Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

УДК 004.457

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: “ Кросплатформенне програмне забезпечення для роботи із захищеними носіями ЕЦП на основі PKCS11”

Спеціальність – 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

БР.ІПЗ - ___.___.____.____ ІПЗ

Студент

ІПЗ-44 МС_____ /Максим СОЛОВЙОВ/

Науковий керівник

д.т.н., професор _____/Олексій БИЧКОВ/

Консультант
з питань нормоконтролю

фахівець_____/Тамара ЧАПОВСЬКА/

Допускається до захисту

Завідувач кафедри

д.т.н., професор _____/ Олексій БИЧКОВ /

Київ – 2021

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

Освітньо-кваліфікаційний рівень - бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖЕНО
Зав. кафедри програмних систем і технологій
_____ (Олексій БИЧКОВ)

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ

Соловйова Максима Едуардовича

(прізвище, ім'я, по батькові)

1. **Тема роботи:** «Кросплатформенне програмне забезпечення для роботи із захищеними носіями ЕЦП на основі PKCS11»

Керівник проекту: д.т.н., професор Бичков Олексій Сергійович

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

Затверджені наказом вищого навчального закладу від «___» _____2020р. №_____

2. **Строк здачі студентом закінченої роботи** «___» _____2021р.

3. **Вихідні дані до дипломної роботи:** монографії, підручники, навчальні посібники, статті та тези конференцій вітчизняних і зарубіжних авторів, Інтернет-ресурси з питань електронного підпису.

4. **Зміст пояснювальної записки:**

1) Аналітична частина:

проаналізувати існуючі програмні рішення та засоби накладання електронного підпису, обґрунтувати доцільність обраної теми дослідження, порівняти з вже існуючими рішеннями на українському та зарубіжному ринку.

2) Практична частина:

спроектувати архітектуру програмного забезпечення для накладання електронного підпису, розробити програмне забезпечення, розробити інтерфейс користувача.

5. **Перелік графічного матеріалу (із зазначенням обов'язкових креслень)**:

1) Аналітична частина: 3 рисунків.
2) Практична частина: 23 рисунків, 15 листів додатку програмного коду.

6. **Консультанти розділів проекту:**

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1 і 2 розділи аналітична частина	Олексій БИЧКОВ	18.01.2021	18.01.2021
3 розділ практична частина	Олексій БИЧКОВ	18.01.2021	18.01.2021

7. **Дата видачі завдання** «___» _____ 2021 р.

Керівник _____ (Олексій БИЧКОВ)

Завдання прийняв до виконання _____ (Максим СОЛОВЙОВ)

Календарний план

№ з/п	Назва етапів виконання етапів бакалаврської роботи	Термін виконання етапів роботи	Відмітка про виконання
1	Уточнення постановки задачі	29.11.2020-05.12.2020	Виконано
2	Аналіз літератури	06.12.2020-04.01.2021	Виконано
3	Аналіз існуючих методів, концепцій та алгоритмів вирішення завдання	09.01.2021-16.01.2021	Виконано
4	Побудова алгоритмічної моделі основних процесів	28.01.2021-14.02.2021	Виконано
5	Опис розробленого алгоритму	15.02.2021-20.03.2021	Виконано
6	Розроблення програмного забезпечення	21.03.2021-30.04.2021	Виконано
7	Тестування розробленого програмного забезпечення	02.05.2021-15.05.2021	Виконано
8	Оформлення і друк пояснювальної записки	16.05.2021-26.05.2021	Виконано
9	Оформлення презентації	27.05.2021-03.06.2021	Виконано
10	Отримання рецензії		
11	Затвердження пояснювальної записки роботи завідувачем кафедри		
12	Захист дипломної роботи		

Студент – бакалавр _____ (Максим СОЛОВЙОВ)

Керівник роботи _____ (Олексій БИЧКОВ)

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 69 с., 26 рис., 17 джерел, 4 додатки.

Тема: «Кросплатформенне програмне забезпечення для роботи із захищеними носіями ЕЦП на основі PKCS11»

Об'єкт дослідження: електронний цифровий підпис.

Предмет дослідження: програмні засоби для накладання підпису.

Мета роботи: підвищити ефективність і легкість роботи з електронним підписом.

При розробці кросплатформенного програмного забезпечення було вирішено такі задачі:

1. Дослідження процесу розробки кросплатформенних застосунків за допомогою сучасних технологій;
2. Проаналізувати особливості розробки на основі WebAssembly;
3. Проаналізувати архітектурні рішення даного способу розробки;
4. Реалізація додатку на основі технологій Electron.js та .NET Core.

Результати дослідження: створене програмне забезпечення може бути використано на кафедрі та факультеті, наприклад для підпису лабораторних, курсових робіт студентів.

Висновок: Розроблений додаток для накладання електронно цифрового підпису, на відмінну від існуючих, надає можливість легко і зручно працювати із захищеним носієм.

Ключові слова: кросплатформенний, електронний цифровий підпис, JavaScript, Express, Node.js, Electron, .NET Core, WebAssembly.

АННОТАЦИЯ

Выпускная квалификационная бакалаврская работа: 69 с., 26 рис., 17 источников, 4 приложений.

Тема: «Кроссплатформенное программное обеспечение для работы с защищенными носителями ЭЦП на основе PKCS11»

Объект исследования: электронная цифровая подпись.

Предмет исследования: программные средства для наложения подписи.

Цель работы: повысить эффективность и легкость работы с электронной подписью.

При разработке кроссплатформенный программного обеспечения были решены следующие задачи:

1. Исследование процесса разработки кроссплатформенних приложений с помощью современных технологий;
2. Проанализировать особенности разработки на основе WebAssembly;
3. Проанализировать архитектурные решения данного способа разработки;
4. Реализация приложения на основе технологий Electron.js и .NET Core.

Результаты исследования: создано программное обеспечение может быть использовано на кафедре и факультете, например для подписи лабораторных, курсовых работ студентов.

Вывод: Разработано приложение для наложения электронной цифровой подписи, в отличие от существующих, позволяет легко и удобно работать с защищенным носителем.

Ключевые слова: кроссплатформенный, электронная цифровая подпись, JavaScript, Express, Node.js, Electron, .NET Core, WebAssembly.

ABSTRACT

Final qualification bachelor's work: 69 p., 26 pictures, 17 sources, 4 annexes.

Topic: “Cross-platform software for working with protected EDS carriers based on PKCS11”

Object of study: electronic digital signature.

Subject of study: signature software.

Purpose of work: to increase the efficiency and ease of work with an electronic signature.

When developing cross-platform software, the following tasks were solved:

1. Researching the process of developing cross-platform applications using modern technologies;
2. Analyze the features of development based on WebAssembly;
3. Analyze the architectural solutions of this development method;
4. Implementation of the application based on Electron.js and .NET Core technologies.

Research results: created software can be used at the department and faculty, for example, for signing laboratory, term papers of students.

Conclusion: An application for the use of electronic digital signatures has been developed, in contrast to the existing ones, it allows you to easily and conveniently work with a protected medium.

Keywords: cross-platform, electronic digital signature, JavaSript, Express, Node.js, Electron, .NET Core, WebAssembly.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ОС – операційна система.

API (прикладний програмний інтерфейс) – це комплекс вже готових класів, функцій, структур і тд., які надаються застосунком (бібліотекою, сервісом) чи операційною системою для використання у зовнішніх програмних продуктах.

CLI (Command Line Interface) – інтерфейс командного рядка, консоль.

HTML – це мова розмітки веб-сторінок у мережі Інтернет.

HTTP – протокол передачі даних, що використовується у інфокомунікаційних мережах.

IDE (Integrated Development Environment) – інтегроване середовище розробки.

JavaScript (JS) – мова програмування високого рівня.

ЕЦП – електронний цифровий підпис.

АЦСК – акредитований центр сертифікації ключів.

ЦЗО – центральний засвідчу вальний орган.

ВСТУП

На сьогоднішній день в багатьох сферах нашого життя все більш стає популярнішим використання електронних документів, не тільки наряду с паперовими документами, а й замість них. Вже зараз технології дозволяють нам безпечно проводити онлайн-транзакції, підписувати платіжні доручення та електроні документи.

На поточний час в Україні на законодавчому рівні існує такий термін як ЕЦП (електронний цифровий підпис). Слід зазначити, що на даний час ЕЦП в Україні має юридичну силу та може бути рівноправним звичайному підпису або печатці. ЕЦП являє собою результат криптографічного перетворення даних у вигляді послідовності цифр і є надійним засобом захисту інформації, що забезпечує можливість контролю цілісності і підтвердження достовірності електронних документів.

В даний час ЕЦП використовується в основному для ідентифікації автора інформації і для доказу того факту, що підписане повідомлення або дані не були модифіковані під час передачі інформації в комп'ютерних мережах.

Механізм електронного цифрового підпису (ЕЦП) виник як побічний ефект криптографії з відкритим ключем. Тому, характерне для систем з відкритим ключем поділ ключа на 2 частини - секретну і несекретну, що дозволяє реалізувати можливість перевірки автентичності без можливості підписати інший документ.

Отже, цифровий підпис це кінцева цифрова послідовність, що залежить від самого повідомлення або документа і від секретного ключа, відомого тільки суб'єкту підписанту, призначена для встановлення авторства. Так як цифровий підпис будується на базі криптосистеми з відкритим ключем, то необхідно мати пару ключів - секретний і відкритий. Секретний ключ використовується для формування цифрового підпису, тому його потрібно зберігати в таємниці. А відкритий ключ використовується для перевірки відповідності підпису документу, тому він повинен бути опублікований, наприклад, в загальнодоступному каталозі.

Переваги електронного цифрового підпису над ручним підписом та печаткою:

– Ідентифікація автора.

- Неможливість підробки підпису.
- Електронна мітка часу, яка дозволяє встановити точний час коли був підписаний документ.
- Можливість ставити ЕЦП в онлайн середовищі, без необхідності друкувати документ.
- Простий спосіб перевірки підпису.

Метою моєї дипломної роботи є підвищити ефективність і легкість роботи з електронним підписом.

Новизна. Кросплатформенність, простий та інтуїтивний інтерфейс, використання сучасних стеку технологій, легкі для імплементації сервіси API.

Практичне значення дипломної роботи є в тому, що створене програмне забезпечення може бути використано на кафедрі та факультеті, наприклад для підпису лабораторних, курсових робіт студентів.

Дипломна робота складається з: вступу, трьох розділів, що включають 18 підрозділів, висновків, переліку джерел посилань із 17 найменувань. У листі дипломної роботи міститься 26 рисунків. Загальний обсяг роботи 69 листів.

РОЗДІЛ 1
АНАЛІЗ ЕЛЕКТРОННОГО ЦИФРОВОГО ПІДПИСУ

1.1. Аналіз алгоритмів шифрування для електронного цифрового підпису

Алгоритм шифрування поділяються на асиметричні (з відкритим, публічним ключем) та симетричний (із закритим, приватним ключем). Алгоритми шифрування, які називаються симетричними, базуються на принципі, що відправник і одержувач, користуються однаковим ключем. Приватний ключ повинен зберігатись у таємниці та передаватися таким чином, щоб запобігти його перехопленню. Якщо приватний ключ захищений, розшифрування повідомлення автоматично ідентифікує відправника що він є власником ключа, за допомогою якого було зашифровано інформацію та одержувач має ключ, за допомогою якого можливо розшифрувати повідомлення.

Безпека цього типу шифрування залежіть від безпеки ключа, а не від збереження алгоритму в таємниці. Зазвичай симетричний варіант ЕЦП передбачає наявність у системі третьої сторони – «арбітр», який виконує обов’язки довіреної особи користувачів. Засвідчення справжності документа визначається самим фактом шифрування його приватним ключем і передавши його арбітру. Використовується рідко, оскільки немає ефективних алгоритмів.

Переваги симетричного методу ЕЦП:

- криптостійкість симетричних схем має високу продуктивність заявки стабільності використовуваних блок-шифрів. Їх надійність також достатньо досліджена;
- якщо стійкість шифру недостатня, його легко замінити іншим.

Недоліки симетричного методу створення ЕЦП:

- необхідно підписати кожен з бітів інформації, що значно збільшує підпис;
- ключі, створені для підписання, можна використовувати лише один раз, оскільки після підпису половина секретного ключа розсекречується.

Будь-які криптосистеми, засновані на використанні приватного ключа, безпосередньо залежать від ступеня секретності цих даних. Користувач може

зберігати ключ на своє персональному комп’ютері, наприклад, захистивши його паролем. Але цей варіант має свої недоліки:

– документи можна підписувати лише на комп’ютері власника ЕЦП;
– зберігання даних ЕЦП безпосередньо залежить від безпеки комп’ютера користувача.

Алгоритм шифрування, який називають асиметричним (або шифрування за допомогою відкритого ключа), в ньому використовується два ключі – секретний ключ (закритий або приватний ключ) і відкритий (публічний ключ), створений таким чином, що їх послідовне застосування до інформаційного об’єкту не змінює цей об’єкт. Ці ключі різні і не можуть бути похідними один від одного, незважаючи на те, що вони генеруються разом.

Для шифрування використовується лише публічний ключ, а для розшифрування лише приватний ключ. Розшифрувати інформацію, не знаючи приватного ключа, є надто складним завданням. Зокрема, проблема обчислення приватного ключа за відомим відкритим ключем практична нерозв’язна.

Основна перевага шифрування із публічним ключем – це простий механізм передачі ключів. При підключенні передається лише відритий ключ. Це дозволяє користуватися звичайним каналом зв’язку і виключає необхідність у спеціальному каналі захисту для передачі ключів.

Асиметрична модель побудови ЕЦП має свої недоліки:

– на основі асиметричних схем ЕЦП, а також асиметричного шифрування, є довільно складною задачею (дискретна задача про логарифм або проблема факторизації);
– криптографічна стабільність алгоритмів шифрування відкритих ключів на сьогодні не доведено математично;
– для вдосконалення криптостійкості необхідно зробити довжину ключів більшою.

Цифровий підпис на основі асиметричного шифрування дозволяє отримувачу повідомлення перевірити достовірність джерела інформації (іншими словами, автора інформації), а також перевірити, чи не змінилася інформація під час передачі

адресату. Тому цифрові підписи є засобом ідентифікації та контролю цілісності даних.

1.2. Аналіз та принцип роботи з PKCS11

В Public-Key Cryptography Standards (PKCS) являє собою групу криптографічних стандартів , які забезпечують керівні принципи і прикладні програмні інтерфейси (API) для використання криптографічних методів. Як випливає з назви PKCS, ці стандарти роблять акцент на використанні криптографії із відкритим ключем (тобто асиметричною).

PKCS11 - це стандарт інтерфейсу криптографічного маркера, який визначає API, який називається Cryptoki . За допомогою цього API програми можуть звертатися до криптографічних пристроїв як маркери і можуть виконувати криптографічні функції, реалізовані цими маркерами. Цей стандарт, вперше розроблений лабораторіями RSA у співпраці з представниками промисловості, науки та урядів, зараз є відкритим стандартним керівництвом, яким керує Технічний комітет OASIS PKCS11.

Він дотримується об'єктно-орієнтованого підходу, що стосується цілей незалежності технологій (будь-який тип HW-пристроїв) та спільного використання ресурсів. Він також представляє додаткам загальний, логічний вигляд пристрою, який називається криптографічним маркером. PKCS11 присвоює ідентифікатор слота кожному маркеру. Додаток ідентифікує маркер, до якого хоче отримати доступ, вказавши відповідний ідентифікатор слота.

1.3. Законність електронного підпису в Україні

Україна вже давно в світі цифрових підписів. Електронні підписи в Україні визнаються з 2003 року. З чинним законодавством, Україна продовжує розширювати свої міжнародні відносини.

Цей закон визначає правовий статус будь-якого електронного документа та передбачає, що юридична дійсність електронних документів та його прийнятність як

доказ у разі потреби не можуть бути спростовані на підставі єдиного фактора, що він знаходиться в електронній формі.

Електронні підписи регулюються окремими законами, спеціальними правовими актами та урядовими постановами в Україні. Існує закон про електронні цифрові підписи, закон про електронні документи та укази електронного документообігу про процедури затвердження та нормативні акти щодо затвердження правил посиленої сертифікації.

Електронні підписи регулюються двома законами. Ці два закони є нейтральними до технологій. Закони України від 22.05.2003 р. №852-IV та від 22.05.2003 р. №851-IV використовуються для контролю та регулювання підписів.

Оскільки закон розрізняє два типи електронних підписів, електронний підпис визначається як дані в електронній формі, які додаються до інших електронних даних для ідентифікації. Електронний цифровий підпис визначається як тип електронного підпису, який отримується через криптографічне перетворення в електронний набір даних.

Електронний цифровий підпис застосовується за допомогою персонального ключа та перевіряється за допомогою відкритого ключа.

Щоб електронний підпис зробив документ легальним, він повинен бути засвідчений відкритим ключем. Сертифікат - це документ, який видається акредитованим центром сертифікації ключів (АЦСК), і він також підтверджує дійсність відкритого ключа.

Без цього документ не може бути законним незалежно від наявності електронного підпису. Можна використовувати два типи сертифікатів, простий сертифікат відкритого ключа та посилений сертифікат відкритого ключа.

Електронний документообіг підтверджує, що документи та контракти не можуть бути відмовлені у виконанні лише тому, що вони знаходяться в електронній формі. У разі виникнення суперечок, сторони повинні представити докази в суді, як це передбачено законом. Договори, підписані електронним підписом, мають ті самі юридичні повноваження, що і договори рукописного підпису.

1.4. Електронна мітка часу

Мітка часу являє собою механізм , який дозволяє довести цілісність ряду даних. Це демонструє, що ці дані існували в певний момент і з тих пір не змінювалися.

Цілісність - саме одна з вимог, яка дозволяє забезпечити безпеку процесу підписання та мати повні юридичні гарантії для всіх сторін.

Таким чином, мітка часу може також гарантувати цілісність набору електронних даних, що складають електронний підпис. Це означає, що позначка часу гарантує, що підпис, зроблений у будь-який момент часу, не може бути змінений згодом.

Крім того, мітка часу також гарантує незмінність ряду даних, пов'язаних з електронним підписом, таких як дата, час і місце, в якому зроблено підпис, електронна адреса відправника документа, електронна адреса підписанта тощо.

У цьому сенсі мітка часу надає додаткову цінність електронному підпису не лише тому, що гарантує цілісність даних, що складають підпис, але й тому, що включає відповідну інформацію про момент його створення.

У деяких випадках може бути дуже важливо точно знати, в який час був підписаний договір, оскільки правовідносини з підписантом починаються саме після моменту, коли контракт підписується.

Саме з цієї причини відмітка про час не може бути надана однією із сторін, що беруть участь у процесі підписання, оскільки в цьому випадку гарантії цілісності підпису та інших пов'язаних даних не будуть надійними.

Відмітку часу повинна надавати довірена третя сторона, яка відома як орган штампа часу.

Орган штампів часу (часто скорочується як TSA) - це постачальник послуг із сертифікації, послуга якого полягає у виконанні довіреної третьої сторони, пропонуючи офіційні позначки часу.

Мітка часу розширює переваги електронного підпису в порівнянні з підписами, зробленими на папері, оскільки надає колишній правовій визначеності.

У разі підписів на папері неможливо визначити час чи місце, де були зроблені підписи. Натомість електронні підписи надають ці докази з усіма гарантіями незмінності, які дає їм мітка часу.

Отже, у випадку суперечки щодо моменту набрання чинності контрактом, наприклад, ті, хто має контракт, підписаний в електронному вигляді за допомогою вдосконаленого електронного підпису, матимуть можливість достовірно довести, хто, коли та де був підписаний контракт.

Цей останній фактор забезпечує велику перевагу для електронного підпису: він не тільки замінює рукописний підпис, але також пропонує юридичну силу всім сторонам, які беруть участь у підписанні договору.

1.5. Акредитований центр сертифікації ключів

Акредитований центр сертифікації ключів (АЦСК) — акредитований державою у встановленому порядку орган, що надає послуги з надання ЕЦП з одночасним постачанням захищених носіїв ключової інформації або з використанням власних носіїв заявників типу USB-флеш-накопичувач.

Центральний засвідчуваний орган здійснює акредитацію сертифікаційних центрів в Україні. Усі данні захищенні від підробок та можливості отримання третіми особами.

Акредитований центр сертифікації ключів має право:

– надавати послуги електронного цифрового підпису та обслуговувати виключно посилені сертифікати ключів;

– отримувати та перевіряти інформацію, необхідну для реєстрації підписанта і формування посиленого сертифіката ключа, безпосередньо у юридичної або фізичної особи чи її представника.

Законодавством України встановлені зобов’язання та вимоги, які має виконувати акредитований центр сертифікації ключів, та додатково зобов’язаний

використовувати для надання послуг електронного цифрового підпису надійні засоби електронного цифрового підпису.

Постановою Кабінета Міністрів України від 19 вересня 2018 р. № 749 встановлений порядок акредитації та вимоги, яким повинен відповідати акредитований центр сертифікації ключів.

Відмінністю АЦСК є те, що він має право обслуговувати виключно посилені сертифікати ключів.

Відкриті ключі й інша інформація про користувачів зберігається центрами сертифікації, у вигляді цифрових сертифікатів, що мають наступну структуру:

– серійний номер сертифіката;

– об’єктний ідентифікатор алгоритму електронного підпису;

– ім’я центра, який оформив сертифікат;

– строк придатності;

– ім’я власника сертифіката (ім’я користувача, якому належить сертифікат);

– відкритий ключ власника сертифіката;

– об’єктні ідентифікатори алгоритмів, асоційованих з відкритими ключами власника сертифіката.

1.6. Аналіз актуального стану ЕЦП

Експерти з використання інфраструктури відкритих ключів пояснюють, що інфраструктура відкритих ключів (у міжнародній термінології PKI - інфраструктура відкритих ключів) складається з усіх промислових розвинутих країн. У той же час національні ІПК не можуть взаємодіють між собою, бо навіть в об’єднаній Європі вони не надають юридичне значення, необхідне для транскордонного спілкування.

Проблема полягає у використанні національних криптографічних алгоритмів, різні для різних країн. Відмінностей в алгоритмах не дозволяють повною мірою говорити про взаємну довіру держав у галузі інформаційної безпеки. Крім того, ліквідація національних електронних підписів за межами своїх держав означитимуть зміну міжнародного права, наприклад, Гаазька конвенція 1961 р., яка регулює

міждержавні відносини взаємне визнання звичайних паперових документів, прийняття окремої спеціалізованої конвенція про процедури електронної ідентифікації документів.

Інша сторона законопроекту - це положення, яке повноваження щодо захисту інформації передаються інформаційно-телекомунікаційній системі, що використовується при наданні послуг ЕЦП, а також створення особи та забезпечення законності функціонування належать органу сертифікації ключів. Крім того, цей центр очікується буде зобов'язаний:

- своєчасно попереджати абонента та додавати відкриті сертифікати ключ від особи, яка заповнила поле про обмеження використання цифрових підпис;
- здійснювати електронну реєстрацію виданих сертифікатів ключів та реєстрація заблокованих та анульованих сертифікатів ключів;
- надати користувачам сертифікатів ключів доступ до іменний записи протягом дня. Зберігати паперові документи та електронні носії масової інформації;
- своєчасно інформувати орган із сертифікації або центральний орган сертифікації для зміни даних, відтворених у самому сертифікаті ключа (зміни до статті 8 Закону № 852-IV).

На основі визначення органу з сертифікації ключів як частини глобальної служби каталогів, яка відповідає за управління криптографічними користувацькими ключами, надання цих дозволів дійсно доречно і покликаний забезпечити захист електронних документів у всіх країнах протягом всього його життєвого циклу.

Повноваження державного нагляду у сфері електронного цифрового підпису, автори законопроекту пропонують забезпечити створений основний орган в системі центральних органів виконавчої влади та забезпечити виконання державної політики у сферах специфіки організації зв'язку та захисту інформації, телекомунікації. Передбачається, що контролюючий орган проведе аудит центрального органу з сертифікації та створить власні сертифікати ключів. У законодавчому полі є багато недоліків щодо ЕЦП, які заважатимуть повному

використанню цієї технології. До цих недоліків може включати, наприклад, надмірно вузький обсяг. Так, положення Конвенції ООН регулюють лише правовідносини, які виникають між комерційними підприємствами, не впливаючи на правовідносини, що виникають між державними органами, державою та бізнесом, державою і громадянином, бізнесом і громадянином або між громадянами.

Більше того, у сфері правовідносин, що виникають між комерційними підприємствами, виключена електронна взаємодія у сфері фінансових та фондових ринків.

З вищевказаних причин можна зробити висновок, що технологія ЕЦП важко застосувати до широкого кола завдань у тих сферах, які є вимушеними регулювати свої процеси через державні структури. Але у випадку приватних компаній ніщо не заважає компаніям застосовувати цю технологію локально, аби не зберігати велику кількість паперових документів в архівах. Завдяки впровадженню ЕЦП компанія, яка потребує автентичності даних, може бути впевнена в документах, що зберігаються та відмовитися від паперових документів.

Висновки до розділу

Електронний цифровий підпис є найбільш перспективним і поширеним методом захисту електронних документів від підробки і незмінності інформації під час її передач. Закон України допускає користування методом ЕЦП для передачі корпоративних, фінансових та інших документів.

Головна проблема поширення електронного підпису – це відсутність визнаного офіційного органу, що впроваджують інфраструктуру відкритого ключа на базі українських алгоритмів. Це є необхідним для організації системи ЕЦП.

Електронний цифровий підпис є ефективним рішенням для тих, хто не хоче чекати поки прибуде кур’єр чи агент компанії для укладання договору та його перевірки. Документи можна підписувати цифровим методом і доставити у потрібне місце за короткий проміжок часу. Усі учасники електронного документообігу знаходяться в однакових умовах, незважаючи на відстань.

З одного боку, закон ЕЦП є більш українським технологічним стандартом. Незважаючи на те, що старий закон ЕЦП був, як щось нове, і є варіант не розглядати таку потрібну всім правову реформу. Втім потрібність такого закону та проблеми з електронним підписом в майбутньому стане ще біль поширеним.

Тому найближчим часом використання технологій накладання ЕЦП, можуть стати необхідними не лише для банків та компаній, що мають високий рівень ризику, але також для середнього та малого бізнесу.

РОЗДІЛ 2
ВИБІР ОПТИМАЛЬНОГО ПІДХОДУ ДО РОЗРОБКИ КРОСПЛАТФОРМЕННОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі описано деякі популярні системи та платформи, які використовувались для створення та роботи з кросплатформенним програмним забезпеченням, властивості та можливості, а також їх аналіз. Тут будуть описуватись такі технології як WebAssembly та ElectronJS, які є важливими для створення кросплатофменного програмного забезпечення, а також інші, такі як, ReactJS та .Net.

2.1. Технологія WebAssembly

Уже два десятиліття є лише одна мова програмування, яка може бути використана у веб-браузері: JavaScript. Повільна смерть сторонніх двійкових плагінів виключає інші мови, такі як Java та Flash ActionScript, як першокласних технологій для веб-розробки. Інші веб-мови, такі як CoffeeScript, просто компілюються в JavaScript.

Але тепер є нова можливість: WebAssembly, або коротше WASM. WebAssembly - це невеликий, швидкий двійковий формат, який обіцяє майже рідну продуктивність веб-додатків. Крім того, WebAssembly розроблений для того, щоб стати ціллю компіляції для будь-якої мови, JavaScript є лише однією з них. З кожним основним браузером, який зараз підтримує WebAssembly , настав час серйозно задуматися над написанням веб-програм на стороні клієнта, які можна скомпілювати як WebAssembly.

WebAssembly, розроблений W3C, є, за словами його творців, "ціллю компіляції". Розробники не пишуть WebAssembly безпосередньо. Вони пишуть мовою на свій вибір, яка потім компілюється в байт-код WebAssembly. Потім байт-код запускається на клієнті - зазвичай у веб-браузері - де він перекладається у власний машинний код і виконується з високою швидкістю.

Код WebAssembly призначений для швидшого завантаження, аналізу та виконання, ніж JavaScript. Коли WebAssembly використовується веб-браузером, все ще залишаються накладні витрати на завантаження модуля WASM та його налаштування, але за інших рівних умов WebAssembly працює швидше. WebAssembly також надає ізольовану модель виконання, засновану на тих самих моделях безпеки, які існують для JavaScript зараз.

Найпростіший варіант використання WebAssembly - це ціль для написання програмного забезпечення в браузері. Компоненти, які компілюються до WebAssembly, можуть бути написані будь-якою з багатьох мов; остаточне корисне навантаження WebAssembly потім доставляється клієнту через JavaScript.

WebAssembly був розроблений з урахуванням багатьох прикладів використання, що базуються на браузері : ігор, потокового передавання музики, редагування відео, САПР, шифрування та розпізнавання зображень.

В цілому, корисно зосередитись на цих трьох областях, визначаючи конкретний випадок використання WebAssembly:

1) Високопродуктивний код, який уже існує мовою, на яку можна націлити. Наприклад, якщо є високошвидкісна математична функція, вже написана на мові C, і є потреба включити її у веб-програму, тоді є можливість розгорнути її як модуль WebAssembly. Менш важливі для продуктивності, орієнтовані на користувача частини програми можуть залишатися в JavaScript.
2) Високопродуктивний код, який потрібно писати з нуля, де JavaScript не є ідеальним. Раніше для написання такого коду можна було використовувати asm.js. Все ще можете це зробити, але WebAssembly позиціонується як краще довгострокове рішення.
3) Перенесення настільної програми у веб-середовище. Багато демо-версій технологій для asm.js та WebAssembly потрапляють до цієї категорії. WebAssembly може забезпечити основу для програм, які є більш амбіційними, ніж просто графічний інтерфейс, представлений через HTML. Однак це не є тривіальною вправою, оскільки всі способи

взаємодії настільного додатка з користувачем повинні бути зіставлені з еквівалентами WebAssembly / HTML / JavaScript.

Таким чином, більшість сценаріїв роботи з WebAssembly передбачають написання коду мовою високого рівня та перетворення його на WebAssembly. Це можна зробити будь-яким із трьох основних способів:

1) Безпосереднє складання. Джерело перекладається на WebAssembly за допомогою власного ланцюжка інструментів компілятора. Такі як, Rust, C/C++, Kotlin/Native та D мають рідні способи випускати WASM від компіляторів, які підтримують ці мови.
2) Сторонні інструменти. Мова не має власної підтримки WASM у своєму ланцюжку інструментів, але утиліта третьої частини може бути використана для перетворення на WASM. Java, Lua та сімейство .Net мають певну підтримку, подібну цій.
3) Інтерпретатор на основі WebAssembly. Тут сама мова не перекладається на WebAssembly. Швидше, інтерпретатор мови, написаний у WebAssembly, запускає код, написаний цією мовою. Це найбільш громіздкий підхід, оскільки інтерпретатор може складати кілька мегабайт коду, але він дозволяє існуючому коду, написаному мовою, запускати всі, крім незмінних. У обох Python та Ruby перекладачі перекладені на WASM.

WebAssembly не підтримує безпосередньо мови, які використовують збірник сміття, моделі пам'яті. Такі мови, як Lua або Python, можна підтримувати, лише обмежуючи набори функцій або вбудовуючи весь час виконання як виконуваний файл WebAssembly. Але триває робота над підтримкою моделей пам’яті, збірником сміттям, незалежно від мови та реалізації .

Вбудована підтримка потоків є загальною для таких мов, як Rust та C++. Відсутність підтримки потоків у WebAssembly означає, що цілі класи програмного забезпечення, орієнтованого на WebAssembly, не можуть бути написані цими мовами. Пропозиція додати багатопоточність WebAssembly використовує C++ потокової моделі як один з його натхненників.

Однією з найбільших проблем із перекладеним JavaScript було труднощі налагодження та профілювання через неможливість співвідношення між перекладеним кодом та джерелом. У WebAssembly у нас є подібна проблема.

2.2. JavaScript та ReactJS

JavaScript - це динамічна мова комп'ютерного програмування. Він легкий і найчастіше використовується як частина веб-сторінок, реалізації яких дозволяють клієнтському сценарію взаємодіяти з користувачем та створювати динамічні сторінки. Це інтерпретована мова програмування з об’єктно-орієнтованими можливостями.

JavaScript спочатку був відомий як LiveScript, але Netscape змінив свою назву на JavaScript, можливо, через хвилювання, яке створює Java. JavaScript вперше з’явився в Netscape 2.0 у 1995 році під назвою LiveScript . Ядро мови загального призначення було вбудовано в Netscape, Internet Explorer та інші веб-браузери.

Специфікація ECMA-262 визначила стандартну версію основної мови JavaScript:

- JavaScript - це легка, інтерпретована мова програмування;
- призначений для створення мережево орієнтованих додатків;
- доповнює та інтегрує Java;
- доповнює та інтегрує HTML;
- відкритий і крос-платформний.

Клієнтський JavaScript - найпоширеніша форма мови. Сценарій повинен бути включений у документ HTML або посилатися на нього, щоб код інтерпретувався браузером.

Це означає, що веб-сторінка не повинна бути статичним HTML, але може включати програми, які взаємодіють з користувачем, керують браузером та динамічно створюють вміст HTML.

Клієнтський механізм JavaScript надає багато переваг перед традиційними серверними сценаріями CGI. Наприклад, ви можете використовувати JavaScript, щоб перевірити, чи ввів користувач правильну адресу електронної пошти у поле форми.

Код JavaScript виконується, коли користувач подає форму, і лише якщо всі записи є дійсними, вони будуть надіслані на веб-сервер.

JavaScript можна використовувати для захоплення ініційованих користувачем подій, таких як клацання кнопок, навігація посиланнями та інші дії, які користувач ініціює явно або неявно.

Переваги використання JavaScript:

- менше взаємодії з сервером – є можливість перевірити введені користувачем дані перед відправкою сторінки на сервер. Це економить серверний трафік, а це означає менше навантаження на сервер;
- негайний відгук відвідувачам - їм не потрібно чекати перезавантаження сторінки, щоб побачити, чи не забули вони щось ввести;
- підвищена інтерактивність – є можливість створювати інтерфейси, які реагують, коли користувач наводить на них курсор миші або активує їх за допомогою клавіатури;
- багатіші інтерфейси – є можливість використовувати JavaScript, щоб включити такі елементи, як компоненти перетягування та повзунки, щоб надати розширений інтерфейс відвідувачам сайту.

React - це бібліотека JavaScript для побудови користувальницьких інтерфейсів. В основі всіх програм React лежать компоненти. Компонент - це автономний модуль, який надає певний результат. Необхідно писати елементи інтерфейсу, як кнопку або поле введення як компонент React. Компоненти можна скласти . Компонент може включати один або кілька інших компонентів у свій результат.

Загалом кажучи, для написання програм React пишуться компоненти React, які відповідають різним елементам інтерфейсу. Потім організовуємо ці компоненти всередині компонентів вищого рівня, які визначають структуру додатку.

Наприклад, візьмімо форму. Форма може складатися з багатьох елементів інтерфейсу, таких як поля введення, мітки або кнопки. Кожен елемент всередині

форми може бути записаний як компонент React. Потім потрібно написати компонент вищого рівня, сам компонент форми. Компонент форми визначає структуру форми та включає кожен з цих елементів інтерфейсу всередині неї.

Що важливо, кожен компонент у програмі React дотримується суворих принципів управління даними. Складні інтерактивні користувальницькі інтерфейси часто включають складні дані та стан додатків. Площа поверхні React обмежена і спрямована на те, щоб дати інструменти, які дозволять передбачити, як буде виглядати додаток за певних обставин.

Перш за все, DOM розшифровується як "Модель об'єкта документа". DOM простими словами представляє інтерфейс вашої програми. Щоразу, коли змінюється стан інтерфейсу додатка, DOM оновлюється, щоб відобразити цю зміну. Зараз улов часто маніпулює DOM, впливає на продуктивність, роблячи її повільною.

DOM представлений у вигляді деревної структури даних. Через це зміни та оновлення DOM відбуваються швидко. Але після зміни оновлений елемент та його діти повинні бути повторно відтворені для оновлення інтерфейсу програми. Повторне відображення або перефарбовування інтерфейсу - це те, що робить його повільним. Отже, чим більше компонентів інтерфейсу, тим дорожчими можуть бути оновлення DOM, оскільки їх потрібно буде рендерити для кожного оновлення DOM.

Ось де концепція віртуального DOM з’являється і працює значно краще, ніж реальна DOM. Віртуальний DOM - це лише віртуальне представлення DOM. Щоразу, коли змінюється стан нашого додатка, віртуальний DOM оновлюється замість реального DOM (рис. 2.1).

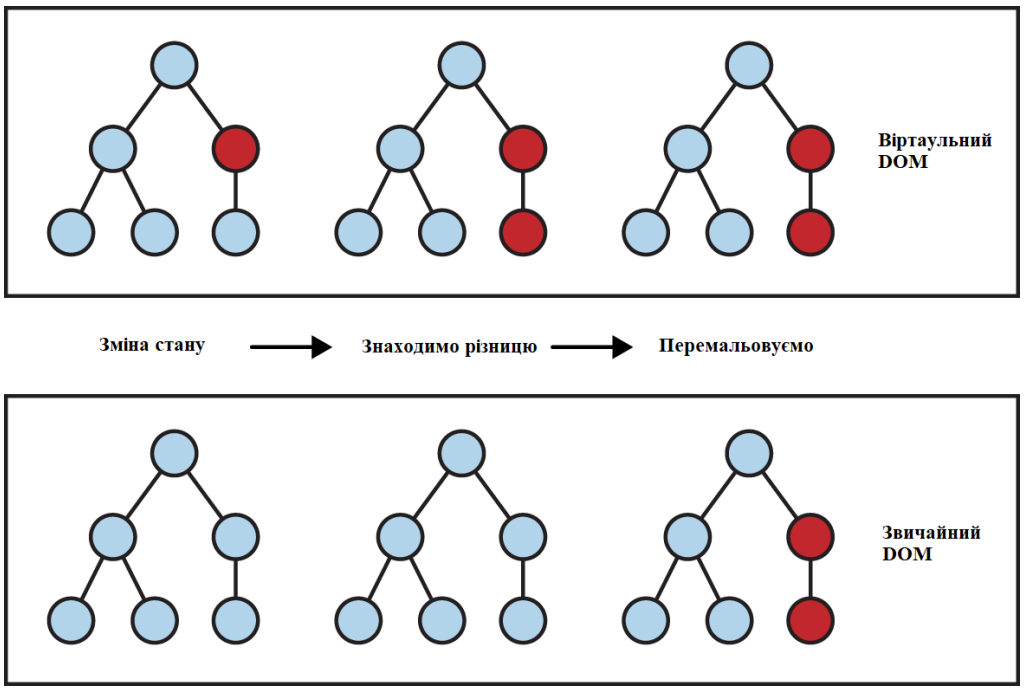

Рис. 2.1 Віртуальний та звичайний DOM

Зафарбовані кола представляють вузли, які змінилися. Ці вузли представляють елементи інтерфейсу, які змінили стан. Потім обчислюється різниця між попередньою версією дерева віртуального DOM і поточним деревом віртуального DOM. Потім все батьківське піддерево отримує рендерінг, щоб отримати оновлений інтерфейс. Потім це оновлене дерево пакетно оновлюється до реального DOM.

2.3. Технологія Electron.js

На найпростішому рівні Electron - це повнофункціональна структура побудови додатків, яка дозволяє створювати настільні програми за допомогою веб-технологій. Це виділяє кілька великих речей:

1) Створені настільні програми є крос-платформними і працюватимуть у Windows, Linux та macOS, не потребуючи декількох баз коду.
2) Написання коду у звичайному HTML, CSS та JavaScript і можливість використовувати будь-які інструменти та веб-фреймворки /бібліотеки.
3) Веб-код може отримати доступ до власних системних API та майже до всього іншого на комп'ютері.

4) Встановлення та автоматичне оновлення надаються безкоштовно.
5) Написаний код та пов’язаний вміст можуть бути локальними для програми або віддалено подаватися із сервера, подібного до гібридного додатка.

Electron використовує модуль npm, який широко використовується для JavaScript. Він складається з рідного меню для діалогів та сповіщень. Інсталятори Windows не потребують будь-якої конфігурації.

Він також має можливість автоматичного оновлення та повідомлення про збої в Windows і Mac за допомогою Squirrel. Звіти про збої передаються на віддалений сервер для подальшого аналізу. Chromium надає такі дії, як відлагодження та профілювання вмісту.

Важкими та нудними частинами створення настільного додатка є спрощення упаковки, встановлення, оновлення, підтримка власних меню, сповіщень, діалогових вікон та врешті-решт оптимізація звітів про збої програми.

Electron в значній мірі піклується про всі ці основні кроки, щоб користувач міг зосередитись на ядрі своєї програми.

Кожен процес має різну роль. Завантаження програми виконується основним процесом. Він може протистояти іншим подіям життєвого циклу програми, таким як запуск, вихід із програми, підготовка до виходу та інші легкі завдання, такі як вихід у фоновий режим та вихід на передній план.

З іншого боку, процес візуалізації породжується основним процесом. Процеси візуалізації відображатимуть інтерфейс користувача програми. Кожен процес використовує багатопроцесорну архітектуру Chromium і працює на своєму потоці.

Архітектура додатку зробленому завдяки Electron виглядає наступним чином (рис. 2.2).

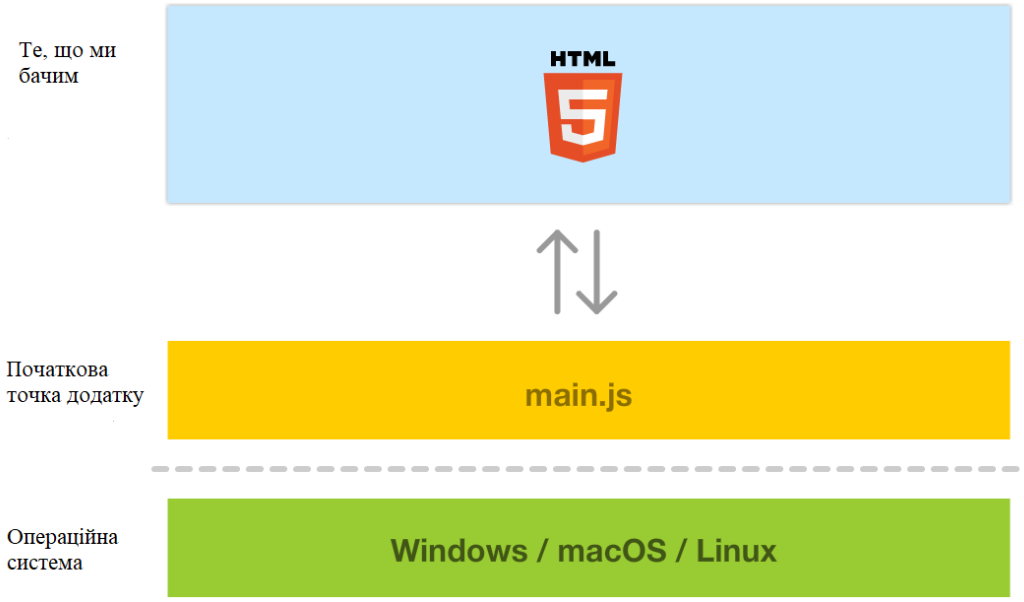

Рис. 2.2 Архітектура Electron додатку

Отже, Electron як ніщо інше підходить, для реалізації кросплатформенного програмного забезпечення.

2.4. Фреймворк .Net

.NET - це безкоштовна платформа з відкритим кодом та крос-платформа для створення сучасних, масштабованих та високопродуктивних настільних, веб-, хмарних та мобільних додатків. Поточна версія .NET - .NET 5.0, яка є наступницею .NET Core 3.1 та .NET Framework 4.6. До .NET 5.0 .NET мав дві версії .NET Framework та .NET Core, але в цій версії обидві об'єдналися, і тепер існує лише версія.

.NET - це єдина уніфікована платформа для створення додатків для настільних комп'ютерів, Інтернету, хмарних, мобільних, ігрових, IoT та AI (рис. 2.3). Екосистема .NET має єдину загальну бібліотеку, середовище виконання, компілятори мови та інструменти.

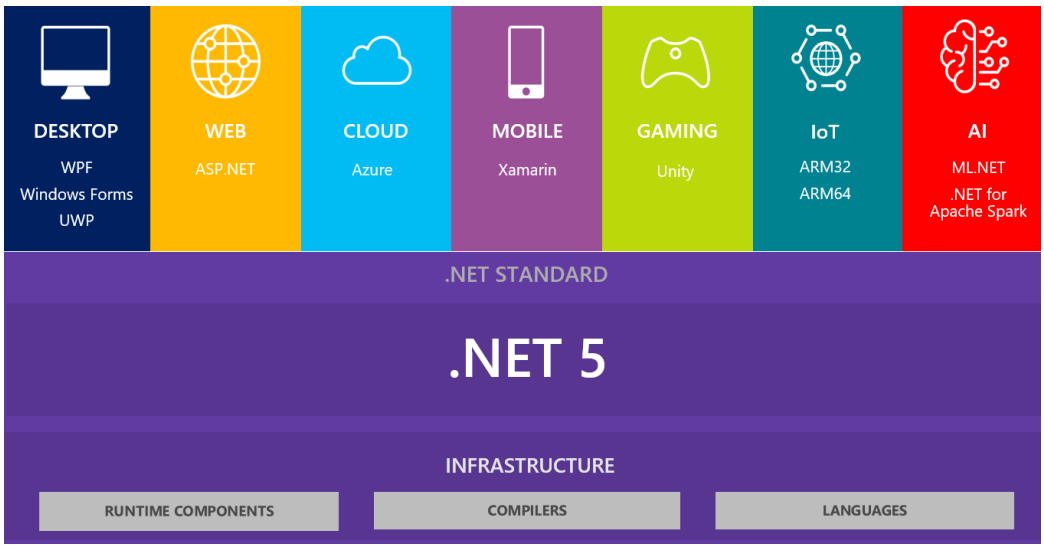

Рис. 2.3 Платформи які підтримує .NET

Історично склалося так, що .NET Framework працював лише на пристроях Windows. Проекти Xamarin та Mono працювали над тим, щоб перенести .NET на мобільні пристрої, macOS та Linux. .NET Core забезпечує стандартну базову бібліотеку, яку тепер можна використовувати в Windows, Linux, macOS та мобільних пристроях (через Xamarin).

Існує чотири основні компоненти архітектури .NET:

1) Загальна специфікація мови (CLS) визначає, як реалізуються об'єкти, щоб вони працювали скрізь .NET. CLS - це підмножина загальної системи типів (Common Type System - CTS), яка встановлює загальний спосіб опису всіх типів.
2) Бібліотека класів Framework (FCL) - це стандартна бібліотека, яка збирає багаторазові класи, інтерфейси та типи значень.
3) Загальномовна середовище виконання (CLR) - це віртуальна машина, яка запускає структуру та керує виконанням програм .NET.
4) Такі інструменти, як Visual Studio, для створення окремих програм, інтерактивних веб-сайтів, веб-програм та веб-служб.

Розробники використовують .NET framework для створення настільних додатків Windows та серверних додатків. Сюди входять веб-програми ASP.NET. .NET Core використовується для створення серверних додатків, що працюють на

Windows, Linux та Mac. На даний момент він не підтримує створення настільних програм з користувальницьким інтерфейсом. Розробники можуть писати програми та бібліотеки у VB.NET, C # та F # в обох робочих середовищах.

Отже, з вище вказаного, для розробки кросплатформенного програмного забезпечення, вибір падає на .NET Core, так як, .NET Framework підтримується лише на операційній системі Windows. Для операційних систем Linux та Mac, поки ще немає підтримки .NET Framework.

2.5. Application Programming Interface

API - це набір визначень та протоколів для побудови та інтеграції прикладного програмного забезпечення. API означає інтерфейс прикладного програмування.

API дозволяють товару або послузі взаємодіяти з іншими продуктами та послугами, не знаючи, як вони реалізовані. Це може спростити розробку додатків, заощадити час і гроші. Коли починається розробка нових інструментів та продуктів - або керування вже існуючими - API надають гнучкість та надають можливості для інновацій.

Оскільки API спрощують те, як розробники інтегрують нові компоненти програми в існуючу архітектуру, вони допомагають бізнесу та IT-командам співпрацювати. Потреби бізнесу часто швидко змінюються у відповідь на постійно мінливі цифрові ринки, де нові конкуренти можуть змінити цілу галузь за допомогою нового додатка. Щоб залишатися конкурентоспроможними, важливо підтримувати швидкий розвиток та впровадження інноваційних послуг. Розробка власних хмарних додатків - це ідентифікований спосіб збільшити швидкість розробки, і він покладається на підключення архітектури додатків мікросервісів через API.

API - це спрощений спосіб підключення власної інфраструктури через розробку власних хмарних додатків, але вони також дозволяють ділитися своїми даними із клієнтами та іншими зовнішніми користувачами. Загальнодоступні API представляють унікальну ділову цінність, оскільки вони можуть спростити та

розширити спосіб зв’язку зі своїми партнерами, а також потенційно монетизувати дані (популярним прикладом є API Карт Google).

2.6. Платформа Node.js та фреймворк Express.js

Node.js визначається як "платформа, створена в середовищі виконання JavaScript в Chrome для легкого створення швидких масштабованих мережних додатків." Простіше кажучи, Node.js це мультиплатформенна середовище з відкритим вихідним кодом, яка дозволяє створювати серверні додатки і інструменти за допомогою JavaScript.

Node.js фактично вивів JavaScript на новий рівень. У той час як JavaScript використовується в якості мови розробки на стороні клієнта, Node.js охоплює розробку на стороні сервера. Завдяки Node.js, JavaScript стала універсальною мовою для фулстек (full-stack) розробки.

Фронтенд, створений на JS, і бекенд, створений на Node.js, легше синхронізувати через однієї мови, використовуваного з обох сторін програми. Node.js дозволяє писати програми на JavaScript і виконувати їх на сервері.

Node.js став одним з найпопулярніших інструментів в бекенд-розробці. Великі компанії, такі як eBay, Netflix і Uber, використання Node.js в своїх сервісах.

З точки зору розробника, Node.js уможливив спільне використання та повторне використання коду. За допомогою модулів Node розробники можуть використовувати готові модулі або перепрофілювати свої власні.

Express.js це мінімальний і гнучкий фреймворк для WEB-додатків з відкритим вихідним кодом Node.js. Ви можете використовувати його поверх Node.js щоб забезпечити кращу WEB-функціональність. Express найпопулярніший WEB-фреймворк Node.js.

Він надає великий набір функцій для створення WEB-додатків (односторінкових, багатосторінкових і гібридних). За допомогою Express можна структурувати WEB-додаток, яке може обробляти кілька HTTP-запитів за певним URL-адресою.

Гнучкість проявляється в численних компонентах, доступних в менеджері пакетів. Ці компоненти автоматично переходять в Express.js.

Причина, по якій Express є найпопулярнішим WEB-фреймворком, полягає в тому, що він полегшує розробку WEB-додатків, WEB-сайтів і API. Він також пропонує базову колекцію топографій.

З Express.js, ви зможете поліпшити різні аспекти WEB-додатки. Ви можете визначити такі параметри, як розташування шаблонів, які будуть використовуватися для відповіді, або порт, який буде використовуватися для встановлення з'єднання.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ КРОСПЛАТФОРМЕННОГО ДОДАТКУ ДЛЯ РОБОТИ З ЕЦП

Реалізований RESTful веб-сервіс дозволяє третім системам отримати дані по ключам користувача, та запросити підпис. Форматом обміну даними з сервісом є JSON. Користувач має можливість налаштувати додаток, так як зручно йому:термін кешування паролю, та тема додатку. Також користувач має можливість продивитись доступні ключі, та інформації по ним, підписати файл перетягнувши його на ключ, яким є бажання підписати. Для розробки використовувався програмний продукт Visual Studio Code, Express.js серверної JavaScript-платформи Node.js, .NET Core для написання бібліотеки для роботи з захищеним носієм, Electron та React для побудови користувацького інтерфейсу.

3.1 Опис структури програмного забезпечення

Враховуючі всі вище описані вимоги, було розроблено архітектуру проекту. Написання бібліотеки на .NET Core, перенесення бібліотеки в WebAssembly, за допомогою цього, ця бібліотека буде працювати в Electron так як, він працює на Chromium (рис. 3.1). Побудова користувацького інтерфейсу на компонентах React. А також, написання API сервісів за допомогою Express.js.

Було обрано найбільш оптимальні засоби розробки, а саме:

- Visual Studio Code – IDE від Microsoft;
- PostMan – для тестування HTTP запитів;
- Node.js – JavaScript платформа для реалізації серверу;
- Express – фреймворк Node.js;
- Electron – платформа за допомогою котрої буде можливість пакувати додаток для різних операційних систем;
- HTML5 – мова розмітки;
- .NET Core – для роботи із захищеними носіями.

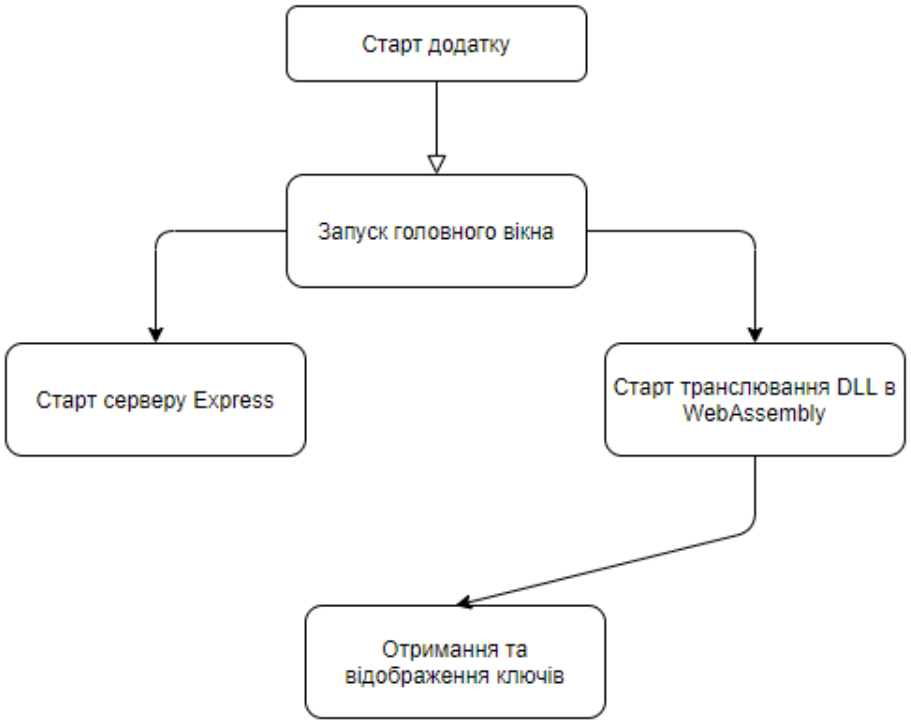

Рис. 3.1 Діаграма старту додатка

3.2 Структура бібліотеки для роботи із захищеним носієм

Для написання бібліотеки було використано .NET Core версії 3.1. Було розроблено 78 класів на мові C#.

Головний клас, який буде точкою входу в проект, і самі ці методи будуть визиватися в Electron називається Startup (лістинг 3.1).

Лістинг 3.1 Клас Startup

```
public class Startup
    {
        public Task<object> Invoke(object input)
        {
            try
            {
                var result = new KeysService().GetKeys();
                return Task.FromResult<object>(ResultHelper.FromResult(result));
            }
            catch (Exception ex)
            {
                return Task.FromResult<object>(ResultHelper.FromException(ex));
            }

        }
        public Task<object> Sign(object input)
        {
            try
            {
                var currentType = typeof(SignRequest);
```

```
                var props = currentType.GetProperties(BindingFlags.Public |
BindingFlags.Instance)
        .Where(x => x.GetSetMethod() != null);

                // create an instance of the type
                var obj = (SignRequest)Activator.CreateInstance(currentType);
                var values = (IDictionary<string, object>)input;
                foreach (var prop in props)
                    prop.SetValue(obj, values[prop.Name]);
                var result = new SignService().Sign(obj);
                return Task.FromResult<object>(ResultHelper.FromResult(result));
            }
            catch (Exception ex)
            {
                return Task.FromResult<object>(ResultHelper.FromException(ex));
            }
        }

        public Task<object> Login(object input)
        {
            try
            {
                var currentType = typeof(LoginRequest);
                var props = currentType.GetProperties(BindingFlags.Public |
BindingFlags.Instance)
        .Where(x => x.GetSetMethod() != null);

                // create an instance of the type
                var obj = (LoginRequest)Activator.CreateInstance(currentType);
                var values = (IDictionary<string, object>)input;
                foreach (var prop in props)
                    prop.SetValue(obj, values[prop.Name]);
                new KeysService().Login(obj);
                LoginRequest result = null;
                return Task.FromResult<object>(ResultHelper.FromResult(result));
            }
            catch (Exception ex)
            {
                return Task.FromResult<object>(ResultHelper.FromException(ex));
            }
        }
          }
```

Як видно з лістингу, основною незвичністю є те, що всі методи які будуть використані через WebAssembly повинні приймати один параметр, з назвою input та типом даних object, а також повертати завжди Task<object>.

Метод Invoke повертає ключі які знаходяться на захищеному носії. Інформація про ключі є публічними даними тому не потребують вводу паролю.

Метод Sign підписує передані дані, для цього методу вже потрібний пароль, тому якщо пароль не був введений або час кешування вийшов, метод поверне помилку і запросить пароль у користувача.

Метод Login передає пароль на захищений носій, що дозволяє використовувати захищені методи носія, такі як, підпис або шифрування.

Для кросплатформенної роботи із захищеним носієм використовується 3 динамічних бібліотеки, а саме:

– CryptokiPKCS11.dll – для операційної системи Windows;
– CryptokiPKCS11.so – для операційної системи Linux;
– CryptokiPKCS11.dylib – для операційної системи MacOS.

Метод який вичислює на якій операційний системі була запущена бібліотека називається GetPathToDll в класі DllHelper (лістинг 3.2).

Лістинг 3.2 Клас DllHelper

```
public static class DllHelper
    {
        public static string GetPathToDll()
        {
            if ((Environment.OSVersion.Platform == PlatformID.Unix ||
                        Environment.OSVersion.Platform == PlatformID.MacOSX ||
                        (int)Environment.OSVersion.Platform == 128))
            {
                return Platform.IsLinux ? PathToSo : PathToDylib;
            }
            else
            {
                return PathToDll;
            }
        }
        public static string PathToDylib = "CryptokiPKCS11.dylib";
        public static string PathToDll = "CryptokiPKCS11.dll";
        public static string PathToSo = "CryptokiPKCS11.so";
          }
```

Після того як система вичислює яку бібліотеку треба динамічно завантажити в пам’ять, її методи стане можливо використовувати.

Для роботи з приватним ключем, було розроблено абстрактні класи, окремі для роботи з ключами на основі ДСТУ та ключами на основі RSA (рис 3.2).

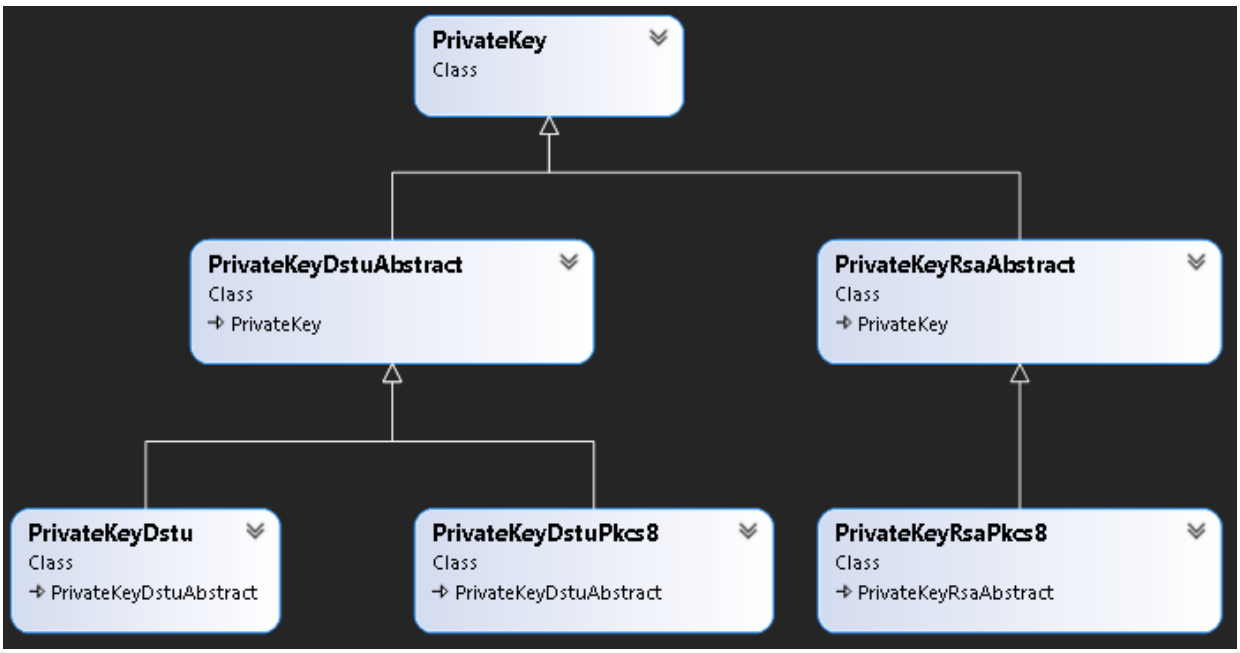

Рис. 3.2 Діаграма класів PrivateKey

3.3 Робота з бібліотекою .NET Core в Electron

Для перенесення бібліотеки .NET Core в WebAssembly було використано публічний пакет npm edge-js. За допомогою нього, є можливість використовувати .NET методи в JavaScript. Було написано файл dll-methods.js (лістинг 3.3).

Лістинг 3.3 Виклик .NET методів в JavaScript

```
const references = [
    "ArCryptolib.dll",
    "Asn1Net.Reader.dll",
    "Pkcs11Interop.dll",
    "Newtonsoft.Json.dll",
];
export const clrMethod = edge.func({
    assemblyFile: "eCryptor.dll",
    methodName: "Invoke",
    references: references,
});

export const signMethod = edge.func({
    assemblyFile: "eCryptor.dll",
    methodName: "Sign",
    references: references,
});

export const loginMethod = edge.func({
    assemblyFile: "eCryptor.dll",
```

```
    methodName: "Login",
    references: references,
});
```

Як видно з лістингу наведеного вище, викликається метод edge.func, котрий приймає в себе об’єкт з 3 властивостями, а саме:

– assemblyFile – посилання на бібліотеку;
– methodName – назва методу, ці методи в казані в лістингу 3.1;
– references – сторонні бібліотеки які пов’язані з нашою бібліотекою.

Метод edge.func повертає посилання на наш метод в WebAssembly, який ми вже можемо викликати в JavaScript (лістинг 3.4).

Лістинг 3.4 Приклад виклику методу

```
clrMethod(null, function (error, result) {
                if (error) throw error;
                console.log(result);
                if (result.Status == "OK") {
                    actions.keyStoreActions.setValue(result.Result);
                    return;
                }
                throw new Error(result.ErrorMessage);
            });
```

Посилання на метод приймає два параметри, перший – це параметр який буде передано до нашої бібліотеки, той самий object input, та другий параметр – це колбек функція, яка в свою чергу теж приймає два параметри, error якщо щось пішло не так, та result результат виконання нашого методу.

3.4 Реалізовані API сервіси

За допомогою Express.js було реалізовано два API сервісів, а саме:

– отримання інформації про ключі користувача;
– запит на підписання даних.

Локальний сервер розгортається при старті додатку, та прослуховує порт 3031.

API сервіс для отримання інформації про ключі користувача доступний за адресою localhost:3031/api/keys GET запит (рис 3.3).

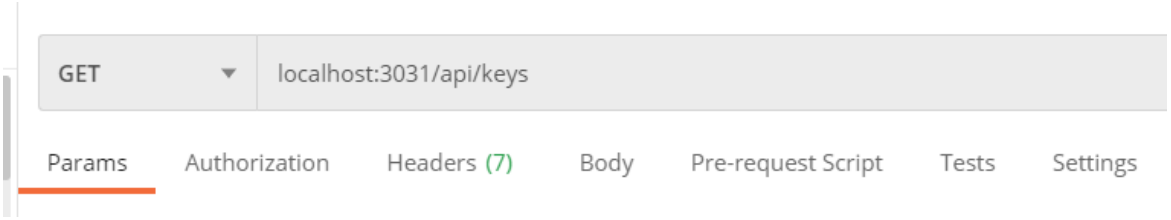

Рис. 3.3 Приклад виклику запиту на отримання інформації про ключі

Після виклику запиту користувачеві буде надіслано запит на підтвердження (рис. 3.4).

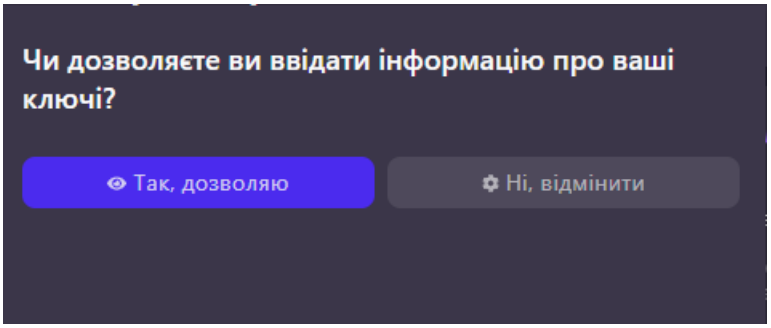

Рис. 3.4 Підтвердження

Якщо користувач обирає «Так, дозволяю» тоді ми отримаємо відповідь с ключами користувача (рис. 3.5).

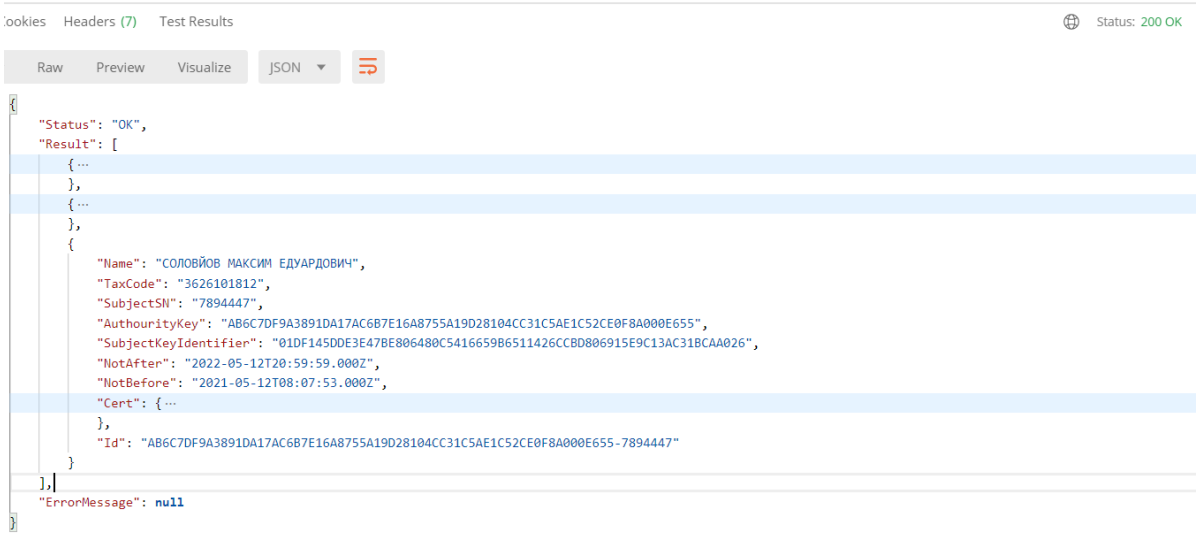

Рис. 3.5 Відповідь на запит ключів

Але якщо користувач обирає «Ні, відмінити» тоді буде надіслана відповідь з HTTP статусом 403 (рис. 3.6).

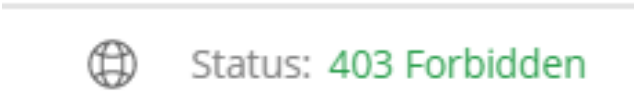

Рис. 3.6 Відхилений запит

Цей API сервіс при позитивній відповіді повертає список ключів користувача. Кожен ключ містить в собі інформацію, а саме:

– Name – назва ключа;
– TaxCode – ІПН власника ключа;
– SubjectSN – серійний номер ключа;
– AuthourityKey – ідентифікатор АЦСК;
– NotAfter – дата до якої дійсний ключ;
– NotBefore – дата з якої дійсний ключ;
– Id – унікальний ідентифікатор ключа.

API сервіс для отримання підпису доступний за адресою localhost:3031/api/sign POST запит (рис 3.7) та приймає json об’єкт.

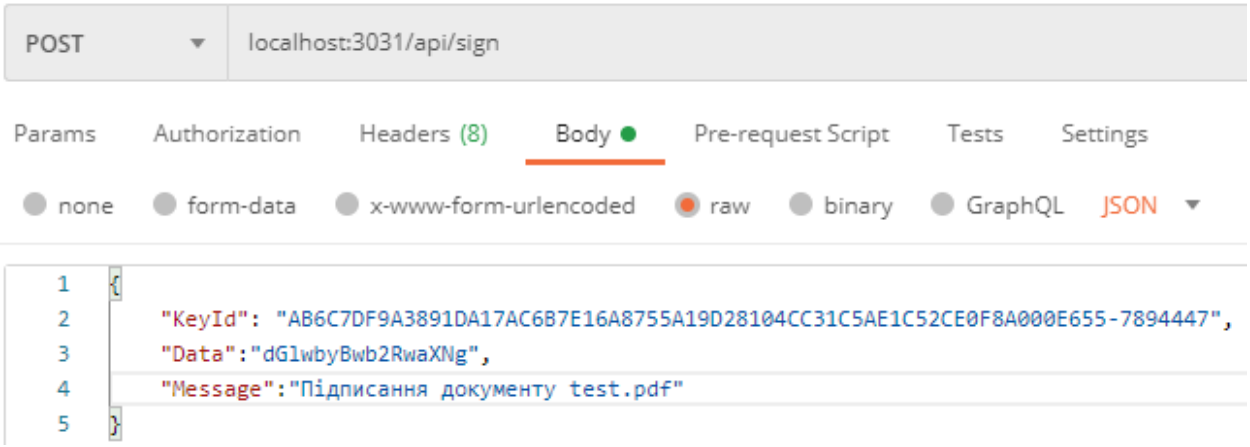

Рис. 3.7 Запит на підпис

JSON об’єкт має 3 властивості, а саме:

- KeyId – унікальний ідентифікатор ключа яким потрібно підписати дані;
- Data – дані в форматі base64 які потрібно підписати;
- Message – текст повідомлення який буде показано користувачу.

Після того як запит було відправлено, користувачу буде показане повідомлення на підтвердження підпису (рис. 3.8).

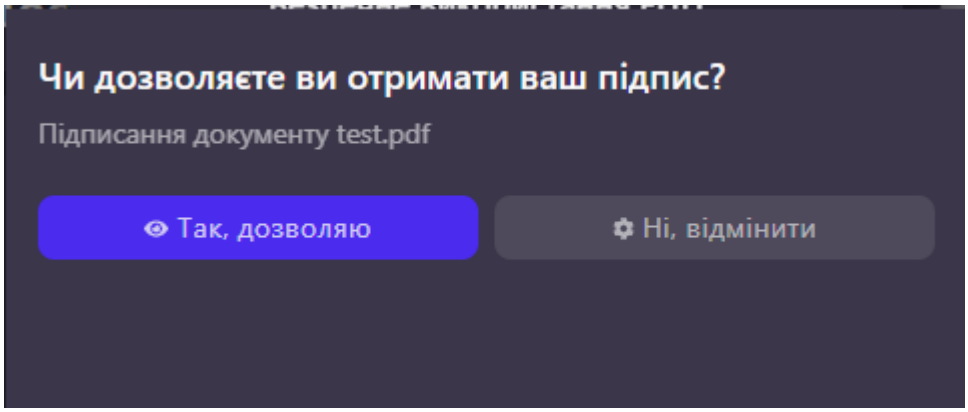

Рис. 3.8 Підтвердження підпису

Якщо користувач обирає «Так, дозволяю» тоді додаток пробує підписати ключем на захищеному носії. Але якщо користувач не вводив пароль до цього, або час кешування паролю вичерпано, тоді користувачу буде показане вікно для вводу паролю від захищеного носія (рис. 3.9).

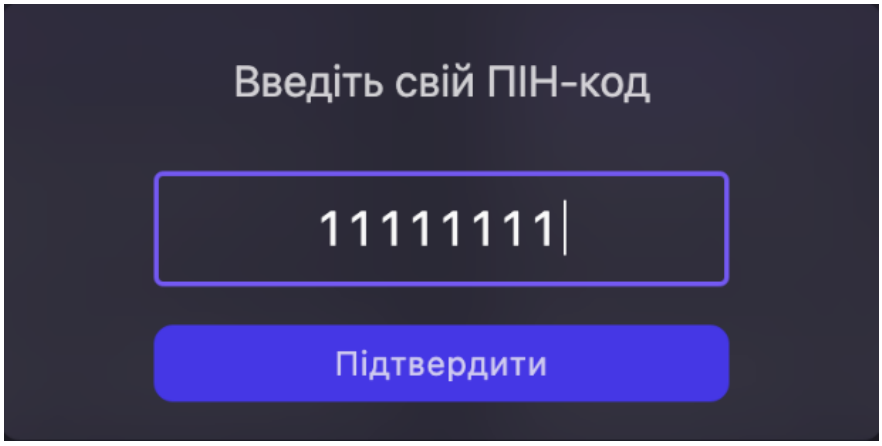

Рис. 3.9 Вікно вводу паролю

Після того як, користувач ввів вірний пароль, йому буде показане повідомлення про успішне накладання підпису (рис. 3.10).

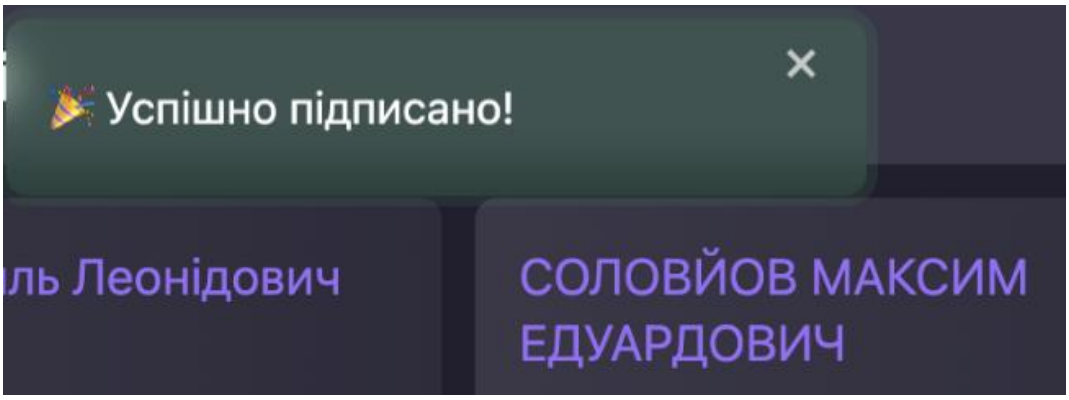

Рис. 3.10 Повідомлення про успішний підпис

Третій системі, яка запитувала підпис, буде поверненні підписані дані (рис. 3.11).

Рис. 3.11 Підписанні дані

Підписання даних можливо використовувати для ідентифікації користувача в системах які розробили інтеграцію з цим додатком.

3.5 Інструкція з використання програмного забезпечення

У користувача є можливість запуску додатку двома способами, перший – це портативний запуск з USB – накопичувача або зовнішній жорсткий диск (рис. 3.12).

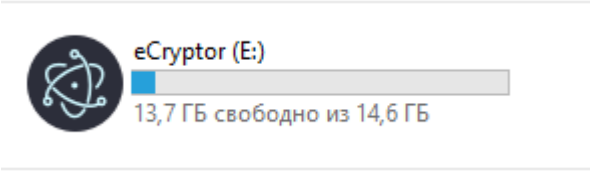

Рис. 3.12 Додаток на USB-накопичувачі

Другий варіант – це встановити інсталятором собі на комп’ютер (рис. 3.13).

Рис. 3.13 Встановлений додаток на комп’ютер

Після того як користувач запустить додаток, система відобразить йому привітальне вікно, з поясненням що це за додаток (рис. 3.14).

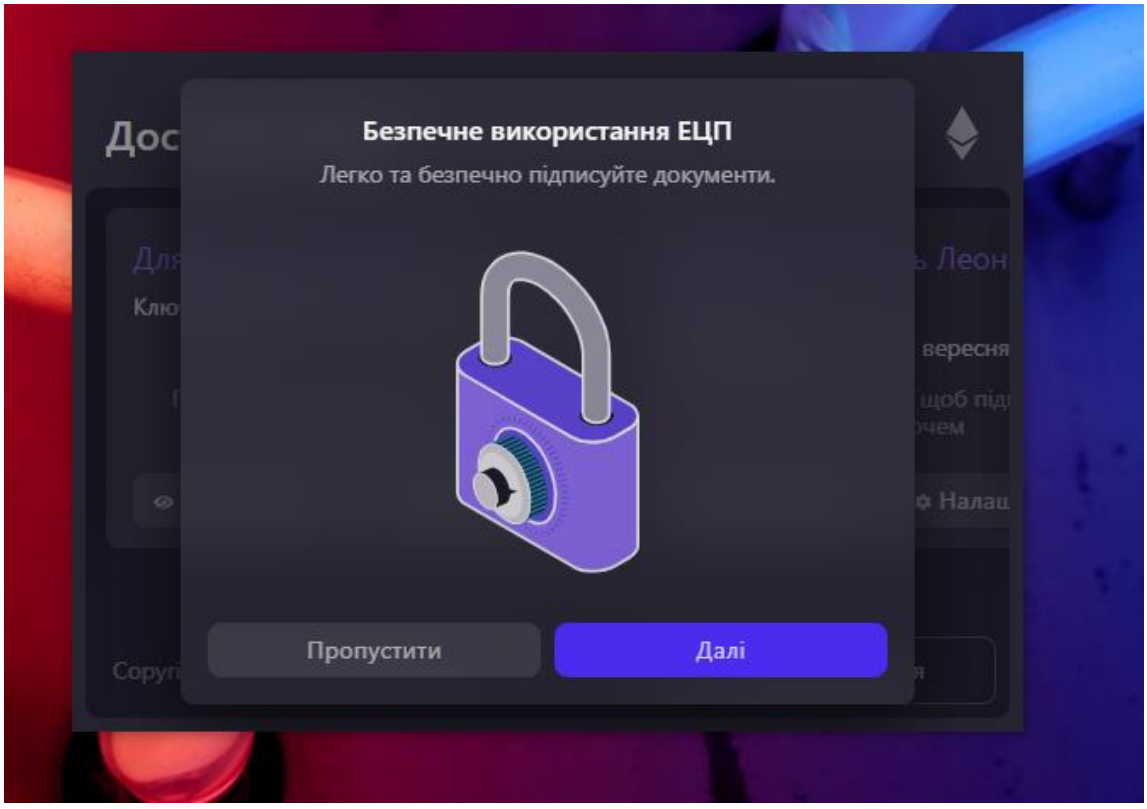

Рис. 3.14 Привітальне вікно

Коли користувач ознайомився з додатком, на головному екрані він побачить список своїх ключів (рис. 3.15).

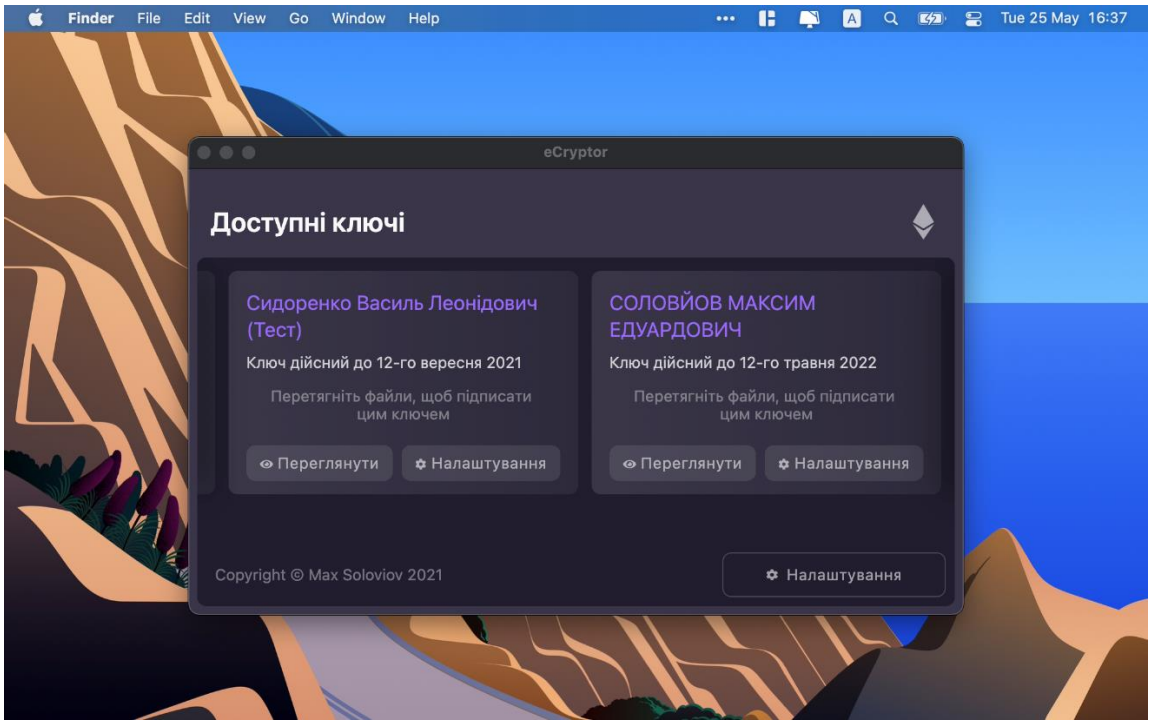

Рис. 3.15 Список ключів користувача

Як видно з наведеного вище рисунку, додаток успішно запускається на MacOS.

Наступною сторінкою користувачу потрібно зайти в розділ «Налаштування» (рис. 3.16).

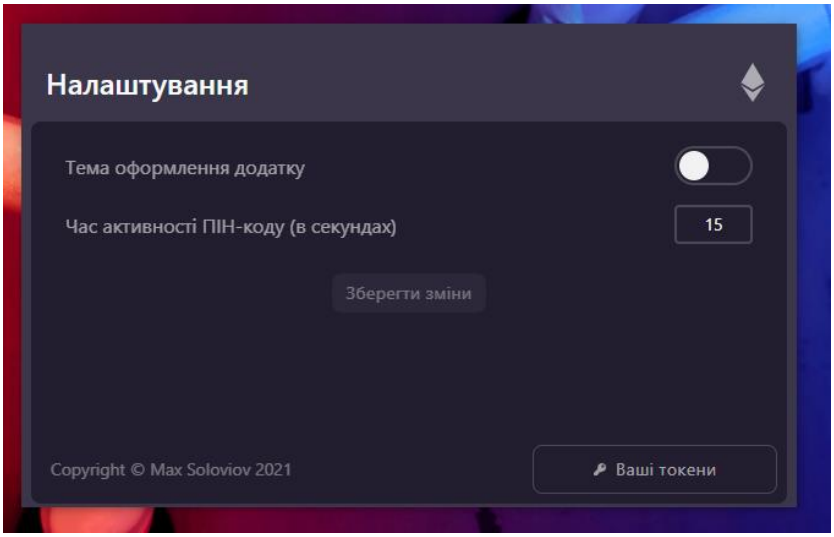

Рис. 3.16 Розділ налаштувань

В цьому розділі у користувача є можливість налаштувати тему додатка та час кешування паролю.

У користувача є вибір між темною темою (рис. 3.17) та світлою темою (рис. 3.18).

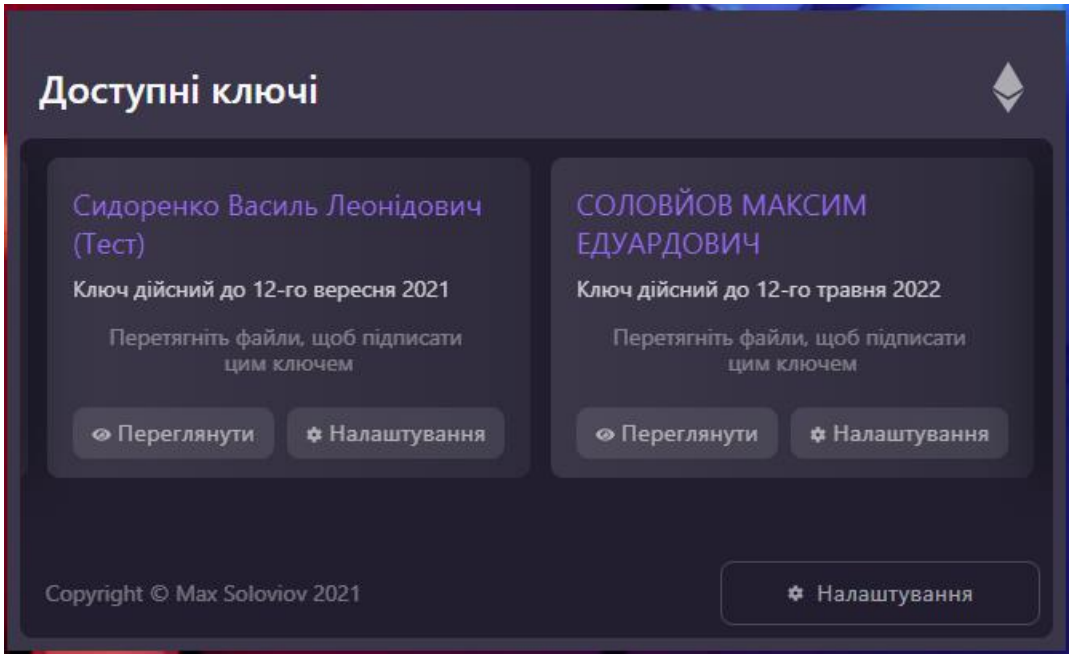

Рис. 3.17 Додаток в темній темі

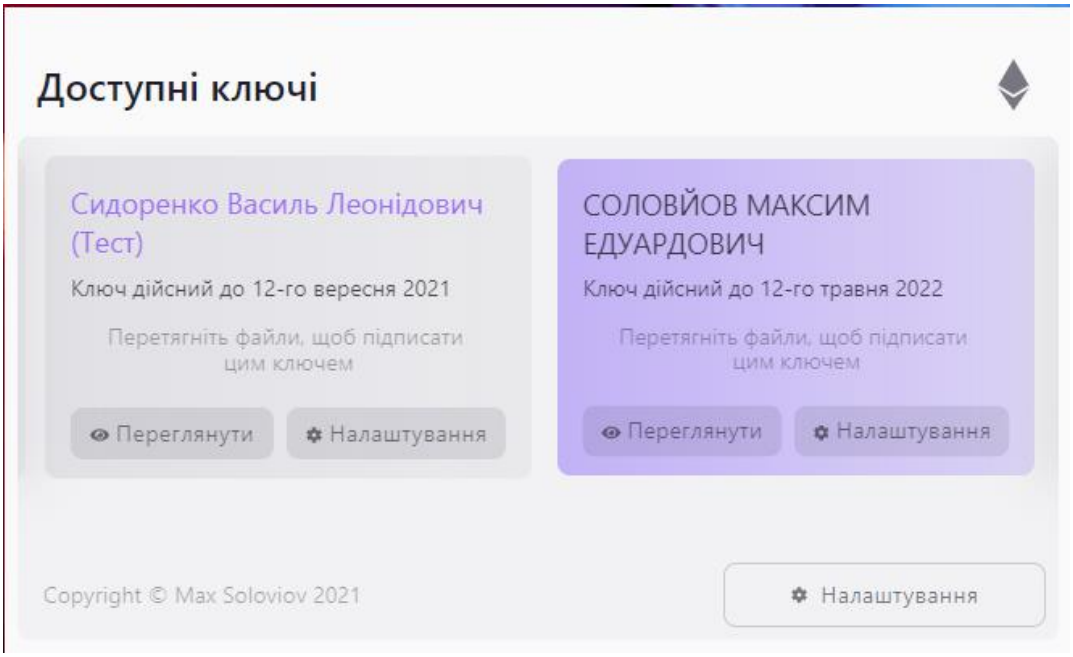

Рис. 3.18 Додаток в світлій темі

Час активності пін-коду – це час на який додаток запам’ятовує пароль користувача, що дозволить не вводити його кожний раз як потрібно накласти підпис, наприклад потрібно підписати 20 документів.

На відмінну від інших подібних додатків, у користувача є можливість підписати документ прямо в цьому ж додатку, все що йому потрібно це перенести файл на ключ, яким він хоче підписати документ (рис. 3.19).

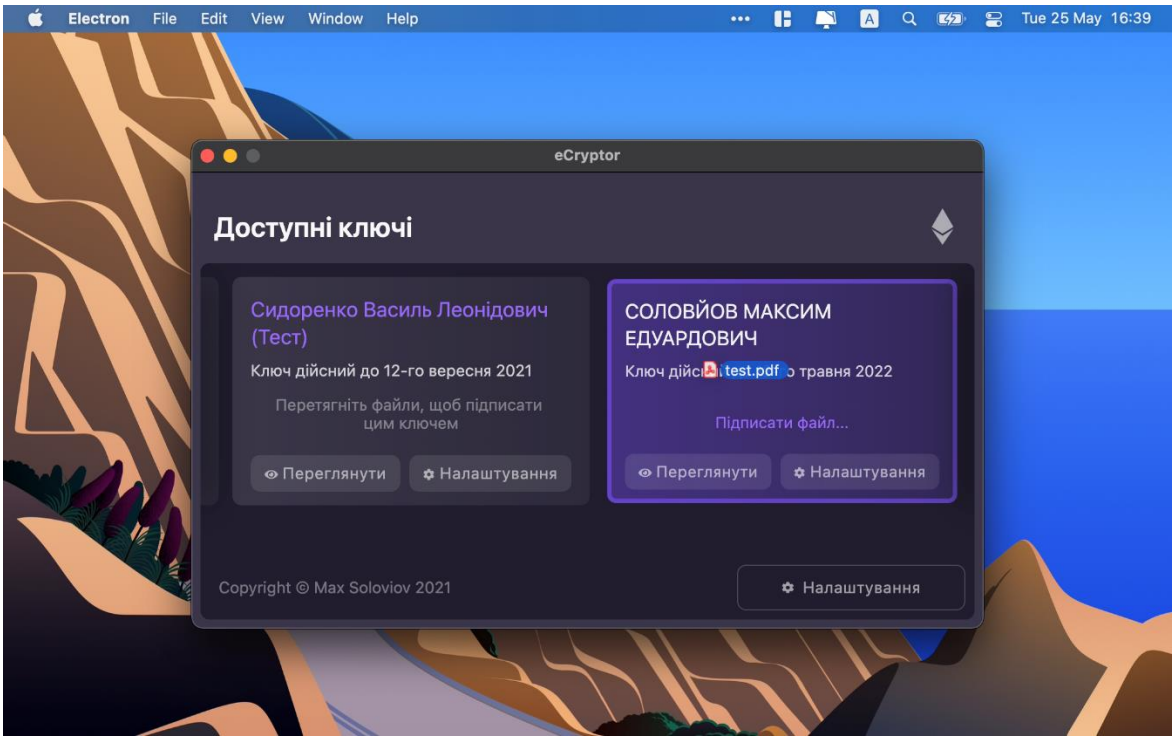

Рис. 3.19 Підписання документу

Якщо користувач не вводив пароль до цього, або час кешування паролю вичерпано, тоді користувачу буде показане вікно для вводу паролю від захищеного носія (рис. 3.20).

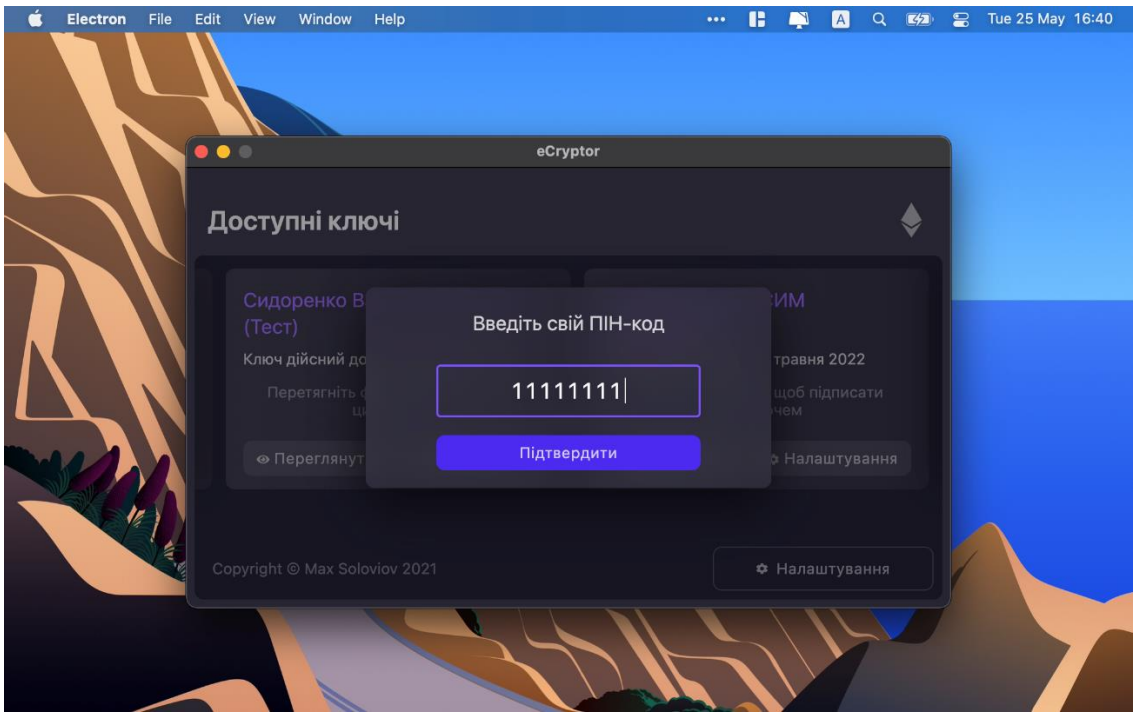

Рис. 3.20 Введення паролю від ключа

Після успішного вводу паролю, документ буде підписано, і користувачу буде запропоновано зберегти підписаний документ, а також буде відображене повідомлення про успішний підпис (рис. 3.21).

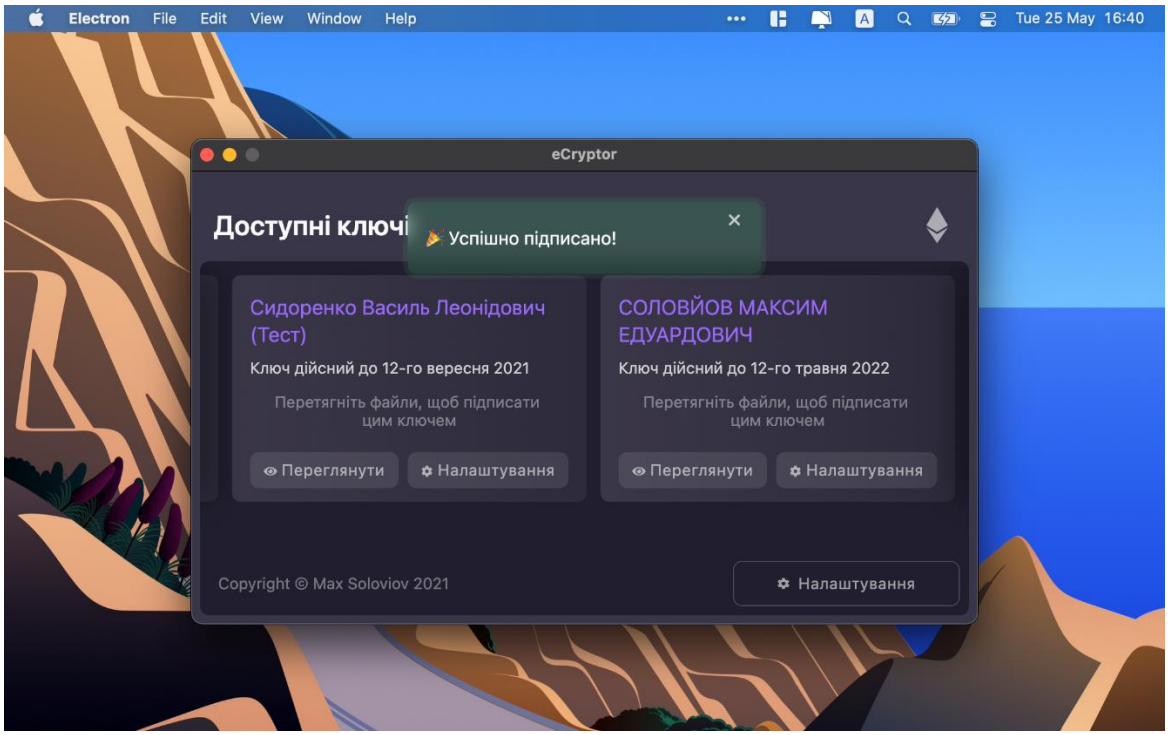

Рис. 3.21 Успішний підписаний документ

Для перевірки підпису, користувач повинен перейти на сайт ЦЗО (центрально засвідчуваний орган) та завантажити підписаний документ на сайт. Коли файл буде перевірено користувач побачить повідомлення про успішну перевірку (рис. 3.22).

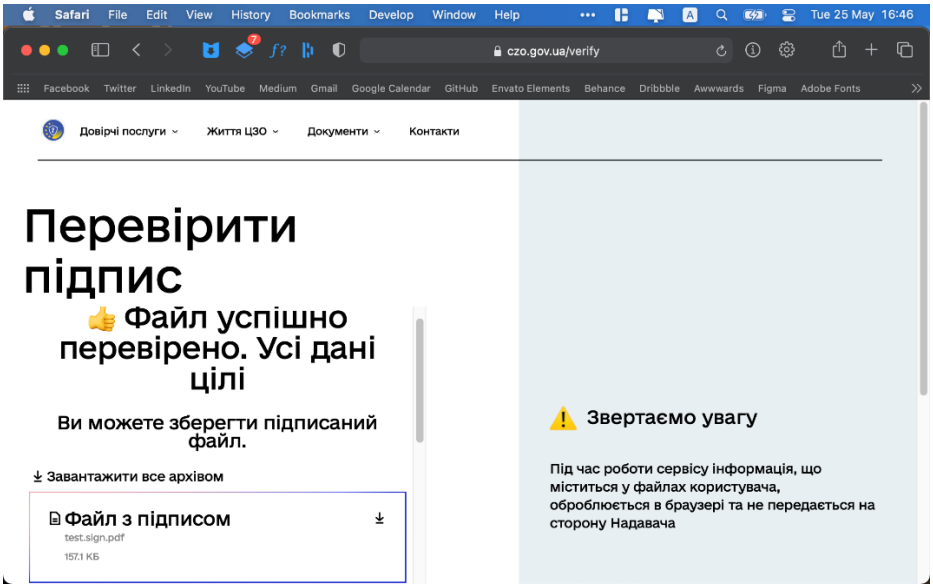

Рис. 3.22 Перевірка підпису

ЦЗО також надає інформацію про підписанта та мітку часу (коли було підписано документ). Користувач має можливість це продивитись спустившись нижче (рис. 3.23).

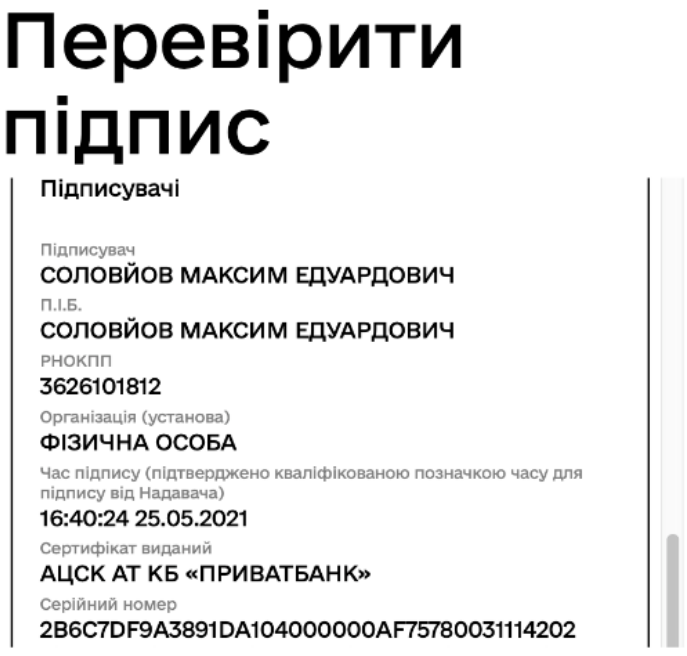

Рис. 3.23 Інформація про підписанта на мітка часу

ВИСНОВКИ

У процесі виконання дипломної роботи було досліджено електронний цифровий підпис та алгоритми його накладання. В ході роботи були проаналізовані поширені архітектурні рішення для розробки кросплатформенного програмного забезпечення, а згодом був розроблений додаток для роботи із захищеними носіями на основі PKCS11.

Мета бакалаврської роботи була досягнута та було вирішено такі завдання: досліджено теоретичні основи для роботи з електронним цифровим підписом, проаналізовано технологічні рішення щодо оптимального підходу до розробки кросплатформенного програмного забезпечення, розроблено кросплатформенне програмне забезпечення для роботи із захищеними носіями ЕЦП на основі PKCS11.

Розроблене програмне забезпечення призначене для накладання електронного цифрового підпису на файли. Він полегшую роботи із захищеним носієм. Показником для впровадження програмного забезпечення у використання є потрібність підписання документів та їх обіг у професійному, науковому, виробничому середовищі. Програмне забезпечення показало високу ступінь ефективності та має широкий спектр потенційного використання.

У майбутньому додаток може бути модифікований та покращений за рахунок додавання нових можливостей.

За допомогою розроблених API сервісів, є можливість інтеграції з третіми системами, для роботи із захищеними носіями користувачів системи таких як, Інтернет-банкінги, онлайн бухгалтерія.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Powers S. Learning Node / S. Powers. – S.: O`Reilly Media, 2012 – 374 с.

2. Роббинс Д. HTML5. Карманный справочник / Дженнифер Роббинс. – М.: Вильямс, 2015 – 192 с.

3. Начало работы с Express [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://metanit.com/web/nodejs/4.1.php

4. REST [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/REST

5. JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JavaScript

6. Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Node.js

7. User Interface style sheet language [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/User_interface_style_sheet_language

8. Смарт Н. Криптография. Москва: Техносфера, 2005. – 528 с.

9. Соловяненко Н.И. Юридическая роль электронной подписи в электронной коммерции. - М.: МЗ Пресс, 2002. - 215 с.

10. Венбо М. Современная криптография. Теория и практика // М.: Вильямс, 2005. — 768 с.

11. Юрасов, А.В. Основы электронной коммерции / А.В. Юрасов. - М.: Горячая линия - Телеком, 2017. - 480 с.

12. Гудман, Д. Java Script и HTML. Сборник для профессионалов / Д. Гудман. - М.: СПб: Питер, 2004. - 523 с.

13. Саломаа А. Криптография с открытым ключом. Пер. с англ. – М.: Мир, 1995. – 318 с

14. Халиков Р. Субъекты электронного документа в банковской сфере // Информационное право. - 2006. - 121 с.

15. Никольский А. П. JavaScript на примерах / А. П. Никольский. – Спб.: Наука и Техника, 2017 – 274 с.

16. Electron.js [Електронний ресурс] – Режим доступу до ресурсу: https://www.electronjs.org/

17. React.js [Електронний ресурс] – Режим доступу до ресурсу: https://reactjs.org/

ДОДАТКИ

ДОДАТОК А

Приклад реалізації сервісу для отримання інформації про ключі

```
public class KeysService
    {
        private string ToHex(byte[] bytes)
        {
            var hexString = BitConverter.ToString(bytes);
            hexString = hexString.Replace("-", "");
            return hexString;
        }

        public virtual void Login(LoginRequest loginRequest)
        {
            var keyInfo = GetKeyInfo(loginRequest.CertId);
            var slot = keyInfo.Slot;

            using (var session = slot.OpenSession(SessionType.ReadWrite))
            {
                session.SavePin(loginRequest);
            }
        }
        public virtual IEnumerable<CertInfo> GetKeys()
        {
            var libraryPath = DllHelper.GetPathToDll();
            try
            {
                Logger.Log($"Library path = {libraryPath}");
                var result = new List<CertInfo>();
                Pkcs11InteropFactories factories = new Pkcs11InteropFactories();
                var test = factories.Pkcs11LibraryFactory.LoadPkcs11Library(factories,
libraryPath, AppType.MultiThreaded);
                var slots = test.GetSlotList(SlotsType.WithOrWithoutTokenPresent);
                Logger.Log($"Get slots = [{JsonConvert.SerializeObject(slots)}]");
                foreach (var slot in slots)
                {
                    try
                    {
                        using (ISession session = slot.OpenSession(SessionType.ReadWrite))
                        {
                            result.AddRange(session.GetCertificates().Select(p =>
CertHelper.GetInfo(p)));
                        }
                    }
                    catch (Exception ex)
                    {
                        Logger.Log($"slot={slot.SlotId} {ex.Message}");
                    }

                }

                return result;
            }
            catch (Exception ex)
            {
                Logger.Log($"Error get keys ex=[{ex.Message}] stackTracke=[{ex.StackTrace}]");
                throw new Exception($"Library path=[{libraryPath}] ex=[{ex.Message}]
stackTracke=[{ex.StackTrace}]");
            }
        }
        public virtual KeyInfo GetKeyInfo(string certId)
```

```
        {
            var libraryPath = DllHelper.GetPathToDll();
            Pkcs11InteropFactories factories = new Pkcs11InteropFactories();
            var test = factories.Pkcs11LibraryFactory.LoadPkcs11Library(factories, libraryPath,
AppType.MultiThreaded);
            var slots = test.GetSlotList(SlotsType.WithOrWithoutTokenPresent);
            foreach (var slot in slots)
            {
                try
                {
                    using (ISession session = slot.OpenSession(SessionType.ReadWrite))
                    {
                        var certificate = session.GetCertificates().Select(p =>
CertHelper.GetInfo(p)).FirstOrDefault(p => p.Id == certId);
                        if (certificate != null)
                        {
                            return new KeyInfo { CertInfo = certificate, Slot = slot };
                        }
                    }
                }
                catch (Exception ex)
                {
                    Logger.Log($"slot={slot.SlotId} {ex.Message}");
                }

            }

            return null;
 }
```

ДОДАТОК Б

Приклад реалізації сервісу для отримання підпису

```
public class SignService
    {
        public byte[] Sign(SignRequest signRequest)
        {
            var keyService = new KeysService();
            var keyInfo = keyService.GetKeyInfo(signRequest.CertId);

            var slot = keyInfo.Slot;
            var cert = new Certificate(keyInfo.CertInfo.Cert);

            //key.Factories.
            using (ISession session = slot.OpenSession(SessionType.ReadWrite))
            {
                session.LoginByCert(signRequest.CertId);

                byte[] attrValue;
                uint keySearchAttribute = extractKeySearchAttribute(cert.getPublicKeyInfo(), out
attrValue);
                uint publicKeyHandle = session.getPublicKeyHandle(keySearchAttribute, attrValue);
                uint bindedObjectHandle = session.getBindedObjectHandle(publicKeyHandle, 385U);

                var privateKey = new PrivateKeyDstu(session, bindedObjectHandle, null, false, -1,
(byte[])null);



                var cms = new CmsAdvanced(true, new InternationalAlgFactory());
                cms.addSigner(cert.getEncoded(), privateKey);
                cms.mustAttachSigningTime(true);

                /* передаем данные, если нужно - порциями */
                cms.update(Convert.FromBase64String(signRequest.DataToSign));

                string tspUri = cms.getSigner(0).getCertificate().getTSAServer(0) ??
"http://ca.oschadbank.ua/public/tsa";
                UriTransport transport = new HttpTransport("", 0, string.Empty, string.Empty);

                cms.ensureSigned(); // подписываем документ

                cms.setSignatureTimeStampServer(transport, tspUri);
                cms.ensureTimeStamped(); // добавляем метку времени

                cms.ensureSigned(); // подписываем документ



                /* подписываем и получаем конверт в виде массива байт */
                byte[] blob = cms.getEncoded();

                return blob;
            }
        }
        protected uint extractKeySearchAttribute(byte[] publicKeyInfo, out byte[] attrValue)
        {
            string oidStr;
            Ddec.DerGetElementOid(publicKeyInfo, "SSD", out oidStr);
            byte[] OutBuf1;
            Ddec.DerGetElementBuf(publicKeyInfo, "SsG", out OutBuf1);
            if (oidStr.Equals("1.2.840.113549.1.1.1"))
```

```
        {
            byte[] OutBuf2;
            if (Ddec.DerGetElementBuf(OutBuf1, "SI", out OutBuf2) != 0)
                throw new Exception("key store not found");
            if (OutBuf2.Length > 1 && OutBuf2[0] == (byte)0)
            {
                Array.Copy((Array)OutBuf2, 1, (Array)OutBuf2, 0, OutBuf2.Length - 1);
                Array.Resize<byte>(ref OutBuf2, OutBuf2.Length - 1);
            }
            attrValue = OutBuf2;
            return 288;
        }
        if (oidStr.Equals("1.2.804.2.1.1.1.1.3.1.1.1.1"))
        {
            byte[] numArray = new byte[OutBuf1.Length - 2];
            Array.Copy((Array)OutBuf1, 2, (Array)numArray, 0, numArray.Length);
            Array.Reverse((Array)numArray);
            Array.Copy((Array)numArray, 0, (Array)OutBuf1, 2, numArray.Length);
        }
        attrValue = OutBuf1;
        return 385;
    }
}
```

ДОДАТОК В

Приклад реалізації класу формування підпису

```
public class CmsAdvanced : CmsSigned
    {
        protected UriTransport timestampTransport;
        protected string timestampUri;
        protected bool wantTspCert;
        protected bool addESSsgCertDNSN;
        protected bool convertingTsu;
        protected bool tsuValidationSelftimed;
        protected bool preferOcspCAdES;
        private CryptoAlgorithmFactory advAlgs;
        private byte[] encodedSgPolicy;
        private bool attachSigningTime;

        /// <summary>Создает новый подписанный конверт</summary>
        /// <param name="attached">True, если содержимое документа должно быть включено внутрь
конверта</param>
        /// <param name="algs">Фабрика алгоритмов</param>
        public CmsAdvanced(bool attached, CryptoAlgorithmFactory algs)
          : base(attached)
        {
            this.attachSigningTime = false;
            this.timestampTransport = (UriTransport)null;
            this.wantTspCert = true;
            this.addESSsgCertDNSN = true;
            this.convertingTsu = true;
            this.tsuValidationSelftimed = false;
            this.preferOcspCAdES = false;
            this.advAlgs = algs;
        }

        /// <summary>Загружает подписанный конверт из двоичных данных</summary>
        /// <param name="derData">Входной блок данных</param>
        /// <param name="algs">Фабрика алгоритмов</param>
        public CmsAdvanced(byte[] derData, CryptoAlgorithmFactory algs)
          : base(derData)
        {
            this.attachSigningTime = false;
            this.timestampTransport = (UriTransport)null;
            this.wantTspCert = true;
            this.addESSsgCertDNSN = true;
            this.convertingTsu = true;
            this.tsuValidationSelftimed = false;
            this.preferOcspCAdES = false;
            this.advAlgs = algs;
        }

        /// <summary>Начать проверку подписи документа</summary>
        /// <remarks>
        /// Поисковик сертификатов требуется если сертификаты подписчиков не включены в конверт.
При использовании данного класса
        /// для формирования конверта, сертификаты включаются автоматически
        /// </remarks>
        /// <param name="certFinder">Поисковик сертификатов или null</param>
        public void verifyBegin(CertificateFinder certFinder)
        {
            this.verifyBegin(this.advAlgs, certFinder);
        }

        /// <summary>Добавляет подписчика к конверту</summary>
        /// <param name="derCert">Сертификат подписчика в der-кодированном виде</param>
```

```
        /// <param name="clonedPrivKey">Личный ключ подписи</param>
        /// <returns>Информация о подписчике</returns>
        public new SignerInfo addSigner(byte[] derCert, PrivateKey clonedPrivKey)
        {
            SignerInfo signerInfo = base.addSigner(derCert, clonedPrivKey);
            this.addCertificate(derCert);
            return signerInfo;
        }

        /// <summary>Добавить к документу политику подписи</summary>
        /// <param name="derSignaturePolicyId">DER-кодированный идентификатор политики</param>
        public void setSignaturePolicy(byte[] derSignaturePolicyId)
        {
            this.encodedSgPolicy = derSignaturePolicyId;
        }

        /// <summary>Добавить к документу политику подписи</summary>
        /// <param name="sigPolicyId">OID политики</param>
        /// <param name="uri">Адрес политики</param>
        public void setSignaturePolicyUri(string sigPolicyId, string uri)
        {
            this.setSignaturePolicy(new SignaturePolicy(sigPolicyId, uri).getEncoded());
        }

        /// <summary>
        /// Получить идентификатор политики, заявленный подписчиком
        /// </summary>
        /// <param name="signerIx">Индекс подписчика</param>
        /// <returns>DER-кодированный идентификатор политики</returns>
        public byte[] getSignaturePolicy(int signerIx)
        {
            SignerInfo signer = this.getSigner(signerIx);
            if (signer == null)
                throw new Exception();
            if (signer.isSignedAttributePresent("1.2.840.113549.1.9.16.2.15"))
                return signer.getSignedAttribute("1.2.840.113549.1.9.16.2.15");
            return (byte[])null;
        }

        /// <summary>Добавлять время, заявленное подписчиком</summary>
        /// <remarks>
        /// Добавляет время момента подписи, установленное на компьютере подписчика. Для
добавления доверенного времени
        /// следует использовать функции setSignatureTimeStampServer() и ensureTimeStamped()
        /// </remarks>
        /// <param name="wantAttach">true - добавлять время</param>
        public void mustAttachSigningTime(bool wantAttach)
        {
            this.attachSigningTime = wantAttach;
        }

        /// <summary>
        /// Установить тип сохраняемых данных при формировании документов долговременного
хранения
        /// </summary>
        ///
        ///             По умолчанию сохраняются записи CRL. Функция позволяет установить режим
сохранения ответов OCSP-сервера вместо CRL.
        ///             Ответы OCSP будут сохраняться только если адрес OCSP-сервера указан в
сертификате.
        ///             <param name="preferOcsp">true - сохранять ответы OCSP</param>
        public void setPreferOcspCAdES(bool preferOcsp)
        {
            this.preferOcspCAdES = preferOcsp;
```

```
        }

        /// <summary>
        /// Установить режим проверки сертификата метки времени для документов долговременного
хранения.
        /// </summary>
        ///
        ///             По умолчанию сертификат метки времени будет проверен на текущий момент.
Функция позволяет установить режим,
        ///             при котором сертификат метки времени будет проверяться на момет
постановки метки. Режим требует определенных политик
        ///             работы с такими сертификатами.
        ///             <param name="selfTimed">true - если сертификат метки времени не
устаревает</param>
        public void setTimeStampModeAsSelftimed(bool selfTimed)
        {
            this.tsuValidationSelftimed = selfTimed;
        }

        /// <summary>Установить адрес сервера меток времени (TSP-сервера)</summary>
        /// <remarks>
        /// Если установлен адрес TSP-сервера, то к документу будет автоматически добавлена метка
времени во время подписи.
        /// Если требуется добавить метку времени к уже сформированному конверту, то
дополнительно следует вызвать функцию ensureTimeStamped().
        /// Установка адреса NSP-сервера также потребуется при преобразовании документа в
документ долговременного хранения.
        /// </remarks>
        /// <param name="transport">Транспорт сервера, обычно экземпляр HttpTransport</param>
        /// <param name="uri">Адрес сервера</param>
        public void setSignatureTimeStampServer(UriTransport transport, string uri)
        {
            this.timestampTransport = transport;
            if (uri == null)
                return;
            this.timestampUri = uri;
        }

        /// <summary>
        /// Добавить метку времени всем подписчикам, если метки еще нет
        /// </summary>
        public void ensureTimeStamped()
        {
            this.ensureSigned();
            for (int index = 0; index < this.getSignerCount(); ++index)
            {
                SignerInfo signer = this.getSigner(index);
                if (!signer.isUnsignedAttributePresent("1.2.840.113549.1.9.16.2.14"))
                {
                    DateTime stampingTime;
                    this.addSignatureTimeStamp(signer, out stampingTime);
                }
            }
        }

        /// <summary>Получить значение метки времени указанного подписчика</summary>
        /// <param name="signerIx">Индекс подписчика</param>
        /// <returns>Дата и время момента добавления метки времени</returns>
        public DateTime getStampingTime(int signerIx)
        {
            if (this.getSigner(signerIx) == null)
                throw new Exception();
            DateTime timestampTm = this.signers[signerIx].timestampTm;
            if (timestampTm == DateTime.MinValue)
```

```
                throw new Exception();
            return timestampTm;
             }
/// <summary>Проверить метку времени и валидность ее сертификата</summary>
        /// <remarks>
        /// Обычно не требуется вызывать эту функцию непосредственно, вызов происходит
автоматически при вызове validateAllAtStampingTime().
        /// </remarks>
        /// <param name="signerIx">Индекс подписчика</param>
        /// <param name="certCache">Итерфейс хранилища сертификатов</param>
        /// <param name="validationPolicy">Политика проверки</param>
        /// <param name="tsTime">Значение метки времени</param>
        /// <param name="status">Статус поверки (константы CVS_x)</param>
        /// <returns>True, если подпись метки верна и ее сертификат действительный</returns>
        public bool validateTimeStamp(
          int signerIx,
          CertificateValidatorCache certCache,
          CertificateValidationPolicy validationPolicy,
          out DateTime tsTime,
          out int status)
        {
            Certificate cert = this.verifyTimeStamp(signerIx, certCache, out tsTime, out status);
            if (cert == null || !new CertificatePathValidator(this.advAlgs,
certCache).validate(cert, "1.3.6.1.5.5.7.3.8", DateTime.MinValue, validationPolicy, out status))
                return false;
            this.signers[signerIx].timestampTm = tsTime;
            return true;
        }

        /// <summary>
        /// Проверить сертификат подписчика на момент постановки метки времени
        /// </summary>
        /// <remarks>
        /// Обычно функция вызывается автоматически при вызове validateAllAtStampingTime
        /// </remarks>
        /// <param name="signerIx">Индекс подписчика</param>
        /// <param name="mustTimeStamped">True, если отсутствие метки времени является ошибкой,
иначе используется текущее время</param>
        /// <param name="intendedUsage">OID расширенного использования ключа</param>
        /// <param name="certCache">Итерфейс хранилища сертификатов</param>
        /// <param name="validationPolicy">Политика проверки</param>
        /// <param name="status">Статус поверки (константы CVS_x)</param>
        /// <returns>True, если сертификат подписчика действительный</returns>
        public bool validateSignerAtStampingTime(
          int signerIx,
          bool mustTimeStamped,
          string intendedUsage,
          CertificateValidatorCache certCache,
          CertificateValidationPolicy validationPolicy,
          out int status)
        {
            status = 5;
            SignerInfo signer = this.getSigner(signerIx);
            if (signer == null)
                throw new Exception();
            DateTime tsTime = this.signers[signerIx].timestampTm;
            if (tsTime == DateTime.MinValue)
            {
                if (signer.isUnsignedAttributePresent("1.2.840.113549.1.9.16.2.14"))
                {
                    if (!this.validateTimeStamp(signerIx, certCache, validationPolicy, out
tsTime, out status))
                        return false;
                }
```

```
                else
                {
                    if (mustTimeStamped)
                        throw new Exception();
                    tsTime = DateTime.UtcNow;
                }
            }
            CertificatePathValidator certificatePathValidator = new
CertificatePathValidator(this.advAlgs, certCache);
            CertstoreCms certstoreCms = new CertstoreCms((CmsSigned)this);
            return certificatePathValidator.validate(signer.getCertificate(), intendedUsage,
tsTime, validationPolicy, (CertificateFinder)certstoreCms, out status);
        }

        /// <summary>
        /// Проверить сертификаты подписчиков на момент постановки метки времени
        /// </summary>
        /// <param name="mustTimeStamped">True, если отсутствие метки времени является ошибкой,
иначе используется текущее время</param>
        /// <param name="intendedUsage">OID расширенного использования ключа</param>
        /// <param name="certCache">Итерфейс хранилища сертификатов</param>
        /// <param name="validationPolicy">Политика проверки</param>
        /// <param name="status">Статус поверки (константы CVS_x)</param>
        /// <returns>True, если все сертификаты подписчиков действительны</returns>
        public bool validateAllAtStampingTime(
          bool mustTimeStamped,
          string intendedUsage,
          CertificateValidatorCache certCache,
          CertificateValidationPolicy validationPolicy,
          out int status)
        {
            if (this.getSignerCount() < 1)
                throw new Exception();
            status = 0;
            for (int signerIx = 0; signerIx < this.getSignerCount(); ++signerIx)
            {
                if (!this.validateSignerAtStampingTime(signerIx, mustTimeStamped, intendedUsage,
certCache, validationPolicy, out status))
                    return false;
            }
            return true;
              }
/// <summary>
        /// Проверить сертификат подписчика в документе долговременного хранения
        /// </summary>
        /// <remarks>
        /// Для проверки целостности документа следует использовать функции verifyBegin(),
verifyUpdate(), verifyAll().
        /// </remarks>
        /// <param name="signerIx">Индекс подписчика</param>
        /// <param name="intendedUsage">OID расширенного использования ключа</param>
        /// <param name="certCache">Итерфейс хранилища сертификатов</param>
        /// <param name="validationPolicy">Политика проверки</param>
        /// <param name="status">Статус поверки (константы CVS_x)</param>
        /// <returns>True, если сертификат подписчика действительный</returns>
        public bool validateSignerCAdES_C(
          int signerIx,
          string intendedUsage,
          CertificateValidatorCache certCache,
          CertificateValidationPolicy validationPolicy,
          out int status)
        {
            status = 5;
            SignerInfo signer1 = this.getSigner(signerIx);
```

```
            if (signer1 == null)
                throw new Exception();
            if (signer1.getCertificate() == null)
                throw new Exception();
            if (!signer1.isUnsignedAttributePresent("1.2.840.113549.1.9.16.2.14"))
                throw new Exception();
            DateTime tsTime;
            Certificate cert = this.verifyTimeStamp(signerIx, certCache, out tsTime, out status);
            if (cert == null)
                return false;
            SignerInfo signer2 = new
CmsAdvanced(signer1.getUnsignedAttribute("1.2.840.113549.1.9.16.2.14"),
this.advAlgs).getSigner(0);
            bool flag;
            if (signer2.isUnsignedAttributePresent("1.2.840.113549.1.9.16.2.21"))
            {
                DateTime tm = DateTime.MinValue;
                if (this.tsuValidationSelftimed)
                    tm = tsTime;
                flag = this.validateCertCAdEStp(cert, certCache, signer2, "1.3.6.1.5.5.7.3.8",
tm, validationPolicy, out status);
            }
            else
                flag = new CertificatePathValidator(this.advAlgs, certCache).validate(cert,
"1.3.6.1.5.5.7.3.8", DateTime.MinValue, validationPolicy, out status);
            if (!flag)
                return false;
            this.signers[signerIx].timestampTm = tsTime;
            return this.validateCertCAdEStp(signer1.getCertificate(), certCache, signer1,
intendedUsage, tsTime, validationPolicy, out status);
        }

        /// <summary>
        /// Проверить сертификаты подписчиков в документе долговременного хранения
        /// </summary>
        /// <remarks>
        /// Для проверки целостности документа следует использовать функции verifyBegin(),
verifyUpdate(), verifyAll().
        /// </remarks>
        /// <param name="intendedUsage">OID расширенного использования ключа или null</param>
        /// <param name="certCache">Итерфейс хранилища сертификатов</param>
        /// <param name="validationPolicy">Политика проверки</param>
        /// <param name="status">Статус поверки (константы CVS_x)</param>
        /// <returns>True, если все сертификаты подписчиков действительны</returns>
        public bool validateAllCAdES_C(
          string intendedUsage,
          CertificateValidatorCache certCache,
          CertificateValidationPolicy validationPolicy,
          out int status)
        {
            if (this.getSignerCount() < 1)
                throw new Exception();
            status = 0;
            for (int signerIx = 0; signerIx < this.getSignerCount(); ++signerIx)
            {
                if (!this.validateSignerCAdES_C(signerIx, intendedUsage, certCache,
validationPolicy, out status))
                    return false;
            }
            return true;
        }

        /// <summary>
```

```
        /// Преобразовать подписанный документ в документ долговременного хранения без включения
полных данных проверки
        /// </summary>
        /// <remarks>
        /// <para>Перед вызовом этой функции следует выполнить проверку содержимого тела
документа, даже если известно, что документ
        /// не модифицирован. Также требуется предварительно установить адрес TSP-сервера.</para>
        /// <para>Функция может использоваться, если создана инфраструктура для хранения и
архивирования сертификатов и CRL.
        /// В других случаях рекомендуется использовать функцию convertToCAdES_X()</para>
        /// </remarks>
        /// <param name="certCache">Итерфейс хранилища сертификатов</param>
        /// <param name="validationPolicy">Политика проверки</param>
        public void convertToCAdES_C(
          CertificateValidatorCache certCache,
          CertificateValidationPolicy validationPolicy)
        {
            this.convertToCAdEStp(certCache, validationPolicy, false);
             }
/// <summary>
        /// Преобразовать подписанный документ в документ долговременного хранения с включением
полных данных проверки
        /// </summary>
        /// <remarks>
        /// <para>Перед вызовом этой функции следует выполнить проверку содержимого тела
документа, даже если известно, что документ
        /// не модифицирован. Также требуется предварительно установить адрес TSP-сервера.</para>
        /// <param name="certCache">Итерфейс хранилища сертификатов</param>
        /// <param name="validationPolicy">Политика проверки</param>
        public void convertToCAdES_X(
          CertificateValidatorCache certCache,
          CertificateValidationPolicy validationPolicy)
        {
            this.convertToCAdEStp(certCache, validationPolicy, true);
        }

        protected override void beforeSigning(CmsSigned.CmsSignerInfo csi)
        {
            SignerInfo info = csi.info;
            this.addSigningCertificateId(info);
            if (this.encodedSgPolicy != null)
                info.addSignedAttribute("1.2.840.113549.1.9.16.2.15", this.encodedSgPolicy,
this.encodedSgPolicy.Length);
            if (!this.attachSigningTime)
                return;
            DerEncoder derEncoder = new DerEncoder();
            derEncoder.addTime(DateTime.UtcNow);
            info.addSignedAttribute("1.2.840.113549.1.9.5", derEncoder.getEncoded(),
derEncoder.getSize());
        }

        protected override void afterSigning(CmsSigned.CmsSignerInfo csi)
        {
            SignerInfo info = csi.info;
            if (this.timestampTransport == null)
                return;
            DateTime stampingTime;
            this.addSignatureTimeStamp(info, out stampingTime);
        }

        protected override int afterVerificationSuccess(CmsSigned.CmsSignerInfo csi)
        {
            if (!csi.info.isSignedAttributePresent("1.2.840.113549.1.9.16.2.47") &&
!csi.info.isSignedAttributePresent("1.2.840.113549.1.9.16.2.12"))
```

```
                return 6;
            return base.afterVerificationSuccess(csi);
        }

        protected void addSignatureTimeStamp(SignerInfo si, out DateTime stampingTime)
        {
            if (this.timestampTransport == null)
                throw new Exception();
            TimeStampRequest timeStampRequest = new TimeStampRequest(si.getDigestOid(),
si.getDigestParams(), this.advAlgs);
            timeStampRequest.reqCertificate(this.wantTspCert);
            byte[] signature = si.getSignature();
            timeStampRequest.update(signature);
            byte[] encoded = timeStampRequest.getEncoded();
            TimeStampResponse timeStampResponse = new
TimeStampResponse(this.timestampTransport.post(this.timestampUri, encoded,
"application/timestamp-query", (string)null));
            if (timeStampResponse.getStatus() != 0)
                throw new MyException((uint)timeStampResponse.getStatus());
            if (!timeStampResponse.compareWithRequest(encoded))
                throw new MyException(2U);
            stampingTime = timeStampResponse.getGenerationTime();
            byte[] cms = timeStampResponse.getCms();
            si.addUnsignedAttribute("1.2.840.113549.1.9.16.2.14", cms, cms.Length);
        }

        protected Certificate verifyTimeStamp(
          int signerIx,
          CertificateValidatorCache certCache,
          out DateTime tsTime,
          out int status)
        {
            status = 5;
            tsTime = DateTime.MinValue;
            SignerInfo signer = this.getSigner(signerIx);
            if (signer == null)
                throw new Exception();
            if (!signer.isUnsignedAttributePresent("1.2.840.113549.1.9.16.2.14"))
                throw new Exception();
            TimeStampResponse timeStampResponse = new
TimeStampResponse(signer.getUnsignedAttribute("1.2.840.113549.1.9.16.2.14"));
            timeStampResponse.compareWithDocBegin((string)null, this.advAlgs);
            timeStampResponse.compareWithDocUpdate(signer.getSignature());
            if (!timeStampResponse.compareWithDocFinal())
            {
                status = 5;
                return (Certificate)null;
            }
            CmsSigned cmsSigned = new CmsSigned(timeStampResponse.getCms());
            cmsSigned.verifyBegin(this.advAlgs, certCache.getCertificateFinder());
            if (!cmsSigned.verifyAll())
            {
                status = cmsSigned.getSigner(0).getVerificationStatus();
                return (Certificate)null;
            }
            Certificate certificate = cmsSigned.getSigner(0).getCertificate();
            if (!certificate.isEkuPresent("1.3.6.1.5.5.7.3.8"))
            {
                status = -12;
                return (Certificate)null;
            }
            tsTime = timeStampResponse.getGenerationTime();
            return certificate.clone();
        }
```

```
        private void addSigningCertificateId(SignerInfo si)
        {
            string digestOid = si.getDigestOid();
            bool flag = digestOid != "1.3.14.3.2.26";
            byte[] encoded = si.getCertificate().getEncoded();
            MessageDigestAlg digestAlg = this.advAlgs.getDigestAlg(digestOid, (byte[])null,
(Certificate)null);
            if (digestAlg == null)
                throw new Exception();
            digestAlg.update(encoded);
            byte[] digest = digestAlg.getDigest();
            DerEncoder derEncoder = new DerEncoder();
            derEncoder.SeqBegin();
            derEncoder.SeqBegin();
            derEncoder.SeqBegin();
            if (flag)
            {
                derEncoder.SeqBegin();
                derEncoder.addOid(si.privateKey.getDigestOid());
                if (si.getDigestParams() != null)
                    derEncoder.addDerParams(si.privateKey.getDigestParams());
                derEncoder.SeqEnd();
            }
            derEncoder.addOctets(digest);
            if (this.addESSsgCertDNSN)
            {
                derEncoder.SeqBegin();
                derEncoder.SeqBegin();
                derEncoder.SpecificBegin((byte)164);
                derEncoder.addObject(si.getCertificate().getIssuerDN());
                derEncoder.SpecificEnd();
                derEncoder.SeqEnd();
                derEncoder.addObject(si.getCertificate().getSerial());
                derEncoder.SeqEnd();
            }
            derEncoder.SeqEnd();
            derEncoder.SeqEnd();
            derEncoder.SeqEnd();
            string oid = "1.2.840.113549.1.9.16.2.12";
            if (flag)
                oid = "1.2.840.113549.1.9.16.2.47";
            si.addSignedAttribute(oid, derEncoder.getEncoded(), derEncoder.getSize());
              }
private void convertToCAdEStp(
          CertificateValidatorCache certCache,
          CertificateValidationPolicy validationPolicy,
          bool withValues)
        {
            this.ensureSigned();
            for (int index = 0; index < this.getSignerCount(); ++index)
            {
                SignerInfo signer = this.getSigner(index);
                if (signer.getCertificate() == null)
                    throw new Exception();
                if (!signer.isUnsignedAttributePresent("1.2.840.113549.1.9.16.2.14"))
                    throw new Exception();
            }
            CertificatePathValidator certificatePathValidator = new
CertificatePathValidator(this.advAlgs, certCache, this.preferOcspCAdES);
            CollectReferencePolicy collectReferencePolicy = new
CollectReferencePolicy(validationPolicy, this.advAlgs);
            for (int index = 0; index < this.getSignerCount(); ++index)
            {
```

```
                SignerInfo signer1 = this.getSigner(index);
                DateTime tsTime;
                int num;
                if (!this.validateTimeStamp(index, certCache,
(CertificateValidationPolicy)collectReferencePolicy, out tsTime, out num))
                    throw new Exception("TimeStamp Is not valid");
                if (this.convertingTsu)
                {
                    CmsSigned cmsSigned = new
CmsSigned(signer1.getUnsignedAttribute("1.2.840.113549.1.9.16.2.14"));
                    SignerInfo signer2 = cmsSigned.getSigner(0);
                    cmsSigned.removeCertificates();
                    cmsSigned.addCertificate(collectReferencePolicy.certs[0].getEncoded());
                    byte[] encodedCertsC =
collectReferencePolicy.getEncodedCertsC(signer2.getDigestOid());
                    signer2.setUnsignedAttribute("1.2.840.113549.1.9.16.2.21", encodedCertsC,
encodedCertsC.Length);
                    byte[] encodedCrlsC =
collectReferencePolicy.getEncodedCrlsC(signer2.getDigestOid());
                    signer2.setUnsignedAttribute("1.2.840.113549.1.9.16.2.22", encodedCrlsC,
encodedCrlsC.Length);
                    if (withValues)
                    {
                        byte[] encodedCertsX = collectReferencePolicy.getEncodedCertsX();
                        signer2.setUnsignedAttribute("1.2.840.113549.1.9.16.2.23", encodedCertsX,
encodedCertsX.Length);
                        byte[] encodedCrlsX = collectReferencePolicy.getEncodedCrlsX();
                        signer2.setUnsignedAttribute("1.2.840.113549.1.9.16.2.24", encodedCrlsX,
encodedCrlsX.Length);
                    }
                    byte[] encoded = cmsSigned.getEncoded();
                    signer1.setUnsignedAttribute("1.2.840.113549.1.9.16.2.14", encoded,
encoded.Length);
                }
                CertstoreCms certstoreCms = new CertstoreCms((CmsSigned)this);
                //if (!certificatePathValidator.validate(signer1.getCertificate(), "any", tsTime,
(CertificateValidationPolicy)collectReferencePolicy, (CertificateFinder)certstoreCms, out num))
                //    throw new DsDataCorruptedException();
                byte[] encodedCertsC1 =
collectReferencePolicy.getEncodedCertsC(signer1.getDigestOid());
                signer1.setUnsignedAttribute("1.2.840.113549.1.9.16.2.21", encodedCertsC1,
encodedCertsC1.Length);
                byte[] encodedCrlsC1 =
collectReferencePolicy.getEncodedCrlsC(signer1.getDigestOid());
                signer1.setUnsignedAttribute("1.2.840.113549.1.9.16.2.22", encodedCrlsC1,
encodedCrlsC1.Length);
                if (withValues)
                {
                    byte[] encodedCertsX = collectReferencePolicy.getEncodedCertsX();
                    signer1.setUnsignedAttribute("1.2.840.113549.1.9.16.2.23", encodedCertsX,
encodedCertsX.Length);
                    byte[] encodedCrlsX = collectReferencePolicy.getEncodedCrlsX();
                    signer1.setUnsignedAttribute("1.2.840.113549.1.9.16.2.24", encodedCrlsX,
encodedCrlsX.Length);
                }
            }
            this.removeCertificates();
            for (int index = 0; index < this.getSignerCount(); ++index)
                this.addCertificate(this.getSigner(index).getCertificate().getEncoded());
        }

        private bool validateCertCAdEStp(
          Certificate cert,
          CertificateValidatorCache certCache,
```

```
          SignerInfo si,
          string intendedEku,
          DateTime tm,
          CertificateValidationPolicy policy,
          out int status)
        {
            byte[] unsignedAttribute1 = si.getUnsignedAttribute("1.2.840.113549.1.9.16.2.21");
            byte[] unsignedAttribute2 = si.getUnsignedAttribute("1.2.840.113549.1.9.16.2.22");
            byte[] certVals = (byte[])null;
            byte[] revocationVals = (byte[])null;
            if (si.isUnsignedAttributePresent("1.2.840.113549.1.9.16.2.23"))
            {
                byte[] unsignedAttribute3 =
si.getUnsignedAttribute("1.2.840.113549.1.9.16.2.23");
                byte[] unsignedAttribute4 =
si.getUnsignedAttribute("1.2.840.113549.1.9.16.2.24");
                certVals = unsignedAttribute3;
                revocationVals = unsignedAttribute4;
            }
            return new CertificatePathValidator(this.advAlgs, certCache).validateLong(cert,
intendedEku, tm, policy, out status, unsignedAttribute1, unsignedAttribute2, certVals,
revocationVals);
             }
```

ДОДАТОК Г

Пояснювальна записка на тему «Кросплатформенне програмне забезпечення для роботи із захищеними носіями ЕЦП на основі PKCS11» полягає в наступному: дослідити процес розробки кросплатформенних застосунків за допомогою сучасних технологій; проаналізувати особливості розробки на основі WebAssembly; проаналізувати архітектурні рішення даного способу розробки; реалізувати додаток на основі технологій Electron.js та .NET Core.