.
25. Морето С. Bootstrap в примерах – М: ДМК Пресс, 2016 – 314с.

ДОДАТОК А

Програмна система попередньої обробки та шифрування даних.

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51165_19Б

Аркушів 1

Київ 2019

Позначення	Найменування	Примітки

Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТВ51165_19Б	Записка.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТВ51165_19Б 12-1	RSA.php	Надає методи шифрування, дешифрування та генерації ключів
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТВ51165_19Б 13-1	DB.php	Надає методи для роботи з базою даних
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТВ51165_19Б 14-1	Додаток В.doc	Опис програмного модуля

ДОДАТОК Б

Програмна система попередньої обробки та шифрування даних.

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51165_19Б 12-1

Аркушів 8

Київ 2019

```
class RSA {
	public $e;
	public $d;
	public $n;

	public function __construct($e = null, $d = null, $n = null) {
		$this->e = $e;
		$this->d = $d;
		$this->n = $n;
	}

	public function crypt($num) {
		return self::powMod($num, $this->e, $this->n);
	}
	public function decrypt($num) {
		if (!is_numeric($num)) return null;
		return self::powMod($num, $this->d, $this->n);
	}

	public function raw($str) {
		$r = "";
		foreach(explode(' ', $str) as $num) {
			//echo $num.PHP_EOL;
			//echo $num . " -> ".$this->decrypt(intval($num)).PHP_EOL;
			$r .= mb_chr($this->decrypt($num));
		}
		return $r;
	}
	public function stew($str) {
		$r = [];
		foreach(mb_str_split($str) as $ch) {
			//echo $ch . " -> ".mb_ord($ch).PHP_EOL;
			//echo $ch . " -> ".uniord($ch).PHP_EOL;
			$r[] = $this->crypt(mb_ord($ch));
		}
		return implode(" ", $r);
	}

	private static function powMod($num, $a, $b) {
```

```
        $s = 1;
        for ($j = 1; $j <= $a; $j++) {
                $s *= $num;
                //echo $s;
                //$s %= $b;
                $s = fmod($s, $b);
                //echo " % $b = $s".PHP_EOL;
                //if ($j > 100) die;
        }
        return $s;
}


private static function getRandomPrime() {
        do {
                $flag = false;
                $x = random_int(200, 1200);

                for ($i = 2; $i <= sqrt($x); $i++)
                        if (!($x % $i)) {
                                $flag = true;
                                break;
                        }
        }
        while ($flag);
        return $x;
}
private static function isCoprime ($a, $b) {
        $num;
        while ( $b ) {
                $num = $a % $b;
                $a = $b;
                $b = $num;
        }
        if (abs($a) == 1) return true;
        return false;
}
public static function generate() {
        $p = self::getRandomPrime();
```

```
                $q = self::getRandomPrime();

                $n = $p*$q;

                $fi = ($p - 1)*($q - 1);

                $possible = [];
                for ($j = 2; $j < $fi; $j++)
                        if (self::isCoprime($j, $fi)) $possible[] = $j;

                $e = $possible[random_int(0, count($possible))];


                for ($d = 0; $d < $fi; $d++)
                        if (fmod($d*$e, $fi) == 1) break;

                return ["e"=>$e, "d"=>$d, "n"=>$n];
        }
}

        // code point to UTF-8 string
function unichr($i) {
        return iconv('UCS-4LE', 'UTF-8', pack('V', $i));
}

// UTF-8 string to code point
function uniord($s) {
        return unpack('V', iconv('UTF-8', 'UCS-4LE', $s))[1];
}
function mb_str_split( $string ) {
        return preg_split('/(?<!^)(?!$)/u', $string);
}




abstract class DB {
        public static $mysqli;
```

```
public static function connect() {
	self::$mysqli = mysqli_connect("localhost", "root", "", "data");
	if (mysqli_connect_errno()) return false;

	return true;
}
public static function _query($query, $arr = []) {

	if (!isset(self::$mysqli)) self::connect();

	$t = self::$mysqli->prepare($query);
	if(!$t) {
		echo "$query<br>".PHP_EOL;
		echo self::$mysqli->error;
		DIE;
	}

	//call_user_func_array([$t, 'bind_param'], $arr);
	if (count($arr)) $t->bind_param(...$arr);
	$t->execute();


	if($t->errno) {
		echo $t->error;
		return;
	}

	$meta = $t->result_metadata();
	if (!$meta) return false;
	while ($field = $meta->fetch_field())
		$params[] = &$row[$field->name];

	call_user_func_array(array($t, 'bind_result'), $params);

	$result = [];
	while ($t->fetch()) {
		foreach($row as $key => $val)
			$c[$key] = $val;
		$result[] = $c;
```

```
                }

                $t->close();
                return $result;
            }
        }


$RSA = new RSA($_SESSION['key']['e'], $_SESSION['key']['d'], $_SESSION['key']['n']);

        if (isset($_POST['record_id'], $_POST['text'], $_POST['client_id'])) {

            //print_r($_POST);
            $d = DB::_query("SELECT * FROM `keys` WHERE `client_id` = ?", ["s", $_POST['client_id']]);
            if (empty($d)) die("Not found");
            //print_r($d);

            $RSA = new RSA($RSA->decrypt($d[0]['e']), $RSA->decrypt($d[0]['d']), $RSA->decrypt($d[0]['n']));
            //print_r($RSA);

            $str = [];
            foreach(mb_str_split($_POST['text']) as $ch) {
                echo $ch . " -> ".mb_ord($ch). " -> ".$RSA->crypt(mb_ord($ch)).PHP_EOL;
                //echo $ch . " -> ".uniord($ch).PHP_EOL;
                $str[] = $RSA->crypt(mb_ord($ch));
            }
            $str = implode(" ", $str);

            $d = DB::_query("UPDATE `records` SET `data` = ? WHERE `record_id` = ?",
                ["ss", $str, $_POST['record_id']]);
            if ($d === false) {

            }
        }
        elseif (isset($_POST['client_id'], $_POST['fio'])) {
```

```
		$d = DB::_query("SELECT * FROM `keys` WHERE `client_id` = ?", ["s", $_POST['client_id']]);
		if (empty($d)) die("Not found");
		$RSA = new RSA($RSA->decrypt($d[0]['e']), $RSA->decrypt($d[0]['d']), $RSA->decrypt($d[0]['n']));

		$str = [];
		foreach(mb_str_split($_POST['fio']) as $ch) {
			//echo $ch . " -> ".mb_ord($ch).PHP_EOL;
			//echo $ch . " -> ".uniord($ch).PHP_EOL;
			$str[] = $RSA->crypt(mb_ord($ch));
		}
		$str = implode(" ", $str);
		$d = DB::_query("UPDATE `keys` SET `fio` = ? WHERE `client_id` = ?",
			["ss", $str, $_POST['client_id']]);
	}
	elseif(isset($_GET['new'])) {
		$d = DB::_query("SELECT * FROM `keys` WHERE `client_id` = ?", ["s", $_GET['new']]);
		if (empty($d)) die("Not found");

		$RSA = new RSA($RSA->decrypt($d[0]['e']), $RSA->decrypt($d[0]['d']), $RSA->decrypt($d[0]['n']));

		$r = DB::_query("INSERT INTO `records` VALUES (?, NULL, ?)",
			["ss", $_GET['new'], $RSA->stew("new record")]);

		header("Location: edit.php?id=".$_GET['new']);
	}


$RSA = new RSA($_SESSION['key']['e'], $_SESSION['key']['d'], $_SESSION['key']['n']);
	//print_r($RSA);

	if(isset($_GET['id'])) {

		$d = DB::_query("SELECT * FROM `records` WHERE `client_id` = ?", ["s", $_GET['id']]);
```

```
		$k = DB::_query("SELECT * FROM `keys` WHERE `client_id` = ?", ["s", $_GET['id']]);
		if (empty($k)) die("key Not found");
		$RSA = new RSA($RSA->decrypt($k[0]['e']), $RSA->decrypt($k[0]['d']), $RSA->decrypt($k[0]['n']));
		//print_r($RSA);

		echo "<a href=\"".$_SERVER['PHP_SELF']."\">Назад</a>";

		echo "<h2 class=\"edit\">".$RSA->raw($k[0]['fio'])."</h2>";


		if(empty($d)) echo("No records found<br>".PHP_EOL);
		else {
			echo "<table>".PHP_EOL;
			foreach($d as $record) {
				//print_r($record);

				echo "\t<tr>".PHP_EOL;
				echo "\t\t<td>{$record['record_id']}</td>".PHP_EOL;
				echo "\t\t<td class=\"edit\">";

				foreach(explode(' ', $record['data']) as $num) {
					//echo $num.PHP_EOL;
					//echo $num . " -> ".$RSA->decrypt($num).PHP_EOL;
					echo mb_chr($RSA->decrypt($num));
				}

				echo "</td>".PHP_EOL;
				echo "\t</tr>".PHP_EOL;
				//sleep(3);
			}
			echo "</table>".PHP_EOL;
		}
		echo "<a href=\"mod.php?new=".$_GET['id']."\">Добавить</a>";
		echo "<script src=\"edit.js\"></script>";
		echo "<style>td {min-width: 40px;border: 1px solid red}</style>";
		die;
	}
```

```
$count = 15;
$page  = empty($_GET['page']) ? 1 : intval($_GET['page']);

$sql = "SELECT * FROM `keys` LIMIT ".($page * $count - $count).", ".$count;
$rows = DB::_query($sql);

//$d = DB::_query("SELECT `client_id`, COUNT(*) FROM `records` GROUP BY `client_id`");
//print_r($d);

echo "<table>".PHP_EOL;
foreach($rows as $row) {
	$RSA = new RSA($_SESSION['key']['e'], $_SESSION['key']['d'], $_SESSION['key']['n']);
	$k = DB::_query("SELECT * FROM `keys` WHERE `client_id` = ?", ["s", $row['client_id']]);
	if (empty($k)) die("key Not found");
	$RSA = new RSA($RSA->decrypt($k[0]['e']), $RSA->decrypt($k[0]['d']), $RSA->decrypt($k[0]['n']));

	echo "\t<tr onclick=\"location.href='?id={$row['client_id']}'\">".PHP_EOL;
	echo "\t\t<td>".$row['client_id']."</td>".PHP_EOL;
	echo "\t\t<td>".$RSA->raw($row['fio'])."</td>".PHP_EOL;
	echo "\t</tr>".PHP_EOL;
}
echo "</table>";
echo "<a href=\"new.php\">Новый</a>";
```

ДОДАТОК В

Програмна система попередньої обробки та шифрування даних.

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51165_19Б 14-1

Аркушів 8

Київ 2019

АНОТАЦІЯ

Розділ містить опис частини, яка слугує для генерації ключів та шифрування-дешифрування тексту, що є структурною одиницею програмного продукту, та забезпечує поєднання можливостей усіх інших модулів для виконання поставлених перед системою завдань. Модуль надає методи для роботи зі звичайним текстом та даними у бінарному вигляді. Модуль написано мовою програмування PHP.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У додатку розглядається один з програмних модулів системи – модуль шифрування з кодом УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТВ51165_19Б 12-1, що міститься у файлі RSA.php. Модуль призначений для перетворення звичайного тексту у шифротекст, розшифрування тексту та генерації ключів. Модуль використовується в клієнтській та серверній частині після попередньої обробки даних. Для роботи потребує спочатку згенерувати ключі, або передати їх в якості параметрів.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Призначенням модулю є шифрування даних для подальшого зберігання або обміну між клієнтом та сервером. Також модуль містить методи для перетворення символів юнікоду до їх числових кодів та навпаки. Використання такого модулю дозволяє створювати програмне забезпечення, де інтерфейс і логіка роботи є незалежними компонентами, що дає можливість використовувати його для зменшення навантаження на клієнтську частину системи.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Модуль реалізує алгоритм RSA, для спрощення операції піднесення багатозначних чисел до багатозначного степеня та знаходження остачі від ділення на багатозначне число використовується прийом знаходження остачі після кожного піднесення до степеня. Модуль надає методи шифрування-дешифрування числа, рядка тексту, методи піднесення до степеня, генерації випадкових простих чисел, перевірки чи є два числа взаємно простими. Більшість методів використовуються всередині модулю, адже вони є допоміжними, але головні забезпечують функціонування всіх інших частин системи.

ТЕХНІЧНІ ЗАСОБИ ЩО ВИКОРИСТОВУЮТЬСЯ

Модуль розроблено у інтегрованому середовищі розробки phpSandstorm, що забезпечує набір сервісних функцій та графічний діалог з користувачем, на комп’ютері, що використовував операційну систему Windows 7.

Для запуску PHP коду потрібен інтерпретатор, який був підключен до веб-серверу Apache.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Програмний модуль реалізований як окремий клас, який забезпечує існування клієнтської частини та бізнес-логіки окремо від одного, але разом із цим запуск обох компонентів відбувається одночасно.

Для використання даного модулю не потрібно ніяких дій, оскільки він автоматично спрацьовує після запуску серверного додатку.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними для модуля є інформація, яку користувач вводить в додатку, або ця ж інформація в зашифрованому вигляді.

Вихідними даними програмного модуля є або шифротекст, якщо передавались дані, або розшифрований текст, якщо передати зашифровані дані.

ДОДАТОК Г

Програмна система попередньої обробки та шифрування даних.

Свідоцтво про реєстрацію авторського права

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51165_19Б 15-1

Аркушів 1

Київ 2019