МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерної алгебри та дискретної математики

(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

на здобуття ступеня вищої освіти «бакалавр»

(освітньо-кваліфікаційний рівень)

на тему

«Верифікація даних через CMC/QR/мобільний додаток з застосуванням цифрового підпису»

«Data verification via CMS / QR / mobile application using digital signature»

Виконав: студент денної форми навчання
напряму підготовки 123 – Комп'ютерна інженерія.

(шифр і назва напряму підготовки, спеціальності)

Домброван Євгеній Віталійович

(прізвище, ім'я, по-батькові)

Керівник к. ф.-м. н., доцент Савастру О.В

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

Рецензент к. ф.-м. н., доцент Варбанець С.П.

(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№6 від «11» червня 2021 р.

Завідувач кафедри

_____ Варбанець П.Д.
(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК № ___
протокол № __ від «___» ____ 2021 р.
Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____ Казакова Н.Ф.
(підпис) (прізвище, ініціали)

Одеса - 2021

АННОТАЦІЯ

Мета дипломної роботи – розробити систему верифікації даних через SMS/QR/мобільний додаток за допомогою цифрового підпису.

Проведено аналіз існуючих рішень, які застосовуються для рішення поточної або схожих проблем. Розроблено алгоритм верифікації даних за допомогою цифрового підпису. Побудована модель системи для підтримки алгоритму верифікації даних.

Програмно реалізований алгоритм верифікації даних за допомогою цифрового підпису. Програмно реалізована система підтримки алгоритму.

АННОТАЦИЯ

Цель дипломной работы - разработать систему верификации данных через SMS / QR / мобильное приложение с помощью цифровой подписи.

Проведен анализ существующих решений, которые применяются для решения текущей или похожих проблем. Разработан алгоритм верификации данных с помощью цифровой подписи. Построена модель системы для поддержки алгоритма верификации данных.

Программно реализован алгоритм верификации данных с помощью цифровой подписи. Программно реализована система поддержки алгоритма.

ABSTRACT

The purpose of the thesis is to develop a data verification system via SMS / QR / mobile application with a digital signature.

Analyzed existing solutions to solve current or similar problems. An algorithm for verifying data using a digital signature has been developed. A system model is built to support the data verification algorithm.

Created software for data verification algorithm using digital signature. Created software to support algorithm-based system.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

ЕЦП – Електронний цифровий підпис

QR код – матричний код для легкого розпізнавання сканувальним обладнанням

RSA – криптографічний алгоритм з відкритим ключем, що базується на обчислювальній складності задачі факторизації великих цілих чисел

Верифікація – підтвердження правдивості, істинності даних

Фреймворк – це інфраструктура програмних рішень, що полегшує розробку складних систем

Трекінг – відстежування певних дій

Бібліотека – це збірка об'єктів чи підпрограм для вирішення близьких за тематикою задач

ВСТУП

На сьогоднішній день глобалізація призвела до постійного зростання логістики. Перевезення і доставка зачіпають всі сфери: промисловість, малий і середній бізнес і навіть особисте життя. Кожен підприємець рано чи пізно стикається з потребою доставки або перевезення товарів. Саме тому трекінг перевезень відіграє важливу роль у розвитку будь-якого бізнесу: потрібно знати, що товар прибув в належній якості і кількості, був оплачений і що процес пройшов без екстрених ситуацій. Якщо великий бізнес і може собі дозволити важкі системи для контролю за все і відразу, дрібні підприємці не в змозі оплатити штат розробників для створення такої системи. А при відсутності системи контролю виникає безліч проблемних моментів:

- обробка екстрених випадків - наприклад, поломка автомобіля або бій товару:
- контроль водія - його початок роботи, кінець роботи;
- контроль оплати і виробництво документів - часто це слабо відстежується і ніяк не перевіряється, все відбувається "в ручному режимі";

Ще важливий аспект, який потрібно брати до уваги - це відсутність зручної комунікації між усіма учасниками процесу: клієнтом, водієм, менеджером, іншими можливими учасниками системи.

Так як питання трекінгу дійсно дуже важливе, за довгий час встигло з'явитися безліч рішень. Велика частина з них - це приватні рішення великих компаній. Наприклад, в Україні це Нова Пошта, Розетка, Метро, UTS, OLX і багато інших. Ці компанії не використовують якісь стандартні рішення, для них розробляються приватні. Хоча це добре працює для великих компаній, бізнес поменше собі дозволити таку розкіш не може.

Друга група - це окремі трекінгові GPS-рішення, що дозволяють стежити за автомобілями і їх станом, а також зв'язуватися з менеджером. Такими прикладами можуть служити Logist.FM або Fixon. Великий недолік цих

рішень полягає в їх однобокості. Для застосування потрібно забезпечити машину за потрібне пристроєм, який буде жорстко прив'язана до однієї компанії.

Але найважливішим недоліком систем другої групи є відсутність універсального та зручного рішення верифікації даних. Логістичні системи різних рівнів користуються паперовими документами, накладними, наказами. Ті системи, що все ж-таки впроваджують ту чи іншу ступінь цифрових рішень, користуються незручними, нестандартизованими додатками. Впровадження нових гілок системи або взаємодія двох різних систем виявляється майже неможливою.

Безпека таких систем також виявляється під питанням. Ланки в логістичних системах нерідко користуються усними або нефіксованими засобами комунікації, що призводить до втрати даних або до збоїв у робочому процесі. Учасники системи застосовують різні пристрої або технології, що також заважає процесу комунікації.

Саме через актуальність цих проблем побудова якісного, зручного та ефективного алгоритму верифікації даних є дуже важливою. Стандартизований підхід до побудови системи дозволить полегшити процеси, що контролюються не тільки трекінговими системами, а взагалі – системами, де є процес, що вимагає координації віддалених один від одного об’єктів, що використовують різні методи комунікації.

Для верифікації даних у багатьох країнах використовується ЕЦП. Впровадження надійного рішення на основі цифрового підпису вирішить проблему безпеки у системах, що будуть використовувати рішення. При доповненні даного рішення вдалою системою підтримки та універсальним алгоритмом дозволить вирішити більшість проблем сучасних логістичних процесів.

Запит на вирішення даної проблеми надійшов із практичної сфери – про незручності в роботі та неточності в комунікації повідомляють менеджери

різних рівнів у логістичних компаніях різного рівня та представники малого та середнього бізнесу не тільки в Україні, а й у всьому світі.

Рішення для даної проблеми може бути доцільним не тільки для логістичних систем. Принцип вирішення задачі верифікації даних, що отримуються допомогою різноманітних засобів комунікації, залишається незмінним і для інших задач, тому що ЕЦП є універсальним та визнаним на державному рівні.

1 ПОСТАНОВКА ЗАДАЧІ

Мета роботи – розробити та реалізувати алгоритм верифікації даних, що надходять з різних джерел комунікації, за допомогою ЕЦП, та побудувати систему для підтримки даного алгоритму. Програмно реалізувати дану систему.

Для досягнення поставленої цілі необхідно вирішити наступні підзадачі:

1) провести аналіз предметної області;
2) розглянути аналоги, якщо такі існують;
3) створити модель алгоритму верифікації;
4) створити модель системи;
5) обрати інструменти розробки;
6) реалізувати алгоритм та систему верифікації.

2 ОГЛЯД АНАЛОГІВ ТА ІСНУЮЧИХ РІШЕНЬ

Перша група рішень – це великі корпоративні системи, що створюються на замовлення для внутрішнього використання однією компанією. Наприклад, Нова Пошта використовує для верифікації користувачів смс та сканування додатку. Для трекінгу логістичної системи Нова Пошта впроваджує власні рішення, включаючи складну систему координації, що складається з великої кількості датчиків. Незважаючи на високий рівень автоматизації, всі документи, що стосуються доставників, водіїв та кур'єрів, існують лише у паперовому вигляді.

Іншим прикладом є компанія Danone. Danone – продуктова компанія, що так само, як і Нова Пошта, має свою інфраструктуру. Проте, часто перевізники наймаються додатково, і компанія співпрацює з великою кількістю найманих працівників. Такі робочі умови сильно ускладнюють процес, в основному через те, що виникають проблеми з документами та паперами. В них відсутня система електронної верифікації, що призводить або до часових витрат та проблем з паперами, або, якщо зневажати паперовою роботою, до великих проблем.

Друга група рішень – це онлайн-рішення [3] для верифікації документів. Дані рішення можуть бути і надійними, і безпечними, але в них бракує інтегративності. Вони працюють лише на сайті локально, не надаючи змоги підключатися стороннім сервісам. Дані системи також не допускають розширення або створення кастомних рішень для потреб користувачів.

3 ОБГРУНТУВАННЯ ПРОЕКТНОГО РІШЕННЯ

Зазначено переваги розробки такої системи. Найважливішими з них є зручний інтерфейс для менеджера або адміністратора, що дозволяє швидко маніпулювати даними, масштабованість, що дозволяє стороннім сервісам та системам підключатись до створюваної системи, та універсальність у використанні. Система передбачає покриття всіх базових потреб користувачів: менеджерів, клієнтів, працівників. За потреби кількість ролей може бути легко розширена.

Ще одним фактором створення даної системи стала її доступність у використанні. Система не буде потребувати великих ресурсів для початку роботи, в процесі використання системи споживання ресурсів буде зростати пропорційно навантаженню.

4 ЕЛЕКТРОННИЙ ЦИФРОВИЙ ПІДПИС

Електронно цифровий підпис (ЕЦП)[5] — вид електронного підпису, отриманого за результатом криптографічного перетворення набору електронних даних, який додається до цього набору або логічно з ним поєднується і дає змогу підтвердити його цілісність та ідентифікувати підписувача. ЕЦП накладається за допомогою приватного ключа та перевіряється за допомогою відкритого ключа.

Одним із елементів обов'язкового реквізиту є електронний підпис, який використовується для аутентифікації автора та/або підписувача електронного документа іншими суб'єктами електронного документообігу. Оригіналом електронного документа вважається електронний примірник з ЕЦП автора.

ЕЦП призначений для використання фізичними та юридичними особами — суб'єктами електронного документообігу для аутентифікації підписувача або для підтвердження цілісності даних в електронній формі.

ЕЦП як спосіб ідентифікації підписувача електронного документа, дозволяє однозначно визначати джерело інформації, що міститься у документі.

Накладання ЕЦП завершує утворення електронного документа, надаючи йому юридичної сили. Згідно закону України «Про електронні документи та електронний документообіг»[4] юридична сила електронного документа з нанесеними одним або множинними ЕЦП та допустимість такого документа як доказу не може заперечуватися виключно на підставі того, що він має електронну форму.

ЕЦП накладається за допомогою особистого ключа та перевіряється за допомогою відкритого ключа. За правовим статусом він прирівнюється до власноручного підпису або печатки. Електронний підпис не може бути визнаний недійсним лише через те, що він має електронну форму або не ґрунтується на посиленому сертифікаті ключа.

За умови нерозголошення власником приватного ключа його підробка неможлива. Електронний документ також не можливо підробити: будь-які зміни, не санкціоновано внесені в текст документа, будуть миттєво виявлені.

Особистий ключ ЕЦП формується на підставі абсолютно випадкових чисел, що генеруються давачем випадкових чисел, а відкритий ключ обчислюється з особистого ключа ЕЦП так, щоб одержати другий з першого було неможливо.

Особистий ключ ЕЦП є унікальною послідовністю символів довжиною 264 біти, яка призначена для створення ЕЦП в електронних документах. Працює особистий ключ тільки в парі з відкритим ключем. Особистий ключ необхідно зберігати в таємниці, адже будь-хто, хто дізнається його, зможе підробити ЕЦП.

Документ підписується ЕЦП за допомогою особистого ключа ЕЦП, який існує в одному екземплярі тільки у його власника. Цьому особистому ключу відповідає відкритий ключ, за допомогою якого можна перевірити відповідність ЕЦП його власнику.

Відкритий ключ використовується для перевірки ЕЦП одержуваних документів. Відкритий ключ працює тільки в парі з особистим ключем. Відкритий ключ міститься в сертифікаті відкритого ключа, і підтверджує приналежність відкритого ключа ЕЦП певній особі. Крім самого відкритого ключа, сертифікат відкритого ключа містить в собі персональну інформацію про його власника (ім'я, реквізити), унікальний реєстраційний номер, термін дії сертифікату відкритого ключа. З метою забезпечення цілісності представлених у сертифікаті даних він підписується особистим ключем центру сертифікації ключів.

Ілюстрація цифрового підпису даних на рисунку 4.1 – шифрування, 4.2 – верифікація.

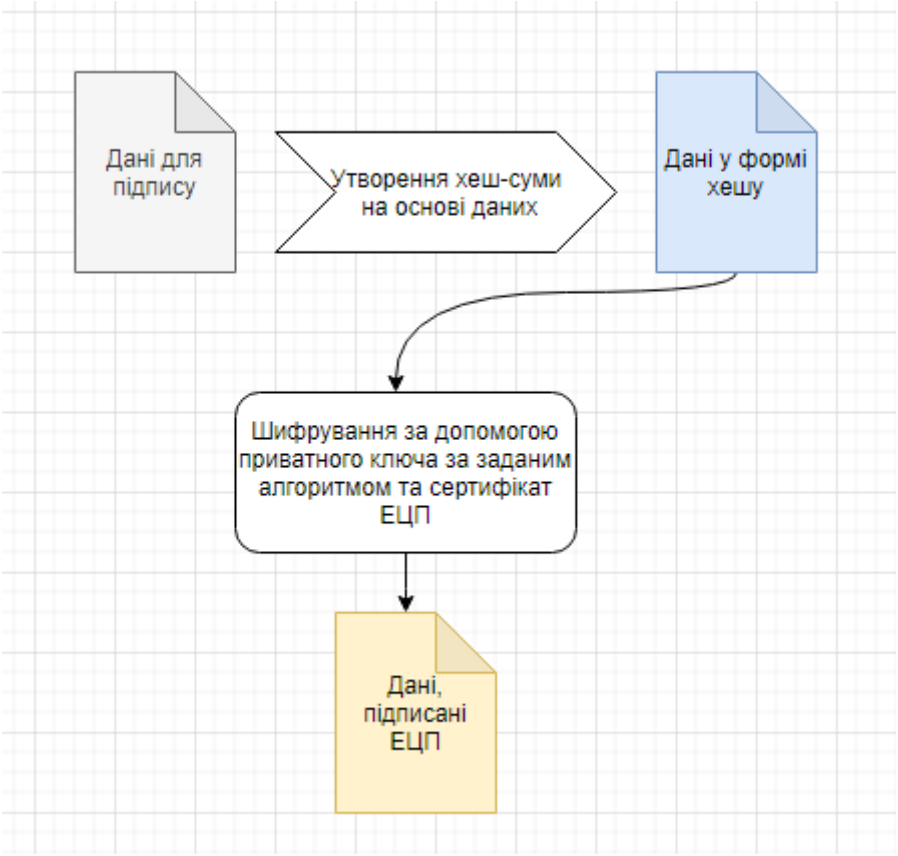

Рисунок 4.1 – Підписання даних за допомогою ЕЦП

Рисунок 4.2 – Верифікація даних за допомогою ЕЦП

При підписанні електронного документа його початковий зміст не змінюється, а додається блок даних, так званий «ЕЦП».

Отримання цього блоку відбувається у два етапи. На першому етапі за допомогою програмного забезпечення і спеціальної математичної функції обчислюється так званий «відбиток повідомлення».

Цей відбиток має такі властивості:

- фіксовану довжину, незалежно від довжини повідомлення;
- унікальність відбитку для кожного повідомлення;
- неможливість відновлення повідомлення за його відбитком.

Таким чином, якщо документ був модифікований, то зміниться і його відбиток, що відобразиться при перевірці ЕЦП.

На другому етапі відбиток документа шифрується за допомогою програмного забезпечення і особистого ключа автора.

Розшифрувати ЕЦП і одержати початковий відбиток, який відповідатиме документу, можна тільки використовуючи сертифікат відкритого ключа автора. Таким чином, обчислення відбитку документа захищає його від модифікації сторонніми особами після підписання, а шифрування особистим ключем автора підтверджує авторство документа.

Перевірка ЕЦП одержаного документа проводиться декількома етапами:

1) адресат за допомогою програмного забезпечення Сертифікатом відкритого ключа автора розшифровує підписаний відбиток і одержує відбиток початкового документа;
2) за допомогою програмного забезпечення і спеціальної математичної функції з документа, який був одержаний, обчислюється його відбиток;
3) при перевірці ЕЦП порівнюються відбитки початкового і одержаного документів. Результат перевірки — одна з відповідей: «вірний»/«невірний».

Криптосистема RSA[6] належить до числа перших криптосистем з відкритим ключем з підтримкою ЕЦП. Одним із алгоритмів, які добре

працюють як при шифруванні, так і для цифрового підпису є алгоритм RSA. Він досі є найпростішим в реалізації поміж алгоритмів з відкритими ключами і успішно протистоїть активному криптоаналізу. Криптографічні системи з відкритим ключем використовують так звані необоротні функції, які володіють наступними властивостями: Якщо відомо x, то f(x) обчислити відносно просто. Якщо відомо, для обчислення x немає простого (ефективного) шляху. Під однонаправленістю мається на увазі не теоретична однонаправленість, а практична неможливість обчислити обернене значення, використовуючи сучасні обчислювальні засоби, за прийнятний проміжок часу. В основу RSA покладено задачу добутку двох великих простих чисел і розкладання складних чисел на прості множники, яка є однонаправленою обчислювальною задачею. Важливим також є той факт, що відкриті ключі часто використовуються для довготривалого безпечного обміну та збереження важливої інформації. Безпека алгоритму RSA побудована на принципі складності факторизації, тобто вона повністю залежить від проблеми розкладання на множники великих чисел.

У широкому вжитку також знаходяться криптосистеми DSA та ECDSA.

В основі криптосистеми DSA лежить Схема Ель-Гамаля[2] - асиметричні алгоритми шифрування, заснована на труднощі обчислення дискретних логарифмів в кінцевому полі. Криптосистема включає в себе алгоритм шифрування і алгоритм цифрового підпису. Ідея схеми заснована на тому, що для обґрунтування практичної неможливості фальсифікації цифрового підпису може бути використана більш складна обчислювальна задача, ніж розкладання на множники великого цілого числа, - завдання дискретного логарифмування.

Нехай є абоненти A, B, C, ..., які хочуть передавати один одному зашифровані повідомлення, не маючи ніяких захищених каналів зв'язку. Фактично використовується схема, аналогічна до Діффі-Хеллмана, щоб сформувати загальний секретний ключ для двох абонентів, які передають одне одному повідомлення, і потім повідомлення шифрується шляхом множення

його на цей ключ. Для кожного наступного повідомлення секретний ключ обчислюється заново.

ЕЦП підтверджує достовірність і цілісність документа. Якщо в документ в процесі пересилки були внесені які-небудь зміни, нехай навіть зовсім незначні, то підміна виявиться. Сертифікат відкритого ключа містить персональну інформацію про власника, що дозволяє однозначно ідентифікувати автора документа.

Однією з додаткових можливостей при роботі з ЕЦП є послуга фіксації точного часу підписання документа. Відмітка точного часу при підписанні документа дозволяє точно ідентифікувати момент накладання підпису, причому змінити його значення згодом, навіть особою, яка наклала підпис, неможливо. Можливе лише повторне підписання з фіксацією нового часу. Точне значення часу, який використовується для формування відмітки точного часу, здійснюється апаратними засобами Центру сертифікації ключів шляхом синхронізації з джерелами точного часу з точністю до 1 секунди.

5 ПОБУДОВА МОДЕЛІ СИСТЕМИ ВЕРИФІКАЦІЇ

Першим кроком у розробці системи верифікації є побудова структури взаємодії учасників. Користувачі системи передбачаються різні та неоднорідного типу. Для задовільнення універсальності система верифікації повинна враховувати як можливість максимально примітивного спілкування, наприклад, наявність одного працівника та наявність одного клієнта, так і заплутані відносини між менеджерами, клієнтами та працівниками. Незалежно від даних умов, система повинна справлятися та контролювати хід роботи однаково. Найпростіший випадок представлено на рисунку 5.1.

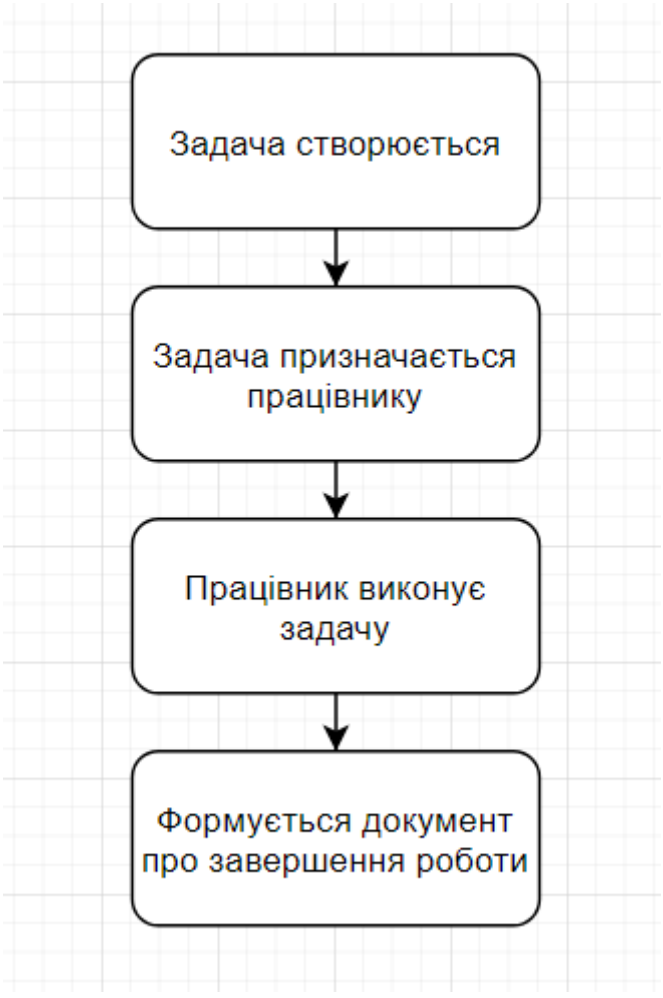

Рисунок 5.1 – Базова модель системи

Дана система взаємодії є найпростішою, але ненадійною на практиці. Вона не обробляє екстрені випадки, наприклад, проблеми в роботі, не відслідковує можливих затримок, про які замовник повинен бути сповіщений. Дану систему потрібно модифікувати, наприклад, як на рисунку 5.2.

Рисунок 5.2 – Друга версія схеми з урахуванням сповіщень

Дана схема комунікації допомагає відстежити у хронологічному порядку роботу та сповіщати учасників системи про актуальний стан процесу. Незважаючи на покращення, дана схема бракує важливого аспекту: захисту даних. Потрібно постійно перевіряти особистості, стежити за жорстким порядком виконання процесу. Дана схема покращена на рисунку 5.3.

Рисунок 5.3 – Схема з урахуванням формування ЕЦП

Дана схема вже є повноцінним варіантом взаємодії учасників. Вона задовольняє наступні пункти:

1) кожен учасник повідомляється про зміну стану задачі або процесу;
2) завершення процесу неможливе без згоди всіх задіяних сторін;
3) всі етапи процесу захищені особистими кодами безпеки.

Такий підхід дозволяє перейти до наступного кроку побудови моделі.

Система верифікації орієнтована на гнучкість та зручність у використанні. Для того, щоб вона могла вільно масштабуватися, система повинна враховувати різноманітні види та підвиди задач. Для досягнення цієї цілі виділені певні критерії, що притаманні всім видам задач:

1) Цілісність етапів

Жодна система не допускає ігнорування етапів робочого процесу. Наприклад, якщо замовлення ще не прийняте працівником у роботу, клієнт не може підтвердити його виконання.

2) Чітке розподілення обов’язків

Якщо виконання певного етапу призначено лише окремій ролі, наприклад, працівнику або менеджеру, дану роль не може виконати замовник. Якщо один працівник взявся до роботи, то без спеціального наказу від менеджера він не може передати роботу іншому працівнику.

3) Незалежність від учасників системи

Робота повинна вестись лише за урахуванням властивостей учасників системи, а не прив’язуватись до конкретних об’єктів. Клієнту недоступна інформація про працівника, навіть якщо спілкування не керується за участю менеджера, так само як і працівнику недоступні дані про клієнта. Процес проходить лише з орієнтирами на статуси.

4) Незалежність від компонентів системи

Розроблювана система верифікації даних матиме можливість інтегруватися с численною кількістю різних засобів комунікації. Це можуть бути мобільні телефони з додатками, дзвінки за GSM-зв’язком, програми для зчитування QR-кодів, тощо.

Всі ці методи передачі та отримання даних не повинні змінювати внутрішню структуру проекту та не можуть впливати на алгоритми верифікації переданих даних. Система може бути універсальною для всіх методів комунікації, але алгоритм повинен залишатися незмінним.

Враховуючи всі критерії, яким повинна задовольняти система верифікації, побудовано наступний алгоритм:

1) при реєстрації процесу на нього підписуються всі учасники;
2) процес розподіляється на етапи;
3) кожному етапу присвоюється статус;
4) кожен з учасників стає відповідальний за один або декілька етапів;
5) кожна зміна етапу може статися лише за підтвердженням всіх учасників, відповідальних за даний етап;
6) з кожною зміною статусу процесу кожен учасник, відповідальний за поточний етап, отримує сповіщення про його успішне завершення, або за його невдале завершення з описом помилки, якщо це потребується умовами. Кожен учасник процесу, відповідальний за наступний етап, отримує короткий код верифікації;
7) при завершені своєї роботи на етапі, учасник відправляє код на сервер, що обслуговує процес за допомогою зручного методу зв’язку;
8) на сервері код обробляється наступним чином:

– код зберігається у об’єкті, відповідальному за поточний процес;
– статус поточного процесу додається до поточного коду процесу;
– формується хеш поточних даних на основі коду та статусу;
– код, що був присланий на сервер, додається до статусу, відповідного статусу учасника, що його прислав;
– якщо учасник має інший статус, або ж він прислав невірний код, хеш не співпадає;
– якщо хеш співпадає, виконується перехід на наступний рівень статусу;

9) коли всі етапи процесу пройшли успішно, на виході формується ЕЦП документу про завершення роботи.

Даний алгоритм дозволяє ефективно керувати верифікацією даних у робочому процесі. Схематичну роботу алгоритму представлено на рисунку 5.4

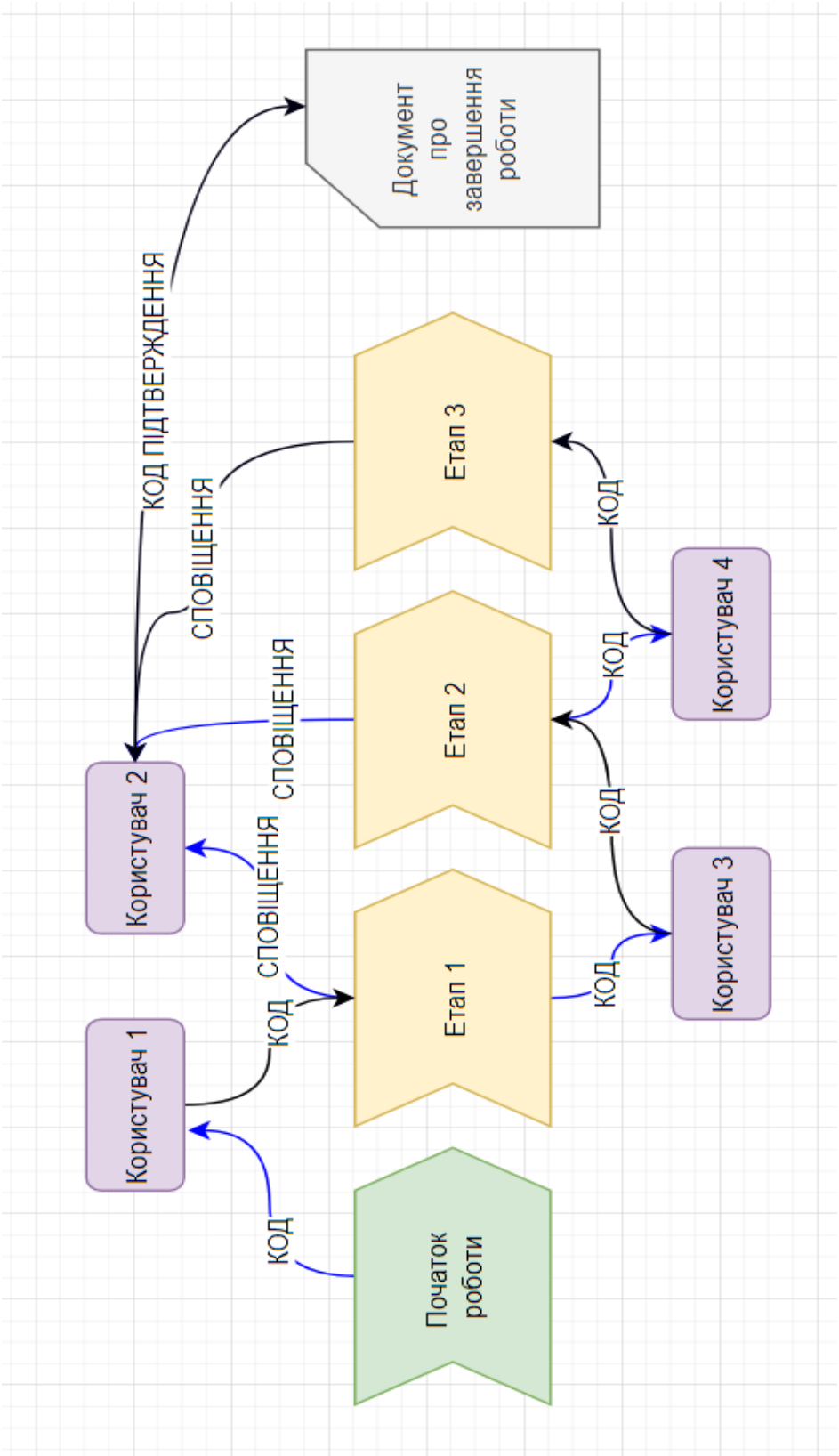

Рисунок 5.4 – Схематична робота алгоритму верифікації

6 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Після того, як побудовано алгоритм верифікації, на основі критеріїв, визначених алгоритмом, будується система підтримки процесу верифікації.

Окрім критеріїв, зазначених у алгоритмі, система повинна забезпечувати наступні функції:

1) зручний інтерфейс користувача (менеджера);
2) можливість підключати різноманітні сервіси для отримання даних;
3) формування процесу;
4) додавання учасників до процесу;
5) обробка даних з джерел, обраних користувачами (QR, SMS, тощо);
6) проведення процесу від початку до завершення;
7) перевірка процесу від початку до кінця;
8) можливість згенерувати документ;

Першим чином виділені дані, що будуть зберігатися у базі даних. Це інформація про:

1) клієнта;
2) працівника;
3) процес.

Для клієнта та працівника потрібно зберігати інформацію про конфігурацію, тобто їх уподобання щодо отримання та відправлення повідомлень. За урахуванням усіх вимог, отримана наступна система (рис. 6.1):

- Таблиця NotificationSettings зберігає налаштування клієнтів та працівників щодо способу отримання даних користувачами.
- Таблиця Contact зберігає контактні дані користувачів.
- Таблиця Task відображає кожен крок, що відбувався в процесі, наприклад, зміни статусу

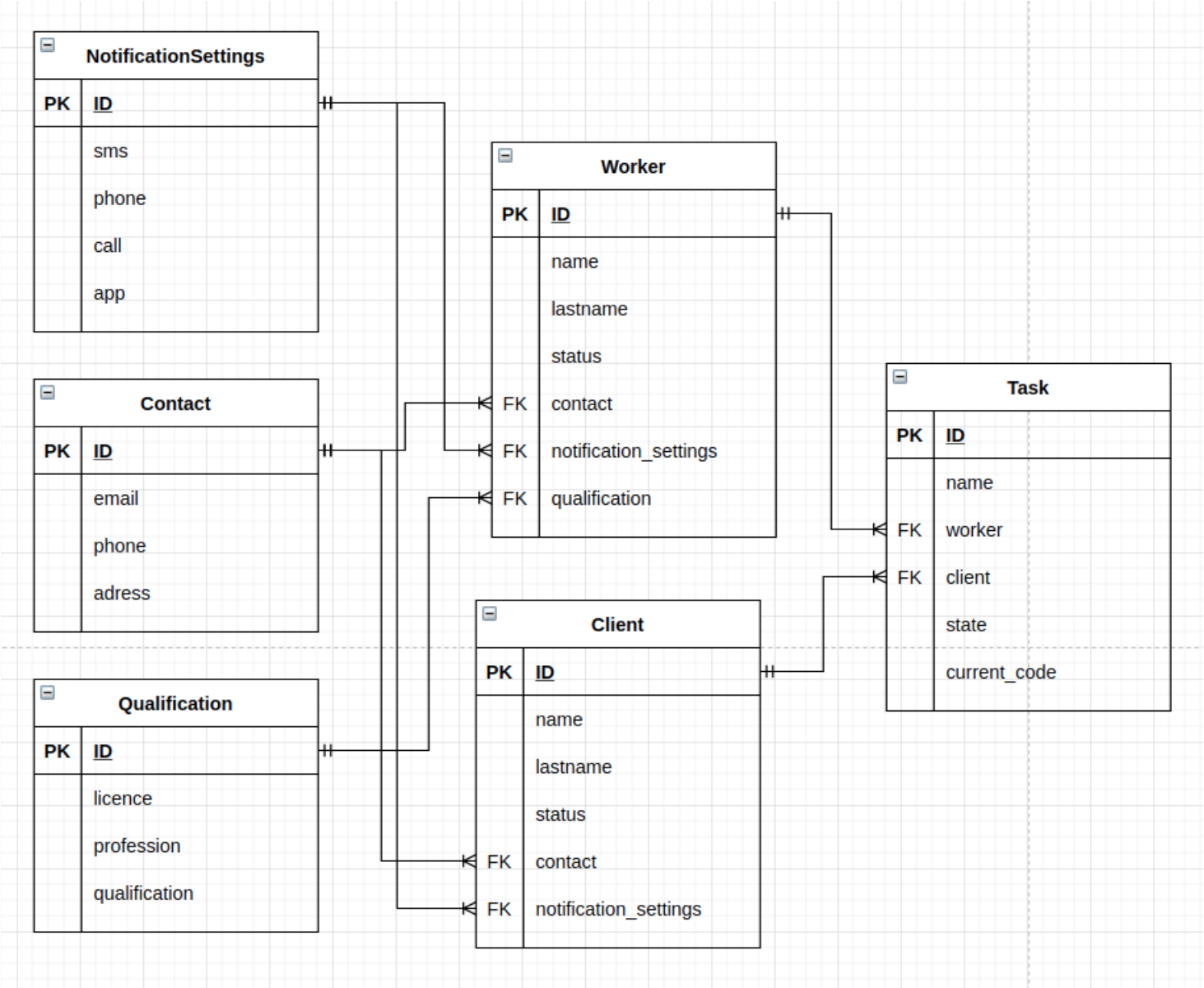

Рисунок 6.1 - Модель бази даних для системи підтримки

7 ВИБІР ІНСТРУМЕНТІВ РЕАЛІЗАЦІЇ

Для реалізації програми обрані наступні інструменти:

Мова програмування - Python 3. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Серед даних пакетів та модулів можна знайти готові рішення, такі як, наприклад, бібліотека cryptography. Даний пакет включає як рішення високого рівня, так і інтерфейси низького рівня до загальних криптографічних алгоритмів, таких як симетричні шифри, дайджести повідомлень та ключові функції виведення.

Бібліотека в цілому поділяється на два рівні. Той, що має безпечні криптографічні рішення, які практично не вимагають вибору конфігурації. Вони безпечні та прості у використанні і не вимагають від розробників приймати багато рішень.

Інший рівень - криптографічні примітиви низького рівня. Вони часто небезпечні і можуть бути використані неправильно. Вони вимагають прийняття рішень та глибокого знання криптографічних концепцій у роботі. Через потенційну небезпеку при роботі на цьому рівні це називають шаром "небезпечних матеріалів" або "hazmat". Вони знаходяться у пакеті cryptography.hazmat. В даній роботі буде використовуватися другий рівень - глибокого доступу.

Середовище розробки - PyCharm 2020.3;

Фреймворк Django. Django - це високорівневий Python веб-фреймворк, який дозволяє швидко створювати безпечні і підтримувані веб-сайти. Створений досвідченими розробниками, Django бере на себе більшу частину турбот веб-розробки, тому можна зосередитися на написанні свого веб-додатки без необхідності винаходити велосипед. Він безкоштовний і з

відкритим вихідним кодом, має зростаючу і активну спільнота, відмінну документацію і безліч варіантів як безкоштовної, так і платної підтримки.

Django слідує філософії «Все в одному» і надає майже все, що розробники можуть захотіти зробити «з коробки». Оскільки все, що потрібно, є частиною єдиного «продукту», все це бездоганно працює разом, відповідає послідовним принципам проектування і має велику і актуальну документацію.

Ще однією з переваг Django є його система безпеки. Django допомагає розробникам уникнути багатьох поширених помилок безпеки, надаючи фреймворк, розроблений для автоматичного захисту сайту. Наприклад, Django надає безпечний спосіб керування обліковими записами користувачів і паролями, уникаючи поширених помилок, таких як розміщення інформації про сеанс в файли cookie, де вона вразлива (замість цього файли cookie містять тільки ключ, а фактичні дані зберігаються в базі даних) або безпосереднє зберігання паролів замість хеша пароля.

Django, за замовчуванням, забезпечує захист від багатьох вразливостей, включаючи SQL-ін'єкцію, міжсайтовий скриптинг, підробку міжсайтових запитів і клікджекінг.

Також важливим фактором є масштабованість. Django використовує компонентну "shared-nothing" архітектуру (кожна її частина незалежна від інших і, отже, може бути замінена або змінена, якщо це необхідно). Чіткий поділ частин означає, що Django може масштабуватися при збільшенні трафіку, шляхом додавання обладнання на будь-якому рівні: сервери кешування, сервери баз даних або сервери додатків.

Для якісного масштабування фреймворку була обрана REST-архітектура.

REST-підхід – це набір правил, що дозволяють створити максимально універсальні та відкриті веб-додатки. Два головних принципи створення додатків в стилі REST:

- сервер не повинен нічого знати про поточний стан Клієнта. У запиті від Клієнта повинна бути вся необхідна інформація для обробки цього запиту сервером;
- кожен ресурс на Сервері повинен мати певний Id, а також унікальний URL, за яким здійснюється доступ до цього ресурсу.

На даний момент ми можемо знайти фреймворк для створення додатків в стилі REST практично для кожної мови програмування, що використовується в веб-розробці. Логіка побудови Web API на Сервері в цих фреймворками реалізована однаково. Django не є виключенням. Для нього так само є фреймворк, що додає можливість побудови REST-архітектури.

Django Rest Framework (DRF) - це бібліотека, яка працює зі стандартними моделями Django для створення гнучкого і потужного API для проекту.

Основні переваги, що виділяють DRF серед інших можливих рішень:

1) веб-переглядний API - це величезна перемога у зручності для розробників;
2) політики аутентифікації, включаючи пакети для OAuth1a та OAuth2;
3) серіалізація, яка підтримує як джерела даних ORM, так і не ORM;
4) настроюється повністю - використовуються звичайні подання на основі функцій, якщо не потрібні більш потужні функції;
5) велика документація та велика підтримка спільноти розробників;
6) використовується та завоював довіру міжнародно визнаних компаній, включаючи Mozilla, Red Hat, Heroku та Eventbrite.

СУБД - PostgreSQL. Такий вибір пов'язаний з рядом переваг PostgreSQL перед іншими СУБД:

— PostgreSQL не просто реляційна, а об'єктно-реляційна СУБД. Це дає їй вагомі переваги над іншими SQL базами даних з відкритим кодом;

— PostgreSQL надає надійні вбудовані оператори та функції. Крім того, він дозволяє створювати власні оператори і функції (включаючи агрегати), а також процедури, що і тригери;

— для PostgreSQL добре опрацьованості документація, що дозволяє вирішувати виникаючі проблеми з деякими зручністю;

— PostgreSQL - безкоштовна СУБД з відкритим кодом.

8 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ВЕРИФІКАЦІЇ

Після того, як обрано інструменти розробки, можна будувати систему за схемою, визначеною у Розділах 5 та 6.

Першим кроком є написання системи підтримки для алгоритму верифікації. Для цього створено веб-додаток Django та підключено базу даних PostgreSQL.

Створення деяких моделей та сервісів:

```
class Task(models.Model):
    name = models.CharField(max_length=100)
    worker = models.ForeignKey(Worker, on_delete=models.CASCADE,
 related_name='worker')
    client = models.ForeignKey(Client, on_delete=models.CASCADE,
 related_name='client')
    state = models.CharField(max_length=50, default='initialized
')
    document = models.ForeignKey(Document, on_delete=models.CASC
ADE, null=True, blank=True, default=None)
    current_code = models.CharField(max_length=4, default='0000'
)

    class Meta:
        db_table = 'tasks'

    def __str__(self):
        return self.name

    def is_approved(self) -> bool:
        return self.state == 'approved'

    def check_hash(self, code: str) -> bool:
        print('hashes')
        print(self.__hash__(), hash((code, self.state)))
        return self.__hash__() == hash((code, self.state))

    def __hash__(self):
        print(f'current code: {self.current_code}')
        print(f'in hash: {hash((self.current_code, self.state))}
')
        return hash((self.current_code, self.state))

    def generate_code(self) -> str:
```

```
        self.current_code = str(random.randint(1000, 9999))
        print(f'current code {self.current_code}')
        return self.current_code

def send_notification(recipient: Union[Worker, Client], task: Ta
sk, code: bool = False) -> None:
    settings = recipient.notification_settings
    content = f'{task.name}'
    if code:
        content += f', code for next step: {task.current_code}'
    contacts = recipient.contact

    if settings.receive_call:
        send_call(contacts.phone, content)
    if settings.receive_email:
        send_email(contacts.email, content)
    if settings.receive_sms:
        send_sms(contacts.phone, content)
    if settings.receive_app_notification:
        send_in_app(recipient.id, content)
```

Другим кроком є додання масштабованості системі. Для цих цілей використовується фреймворк DRF. Вже створені моделі та сервіси підключуються в ендпоінти.

Створення деяких ендпоінтів:

```
class WorkerStartView(views.APIView):
    class InputSerializer(serializers.Serializer):
        code = serializers.CharField()

    def post(self, request, pk=None):
        serializer = self.InputSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        data = serializer.data
        task = Task.objects.get(id=pk)
        worker = task.worker
```

```
        if task.check_hash(data['code']):
            send_notification(worker, task, True)
            send_notification(task.client, task)
            _ = task.generate_code()
        else:
            return Response(status=status.HTTP_400_BAD_REQUEST)
        if report_work_start(task):
            return Response(status=status.HTTP_200_OK)
        else:
            return Response(status=status.HTTP_400_BAD_REQUEST)
```

Завершальним кроком у побудові системи верифікації є реалізація алгоритму. Для цього використана бібліотека cryptography.

```
def generate_private_key():
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
        backend=default_backend()
    )
    return private_key

def sign_document(content, private_key):
    signature = private_key.sign(
        content,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )
    return signature
```

Решту програмного коду системи можна знайти у Додатку А.

Результат роботи програми можна побачити на рис. 7.1 – 7.7.

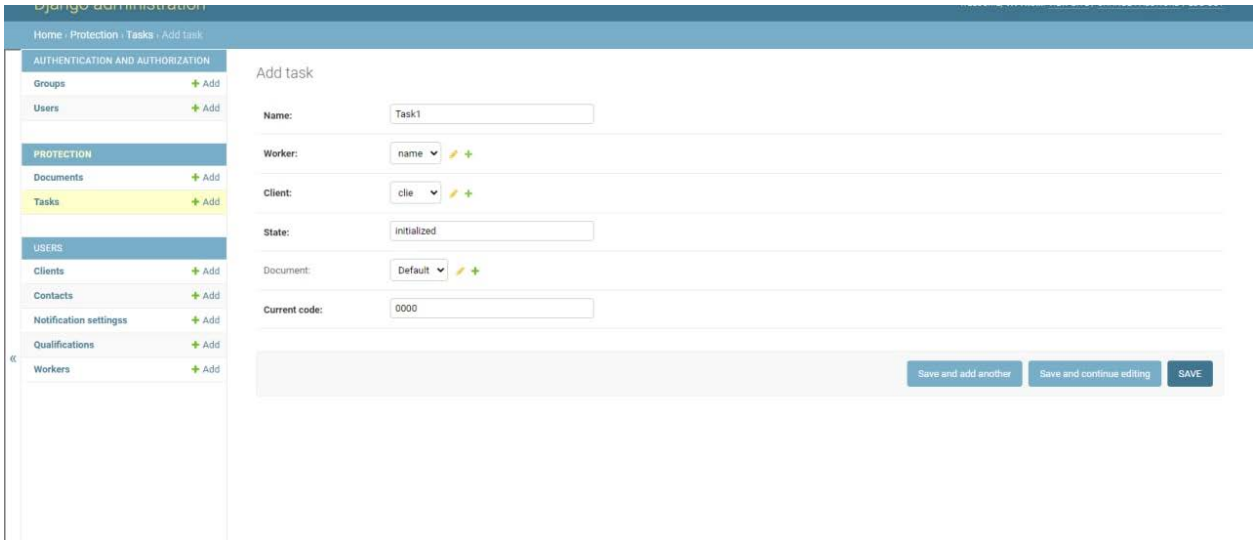

Рисунок 8.1 – Створення задачі та призначення працівника

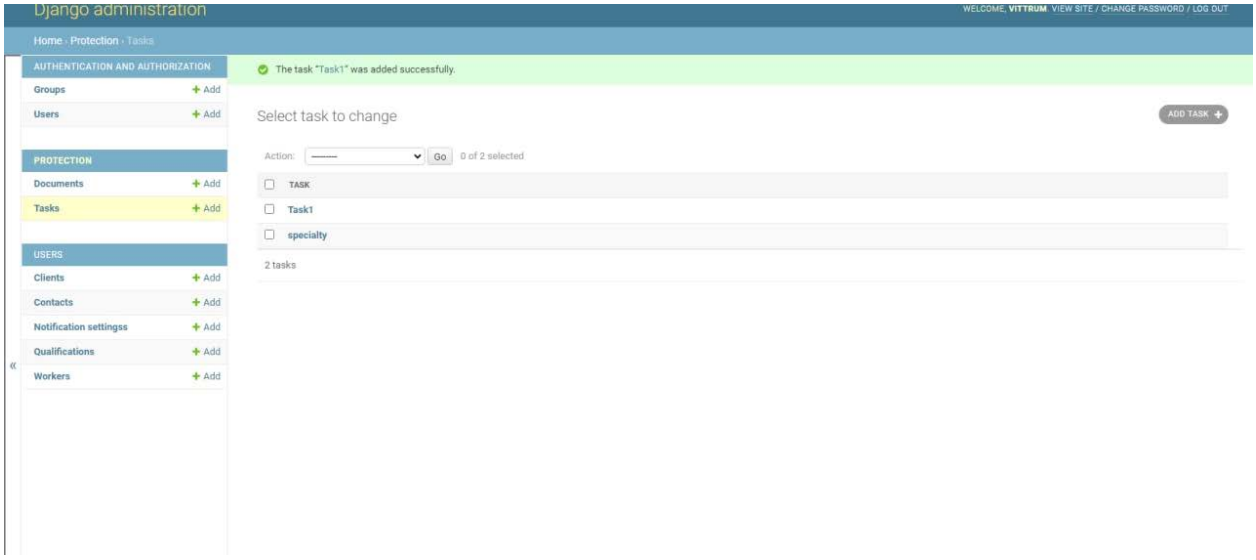

Рисунок 8.2 – Задачу створено

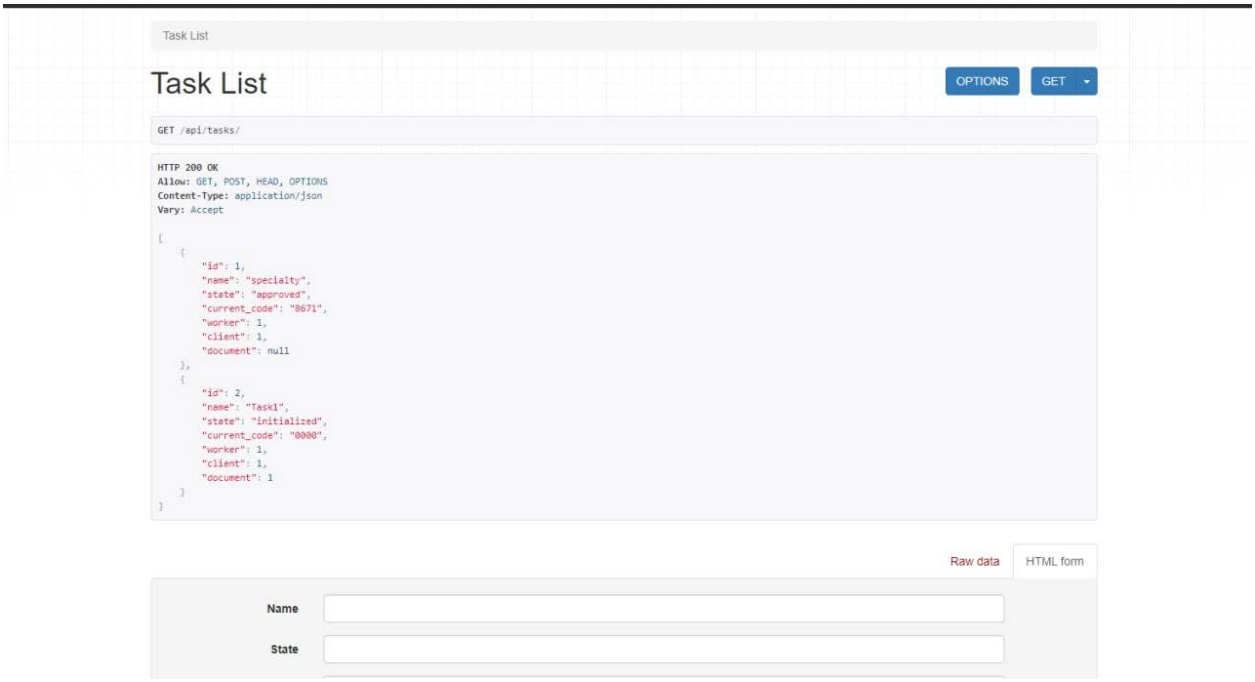

Рисунок 8.3 – Ендпоінт для вибору задач

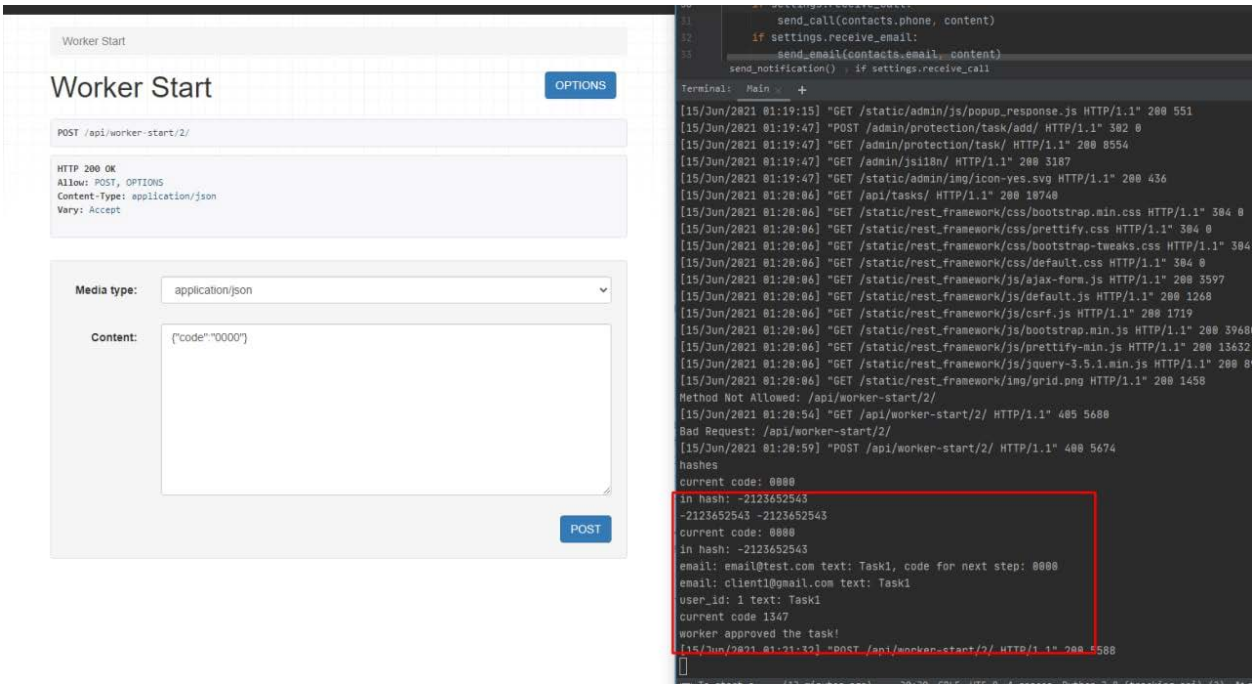

Рисунок 8.4 – Початок задачі із застосуванням правильного коду

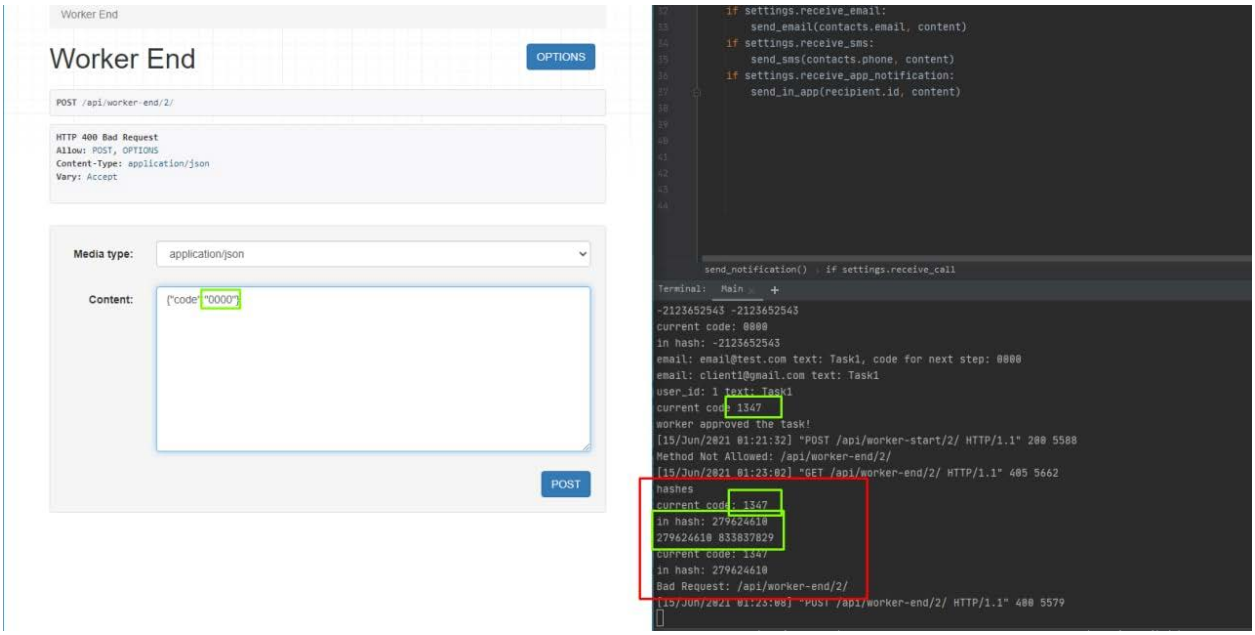

Рисунок 8.5 – Завершення задачі з невірним кодом. Задачу не завершено через неспівпадання хеш-значень.

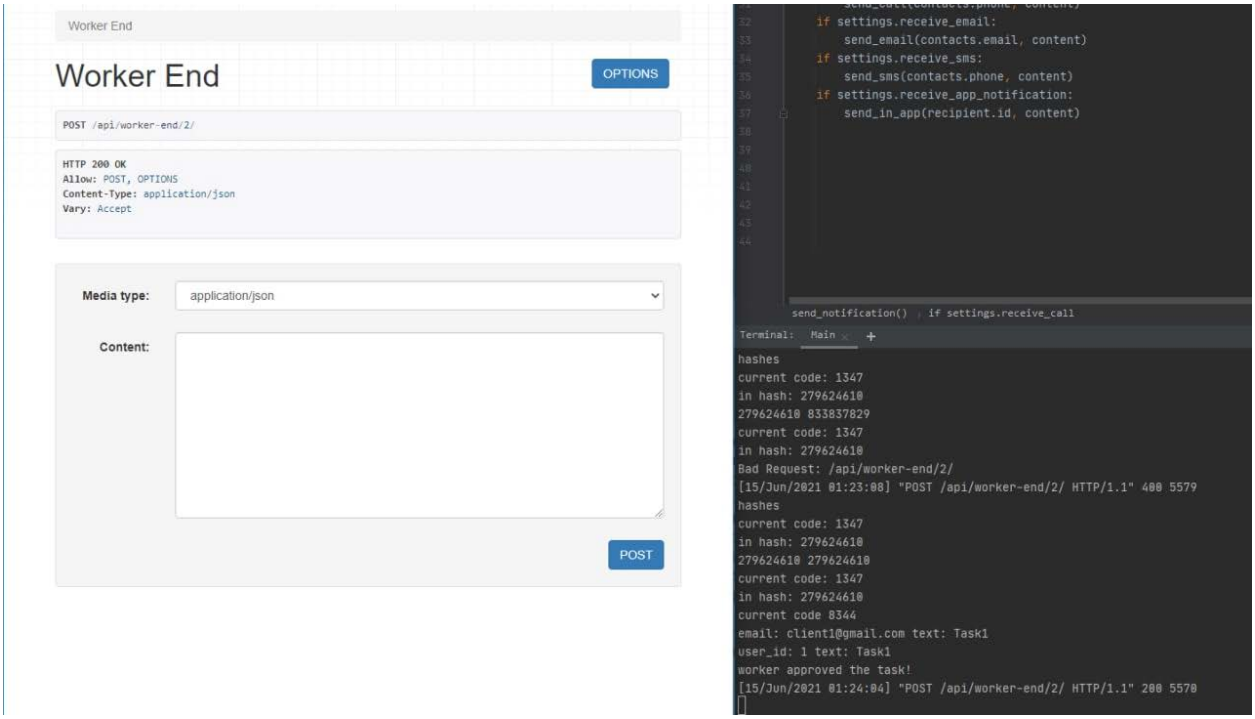

Рисунок 8.6 – Завершення задачі з коректним значенням

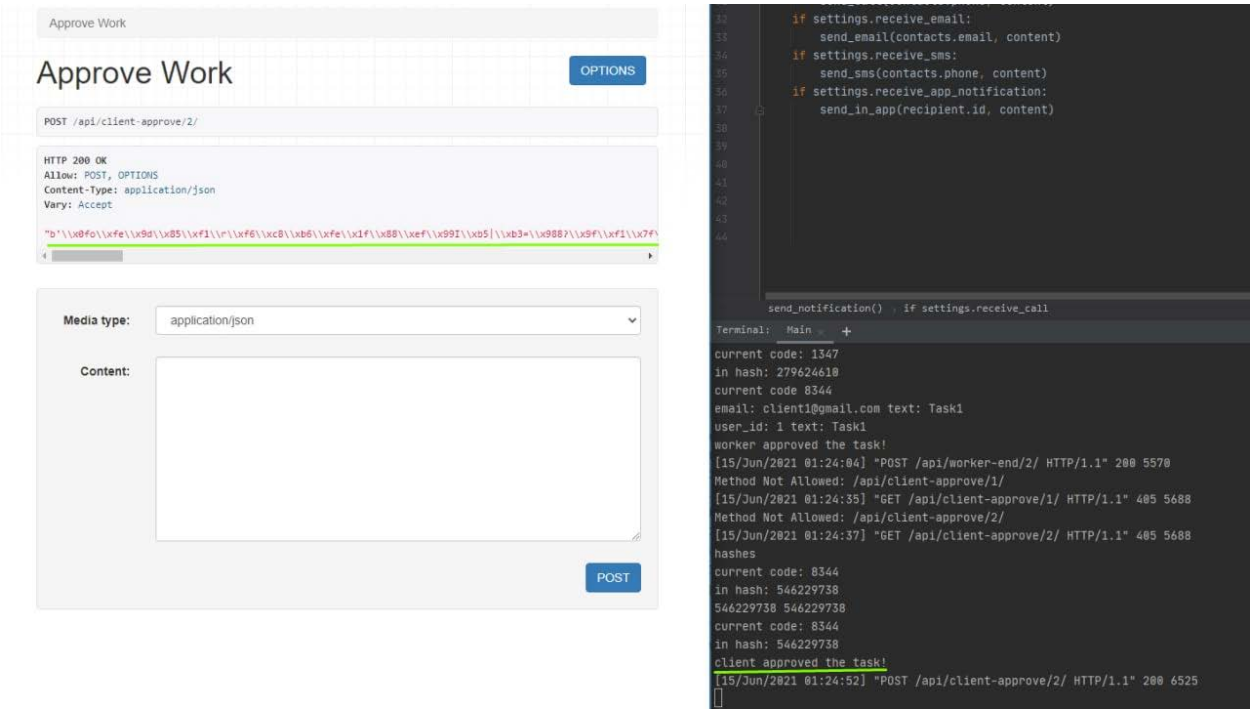

Рисунок 8.7 – підтвердження задачі клієнтом. При успішному підтвердженні згенеровано байтову стрічку ЕЦП

ВИСНОВОК

У процесі виконання дипломної роботи розроблена система верифікації даних через SMS/QR/мобільний додаток за допомогою цифрового підпису.

Проведено аналіз існуючих рішень, які застосовуються для рішення поточної або схожих проблем, розглянуто декілька корпоративних та декілька доступних рішень. Розроблено алгоритм верифікації даних за допомогою цифрового підпису. Верифікація проходить у декілька етапів перед формуванням цифрового підпису, порівнюючи хеши кодів та статусів. Цифровий підпис будується за допомогою хешування даних та шифрування їх за допомогою алгоритму RSA.

Побудована модель системи для підтримки алгоритму верифікації даних. У якості інструментів розробки були обрані мова Python - як основа, фреймворк Django та база даних PostgreSQL для серверної частини. Також побудовано API для комунікації та масштабування системи за допомогою Django Rest Framework.

Програмно реалізований алгоритм верифікації даних за допомогою цифрового підпису. Побудована база даних для системи підтримки. Програмно реалізована система підтримки алгоритму.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Криптосистеми та ЕЦП [Електронний ресурс] – режим доступу: http://crypto-r.narod.ru/glava6/glava6_3.html
2. Криптосистема Ель Гамаля [Електронний ресурс] – режим доступу: https://it.rfei.ru/course/~k017/~V8u3Fj4l/~hIGNMjZS
3. Система ИСТИНА [Електронний ресурс] – режим доступу: https://docs.istina.msu.ru/data_input/verification.html
4. Закон України про електронні документи [Електронний ресурс] – режим доступу: https://zakon.rada.gov.ua/laws/main/2155-19#Text
5. Електронний Цифровий Підпис [Електронний ресурс] – режим доступу: https://leksika.com.ua/14451127/legal/tsifroviy_elektronniy_pidpis
6. Алгоритм RSA – режим доступу: https://www.e-nigma.ru/stat/rsa/

ДОДАТОК А

Моделі:

```
import random

from django.db import models

# Create your models here.
from users.models import Client, User, Worker

class Document(models.Model):
    name = models.CharField(max_length=100)
    content = models.CharField(max_length=500)

    class Meta:
        db_table = 'documents'

    def __str__(self):
        return self.name

class Task(models.Model):
    name = models.CharField(max_length=100)
    worker = models.ForeignKey(Worker, on_delete=models.CASCADE,
 related_name='worker')
    client = models.ForeignKey(Client, on_delete=models.CASCADE,
 related_name='client')
    state = models.CharField(max_length=50, default='initialized
')
    document = models.ForeignKey(Document, on_delete=models.CASC
ADE, null=True, blank=True, default=None)
    current_code = models.CharField(max_length=4, default='0000'
)

    class Meta:
        db_table = 'tasks'
```

```
    def __str__(self):
        return self.name

    def is_approved(self) -> bool:
        return self.state == 'approved'

    def check_hash(self, code: str) -> bool:
        print('hashes')
        print(self.__hash__(), hash((code, self.state)))
        return self.__hash__() == hash((code, self.state))

    def __hash__(self):
        print(f'current code: {self.current_code}')
        print(f'in hash: {hash((self.current_code, self.state))}
')
        return hash((self.current_code, self.state))

    def generate_code(self) -> str:
        self.current_code = str(random.randint(1000, 9999))
        print(f'current code {self.current_code}')
        return self.current_code


from django.db import models

# Create your models here.
class Contact(models.Model):
    email = models.CharField(max_length=256, blank=True, null=Tr
ue)
    phone = models.CharField(max_length=11, blank=True, null=Tru
e)
    adress = models.CharField(max_length=100, blank=True, null=T
rue)
```

```
    def __str__(self):
        return self.email

    class Meta:
        db_table = 'contacts'

class NotificationSettings(models.Model):
    receive_sms = models.BooleanField(default=False)
    receive_call = models.BooleanField(default=False)
    receive_email = models.BooleanField(default=True)
    receive_app_notification = models.BooleanField(default=False
)

    def __str__(self):
        return f'{self.receive_call}, {self.receive_sms}, {self.
receive_email}'

    class Meta:
        db_table = 'notification_settings'

class User(models.Model):
    name = models.CharField(max_length=50)
    lastname = models.CharField(max_length=50)
    status = models.IntegerField()  # add choices
    contact = models.ForeignKey(Contact, on_delete=models.CASCAD
E)
    notification_settings = models.ForeignKey(NotificationSettin
gs, on_delete=models.CASCADE)

    class Meta:
        abstract = True

class Qualification(models.Model):
    licence = models.CharField(max_length=100, blank=True, null=
True)
```

```
    profession = models.CharField(max_length=100)
    qualification = models.CharField(max_length=100)

    def __str__(self):
        return self.licence

    class Meta:
        db_table = 'qualifications'

class Client(User):
    class Meta:
        db_table = 'clients'

    def __str__(self):
        return self.name

class Worker(User):
    qualification = models.ForeignKey(Qualification, on_delete=m
odels.CASCADE)

    class Meta:
        db_table = 'workers'

    def __str__(self):
        return self.name
```

Адміністративна панель:

```
from django.contrib import admin

from .models import Worker, Client, NotificationSettings, Contac
t, Qualification

# Register your models here.
```

```
admin.site.register(Client)
admin.site.register(Qualification)
admin.site.register(Contact)
admin.site.register(NotificationSettings)
admin.site.register(Worker)


# Register your models here.
from protection.models import Document, Task

admin.site.register(Task)
admin.site.register(Document)
```

Представлення:

```
from rest_framework import views, viewsets, status, serializers
# Create your views here.
from rest_framework.response import Response

from protection.models import Task
from .serializers import TaskSerializer
from .services.notifications import send_notification
from .services.workflow import report_work_end, report_work_star
t, approve_task, generate_document

class TaskViewSet(viewsets.ModelViewSet):
    queryset = Task.objects.all()
    serializer_class = TaskSerializer

class WorkerStartView(views.APIView):
    class InputSerializer(serializers.Serializer):
        code = serializers.CharField()

    def post(self, request, pk=None):
```

```
        serializer = self.InputSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        data = serializer.data
        task = Task.objects.get(id=pk)
        worker = task.worker
        if task.check_hash(data['code']):
            send_notification(worker, task, True)
            send_notification(task.client, task)
            _ = task.generate_code()
        else:
            return Response(status=status.HTTP_400_BAD_REQUEST)
        if report_work_start(task):
            return Response(status=status.HTTP_200_OK)
        else:
            return Response(status=status.HTTP_400_BAD_REQUEST)

class WorkerEndView(views.APIView):
    class InputSerializer(serializers.Serializer):
        code = serializers.CharField()

    def post(self, request, pk=None):
        serializer = self.InputSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        data = serializer.data
        task = Task.objects.get(id=pk)
        if task.check_hash(data['code']):
            _ = task.generate_code()
            send_notification(task.client, task)
        else:
            return Response(status=status.HTTP_400_BAD_REQUEST)
        if report_work_end(task):
            return Response(status=status.HTTP_200_OK)
        else:
            return Response(status=status.HTTP_400_BAD_REQUEST)
```

```
class ApproveWorkView(views.APIView):
    class InputSerializer(serializers.Serializer):
        code = serializers.CharField()

    def post(self, request, pk=None):
        serializer = self.InputSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        data = serializer.data
        task = Task.objects.get(id=pk)
        if task.check_hash(data['code']) and approve_task(task):
            doc = generate_document(task, b'TEXT FOR DOCUMENT')
            return Response(data=str(doc), status=status.HTTP_20
0_OK)
        else:
            return Response(status=status.HTTP_400_BAD_REQUEST)
```

Сервіси

```
from protection.models import Task
from protection.services.encryptions import sign_document, gener
ate_private_key

def report_work_start(task: Task) -> bool:
    try:
        task.state = 'started'
        task.save()
        print('worker approved the task!')
        return True
    except RuntimeError:
        print('Task is not approved by worker!')
        return False

def report_work_end(task: Task) -> bool:
    try:
        task.state = 'finished'
        task.save()
```

```
        print('worker approved the task!')
        return True
    except RuntimeError:
        print('Task is not ended by worker!')
        return False

def approve_task(task: Task) -> bool:
    try:
        task.state = 'approved'
        task.save()
        print('client approved the task!')
        return True
    except RuntimeError:
        print('Client not performed a task!')
        return False

def generate_document(task: Task, text: bytes, private_key: str
= None) -> str:
    if private_key is None:
        private_key = generate_private_key()
    if task.is_approved():
        result = sign_document(text, private_key)
    else:
        return 'failed to generate document'
    return result


from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding

def generate_private_key():
    private_key = rsa.generate_private_key(
        public_exponent=65537,
```

```
        key_size=2048,
        backend=default_backend()
    )
    return private_key

def sign_document(content, private_key):
    signature = private_key.sign(
        content,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )
    return signature

from typing import Union

from protection.models import Task
from users.models import Client, Worker

def send_email(email: str, text: str) -> None:
    print(f'email: {email} text: {text}')

def send_sms(phone: str, text: str) -> None:
    print(f'phone: {phone} text: {text}')

def send_in_app(user_id: int, text: str) -> None:
    print(f'user_id: {user_id} text: {text}')

def send_call(phone: str, text: str) -> None:
    print(f'phone: {phone} text: {text}')

def send_notification(recipient: Union[Worker, Client], task: Ta
sk, code: bool = False) -> None:
```

```
settings = recipient.notification_settings
content = f'{task.name}'
if code:
    content += f', code for next step: {task.current_code}'
contacts = recipient.contact

if settings.receive_call:
    send_call(contacts.phone, content)
if settings.receive_email:
    send_email(contacts.email, content)
if settings.receive_sms:
    send_sms(contacts.phone, content)
if settings.receive_app_notification:
    send_in_app(recipient.id, content)
```