НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ *Сергій СТІРЕНКО*
(підпис)
“___”_____2021 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп’ютерні системи та мережі»

спеціальності 123 «Комп’ютерна інженерія»

на тему: «Система SSO автентифікації на базі цифрових сертифікатів»

Виконав:
студент _IV_курсу, групи__ІВ-72__
Голіней Андрій Михайлович _____
(підпис)

Керівник:
Доцент, кандидат технічних наук
Роковий Олександр Петрович _____
(підпис)

Консультант з нормоконтролю:
Професор, доктор технічних наук
Сімоненко Валерій Павлович _____
(підпис)

Рецензент

_____
(підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2021 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма
«Комп'ютерні системи та мережі»
спеціальність 123 «Комп'ютерна інженерія»

Затверджую:
зав. кафедрою
_____

«____»_____2021 р.

ЗАВДАННЯ

на бакалаврський дипломний проєкт студента

Голінея Андрія Михайловича

1. Тема проєкту *Система SSO автентифікації на базі цифрових сертифікатів*, керівник проєкту Роковий Олександр Петрович, доцент, кандидат технічних наук,

затверджена наказом по університету від « *11* » травня 2021 р. №1139-с

2. Термін здачі студентом закінченого проєкту _____*.2021р.*
3. Вихідні дані до проєкту *технічне завдання, теоретичні данні.*
4. Зміст розрахунково-пояснювальної записки: *опис предметної області, дослідження веб-сервісів пошуку оренди житла, веб-сервіс пошуку оренди житла з покращеною системою ідентифікації особистості.*

5. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
нормоконтроль	д.т.н., проф. Сімоненко В. П.		

6. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітки
1.	Затвердження теми роботи	10.12.2020 - 15.12.2020	
2.	Вивчення та аналіз завдання	15.12.2020 - 15.03.2021	
3.	Написання вступної частини та огляду предметної області	15.03.2021 - 25.03.2021	
4.	Розробка архітектури системи	25.03.2021 - 05.04.2021	
5.	Програмна реалізація системи	05.04.2021 - 15.04.2021	
6.	Оформлення документації дипломної роботи	15.04.2021 - 20.05.2021	
7.	Передзахист	23.05.2021	
8.	Захист	18.06.2021	

Студент _____ Андрій ГОЛІНЕЙ
(підпис)

Керівник проекту _____ Олександр РОКОВИЙ
(підпис)

АНОТАЦІЯ

В даному бакалаврському дипломному проєкті реалізована система єдиного входу з можливістю автентифікації за допомогою цифрових сертифікатів. Сервіс надає безпарольний доступ до підключених та попередньо налаштованих для SSO (Single Sign-On) застосунків. Є можливість додавати кілька користувачів і створювати корпораційну мережу. Сервіс формує звіт із здійснених входів, а також візуалізує стан прикріплених користувачами цифрових сертифікатів.

Програмний продукт був створений на мові JavaScript з використанням платформи Node.js та її бібліотеки NW.js, розробка велася у середовищі програмування WebStorm 2021. Також були застосовані доповнення до Node.js, такі як: завантажувач пакетів npm, модуль роботи із БД pg, фреймворк Express. Для контролю версії проекту було застосовано систему Git.

Дана реалізація веб-сервісу була побудована на основі вже існуючих аналогів. Враховані усі їх недоліки, зокрема була розроблена краща система безпеки передачі даних, а також додані інноваційні рішення запобіганню компрометації кожного користувача.

Ключові слова: Single Sign-On, безпека, цифровий сертифікат.

Розмір пояснювальної записки – 60 аркушів, містить 20 ілюстрацій, 4 додатки.

ANNOTATION

This bachelor's degree project implements a single-entry system with the possibility of authentication using digital certificates. The service provides passwordless access to connected and preconfigured for SSO (Single Sign-On) applications. It is possible to add multiple users and create a corporate network. The service generates a report on the logins made, as well as visualizes the status of digital certificates attached by users.

The software product was created in JavaScript using the Node.js platform and its NW.js library, development was carried out in the WebStorm 2021 programming environment. Additions to Node.js were also applied, such as: npm package loader, pg database module, Express framework. The Git system was used to control the version of the project.

This implementation of the web service was built on the basis of existing analogues. All their shortcomings have been taken into account, in particular, a better data security system has been developed, as well as innovative solutions have been added to prevent compromising each user.

Keywords: Single Sign-On, security, digital certificate.

Explanatory note size - 60 pages, contains 30 illustrations, 4 applications.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

до дипломної роботи
освітньо-кваліфікаційного рівня__бакалавр___

на тему: “ Система SSO автентифікації на базі цифрових сертифікатів ”

Київ – 2021 року

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	3	
2	А4	ІАЛЦ.467100.001 ВП	Відомість дипломного проекту	1	
3	А4	ІАЛЦ.467100.002 ТЗ	Технічне завдання	3	
4	А4	ІАЛЦ.467100.003 ПЗ	Пояснювальна записка	60	
5	А3	ІАЛЦ.467100.004 Д1	Схема бази даних	1	
6	А3	ІАЛЦ.467100.005 Д2	Принципова схема безпарольного входу	1	
7	А3	ІАЛЦ.467100.006 Д3	Функціональна схема авторизації	1	
8	А4	ІАЛЦ.467100.007 Д4	Текст програми	17	

					ІАЛЦ.467100.001 ВП		
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розроб.		*Голіней А.М.*			*Відомість дипломного проекту*	*Літ.*	*Арк.*	*Аркушів*
Перевір.		*Роковий О.П.*					*1*	*1*
						НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ ІВ-72		
Н. Контр.		*Сімоненко В.П.*						
Затверд.								

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломної роботи
освітньо-кваліфікаційного рівня___бакалавр___

на тему: “ Система SSO автентифікації на базі цифрових сертифікатів ”

Київ – 2021 року

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку програмного продукту по темі «Система SSO автентифікації на базі цифрових сертифікатів». Область застосування: захищений безпарольний доступ до застосунків.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка системи єдиного входу із використанням цифрових сертифікатів.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з теорії і практики програмування, публікації в Інтернеті з даних питань.

5. ТЕХНІЧНІ ВИМОГИ

5.1. ВИМОГИ ДО РОЗРОБЛЯЄМОГО ПРОДУКТУ

- Зручний програмний інтерфейс.
- Можливість безпарольно аутентифікуватися до попередньо підключених застосунків.
- Безпечність використання сервісу, що включає в себе можливість

авторизації за допомогою цифрового сертифікату стандарту X.509.

- Технічна коректність виконання усіх заданих алгоритмів.

5.2. ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

- Доступ до мережі Інтернет.
- Операційна система MS Windows 8, MS Windows 10.

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення систем аналогів, аналіз їх переваг та недоліків	20.12.2020
Складання і узгодження технічного завдання	03.02.2021
Створення модулів сервісу, що розробляється	16.03.2021
Тестування окремих модулів системи	11.05.2021
Допрацювання, налагодження і виправлення помилок	01.06.2021
Оформлення документації дипломної роботи	09.06.2021

Пояснювальна записка
до дипломного проєкту

на тему: «Система SSO автентифікації на базі цифрових сертифікатів»

Київ – 2021 р.

ЗМІСТ

ВСТУП

В теперішньому світі більшість програм розгортаються в центрах обробки даних та хмарах. Кожна бізнес-програма вимагає автентифікації користувачів, перш ніж вони отримають доступ до ресурсу. Для полегшення проблеми постійних запитів на доступ запровадили технологію єдиного входу (Single Sign-On, SSO). У дні до SSO кожен раз, коли користувачеві потрібно було переходити між програмами, йому доводилося входити з набором облікових даних. Здебільшого кожна програма мала окремий набір облікових даних, і це призводило до поганого користувацького досвіду, невдалих входів в результаті забутих облікових даних, непослідовних політик контролю доступу та вищих витрат на підтримку цих програм.

SSO спростив спосіб взаємодії та доступу користувачів до своїх програм. За допомогою SSO користувачі можуть заощадити час, отримуючи доступ до всіх своїх настільних, корпоративних та веб-додатків, а також інших корпоративних ресурсів, таких як спільні файли мережевих файлів, лише з одним набором облікових даних. Основним фактором є безпечність такої операції.

Існують різні практики безпечної передачі особистої інформації, проте більшість є або дорого вартісними, або ресурснозатратними, або не зовсім зручними.

Завданням цієї дипломної роботи є вирішення проблеми безпечного єдиного входу шляхом взаємної аутентифікації способом обміну цифровими сертифікатами. Також потрібно спроектувати систему, яка надавала б зручний функціонал для роботи із сертифікатами.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1. Визначення систем SSO автентифікації

Системи єдиного входу (SSO, Single Sign-On) - це комплекс програмних засобів або вбудована технологія, що дозволяють користувачу переходити по різних розділах і сервісах одного порталу, такого як форум, блог та інші, без повторної аутентифікації. Інакше кажучи, користувач проходить процедуру аутентифікації в одному місці, після чого отримує доступ до всіх пов'язаних розділів, і йому не доводиться вводити свої облікові дані в кількох місцях при роботі з різними програмами [5].

Системи SSO відрізняються від стандартних методів аутентифікації при спробах користувача отримати доступ до сервісу: аутентифікаційні дані запитуються не від користувача, а з SSO-додатку. Сервіси SSO не зберігають облікових даних користувача і не використовують їх для кожної програми. Для цього вони використовують токен, наданий під час першої автентифікації. Це можна вважати концепцією одноразового паролю. Існують кілька способів застосування технології Single Sign-On, а саме:

- Web SSO;
- Серверний;
- Клієнтський;
- Комбінований.

При цьому при використанні технології Single Sign-On можуть застосовуватися такі способи реалізації:[5]

- Методи автоматичної підстановки паролів.
- Технологія PKI і застосування цифрових сертифікатів.
- Аутентифікація на основі Kerberos.
- Маркери доступу.
- Смарт-карти або USB-токени.

У великих мережах для єдиного входу використовуються різні протоколи аутенифікації:

1. **Kerberos** – працює між різними платформами, шифрує дані і забезпечує захист від атак повторним відтворенням.
2. **OpenID** – дає змогу використовувати обліковий запис на інтернет-ресурсах, які не пов'язані між собою.
3. **WS-Security** – виконує аутентифікацію, використовуючи на стороні клієнта тільки браузер.
4. **SAML** – дозволяє виконувати аутентифікацію на рівні мережі, використовуючи тільки браузер користувача.
5. **Oauth** – дозволяє надавати ресурси користувача, приховуючи дані ідентифікації; доступ здійснюється від імені облікового запису власника даних.

Зазначені протоколи мають різні сфери використання. Наприклад, Kerberos використовує пряме мережеве з’єднання, тому цей протокол добре працює у приватних і внутрішніх корпоративних мережах. Для хмарних або інтернет-програм краще підійде сучасна автентифікація. До сучасної SSO автентифікації відносяться веб-протоколи автентифікації, які використовуються хмарними програмами(OAuth, Open ID Connect і SAML2). Вони працюють через інтернет і шифрують зв’язок за допомогою HTTPS.[3]

В даній роботі розглянуться програмні рішення для SSO автентифікації, які будуть використовувати технологію PKI і застосування цифрових сертифікатів для реалізації ряду функціональних рішень для вирішення проблемних питань.

1.2. Аналіз існуючих систем SSO автентифікації

Розглянемо перелік існуючих SSO сервісів, в яких враховано багато факторів для легкої інтеграції без додаткового навантаження на IT-службу та

комфортного використання у середовищі компанії або індивідуально. Проаналізуємо основні критерії безпеки та ідентифікації особистостей для кожного з них у наведеній таблиці:

Таблиця 1.1. – Критерії безпеки SSO систем.

Критерій	Коментар
Підтримка технології строгої автентифікації	Якщо при одноразовій аутентифікації використовується пароль, то дізнавшись даний пароль, можна отримати доступ до всіх програм, що входять в SSO-профіль користувача. Дуже важливо використовувати надійні технології одноразової аутентифікації.
Наповнення SSO-профілю співробітника обліковими даними	1. Автоматична генерація при першому вході в додаток 2. Самонавчання (користувач вводить дані при першому вході) 3. Централізоване наповнення (адміністратор вказує логін і пароль) 4. Імпорт з файлу
Централізоване адміністрування системи	Можливість своєчасно відкликати доступ співробітника.
Використання рольової моделі надання доступу	Дозволяє встановити відповідність між посадою співробітника і набором доступних йому додатків.
Підтримка нового цільового додатку силами клієнта	Можливість самостійно інтегрувати новий додаток в SSO-систему.
Інтеграція з популярними Identity Management (IDM) системами	Дозволяє домогтися повної автоматизації в наданні доступу користувачам. IDM система автоматично створює в системі всі необхідні облікові записи нового користувача. Він отримує прозорий доступ у всі необхідні додатки.
Підтримка PKI інфраструктури	Можливість шифрування паролів цифровим сертифікатом користувача.

Для комфортності аналізу слід розділити SSO на два основних види технологій єдиного входу:

1) **Корпоративний (Enterprise) Single Sign-on**, що являє собою установку програми на робочі станції користувачів. Цей вид забезпечує автоматичну підстановку в аутентифікаційні вікна додатків логіна і пароля користувача.
2) **Web Single Sign-on**, що забезпечує аутентифікацію в web-додатках. Ця технологія надає доступ до всіх підключених додатків організації з підтримкою протоколів OpenID OAuth і SAML. При цьому використовується один обліковий запис.

Відповідно до інформації, наданої світовим аналітичним агентством MarketsandMarkets, загальний ринок Single Sign-On систем у 2021 році складає $1599,8 млн у порівнянні з $ 846,6 млн станом на 2016 рік. У звіті MarketsandMarkets розглянуті Web, Enterprise та Federated SSO-рішення постачальників - IBM Corporation, Oracle Corporation, Ping Identity Corporation, Dell Software, OKTA Inc., OneLogin Inc. та ін.[4]

Лідерами ринку за даними звіту Gartner (https://www.gartner.com) визнають:

- Okta
- Microsoft
- OneLogin
- Ping Identity
- Auth0

Згідно цих звітів, основними критеріями зростання ESSO вважають зручність використання, продуктивність роботи та оптимізацію при управлінні обліковими записами і паролями.

1.2.1. Українські сервіси єдиного входу

Український ринок систем єдиного входу є не обширним: переважають провайдери єдиного доступу – компанії, які надають послуги єдиного входу, підтримувані закордонними сервісами. Втім, серед українських можна виділити такі сервіси як:

- «ID.GOV.UA» (https://id.gov.ua/)
- «SSOid.net» (https://ssoid.net/)

ID.GOV.UA є продуктом державного підприємства «ДІЯ» за підтримки Міністерства цифрової трансформації України. Це універсальна платформа для електронної ідентифікації та аутентифікації користувачів. Фізичні та юридичні особи можуть зручно і безпечно, проходити електронну ідентифікацію за допомогою електронних підписів та аутентифікуватися на всіх державних сайтах України.[2]

Платформа ICEI надає зручну та безпечну електронну ідентифікацію за допомогою електронних підписів, MobileID та BankID НБУ. Завдяки цій системі отримувати електронні послуги ще простіше і швидше, використовуючи кваліфікований електронний підпис.

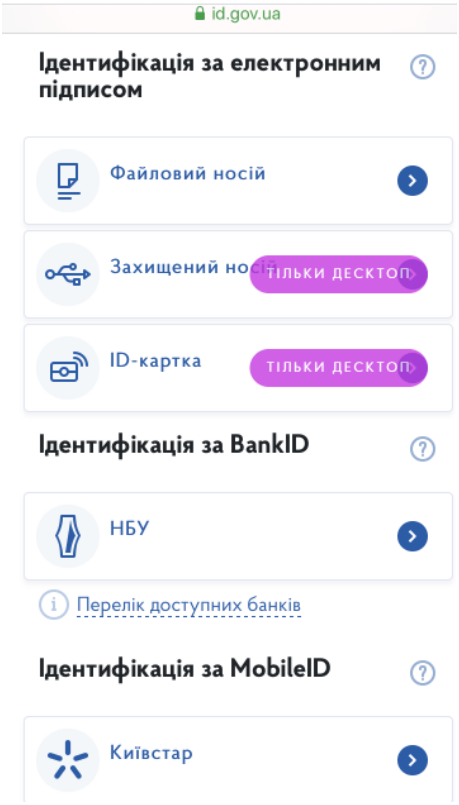


Рисунок 1.1 Інтерфейс та підтримувані технології системи ID.GOV.UA

З системою співпрацюють:

- 23 надавачі електронних довірчих послуг
- 3 надавачі послуг Mobile ID
- 27 банків-ідентифікаторів системи BankID НБУ
- 58 схем електронної ідентифікації
- 203 системи для автентифікації
- 110 систем для створення та перевірки кваліфікованого електронного підпису.

Рисунок 1.2 Вхід в електронний кабінет водія з використанням сервісу ID.GOV.UA

ICEI дозволяє отримувати публічні послуги онлайн, користуватися електронним документообігом. Тут наявний достатній функціонал, щоб спростити ведення бізнесу. Платформа має атестат відповідності комплексної системи захисту інформації, тому персональні дані користувачів надійно захищені. Система передає автентифікаційні дані безпосередньо надавачу послуг, через сервіс якого здійснюється ідентифікація, верифікація чи реєстрація.

В загальному, продукт ID.GOV.UA активно використовується з 2019 року, але він не є завершеним: функціонал системи доповнюють, розширюють стек підтримуваних технологій. Зараз це невід'ємна частина конкретно державних інформаційних адміністративних послуг, але вона не може задовільнити всіх потреб користувачів корпоративних чи Web-SSO систем.

Наступний сервіс – «SSOid.net» є повноцінно спрямованим на компанії або окремих користувачів. Цей продукт теж з'явився на ринку недавно, про що свідчать незакінчені розділи на головній сторінці.

Рисунок 1.3. Головна сторінка сервісу «SSOid.net»

«SSOid.net» забезпечує зручний доступ до всіх корпоративних ресурсів із єдиним набором облікових даних.

Можливості:

- Керування користувачами.
- Аутентифікація із зовнішніми IdPs.
- Інтеграція каталогів.

Тут можна легко інтегрувати свою існуючу LDAP / Active Directory, забезпечивши цим, використовуючи існуючі облікові дані, вхід у систему і безпечний доступ до додатків. Сервіс інтегрується з SSO із використанням SSO-конекторів на різних платформах, таких як Node.js, .Net, PHP, Java, Ruby. Цей продукт представляє багато просунутих рішень, що є корисно для додатків, які не мають вбудованої підтримки будь-якого SSO-протоколу. Тут підтримуються декілька IDP, що дозволяє автоматично керувати обліковими записами користувачів у додатках, а це в свою чергу економить час і забезпечує актуальність прав доступу користувача.

«SSOid.net» – це сучасна українська система єдиного входу. Вона дозволяє використовувати єдиний логін та пароль на всіх сервісах, які підтримують авторизацію за допомогою SSOid.net.

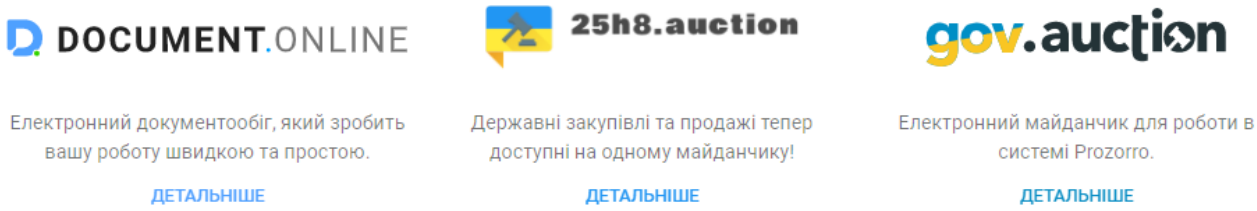


Рисунок 1.4. Сервіси-партнери «SSOid.net»

Користувач з метою отримання послуг Одержувача (сервісу-партнера), може скористатися сервісом SSOid.net та пройти ідентифікацію за його допомогою, в такому порядку:

1)Перейти на Сайт Одержувача;

2)На Сайті Одержувача обрати сторінку реєстрації або авторизації та перейти на Сторінку сервісу SSOid.net;

3)Пройти ідентифікацію на Сторінці сервісу SSOid.net;

4)Авторизуватися (зареєструватись) в Сервісі Одержувача.

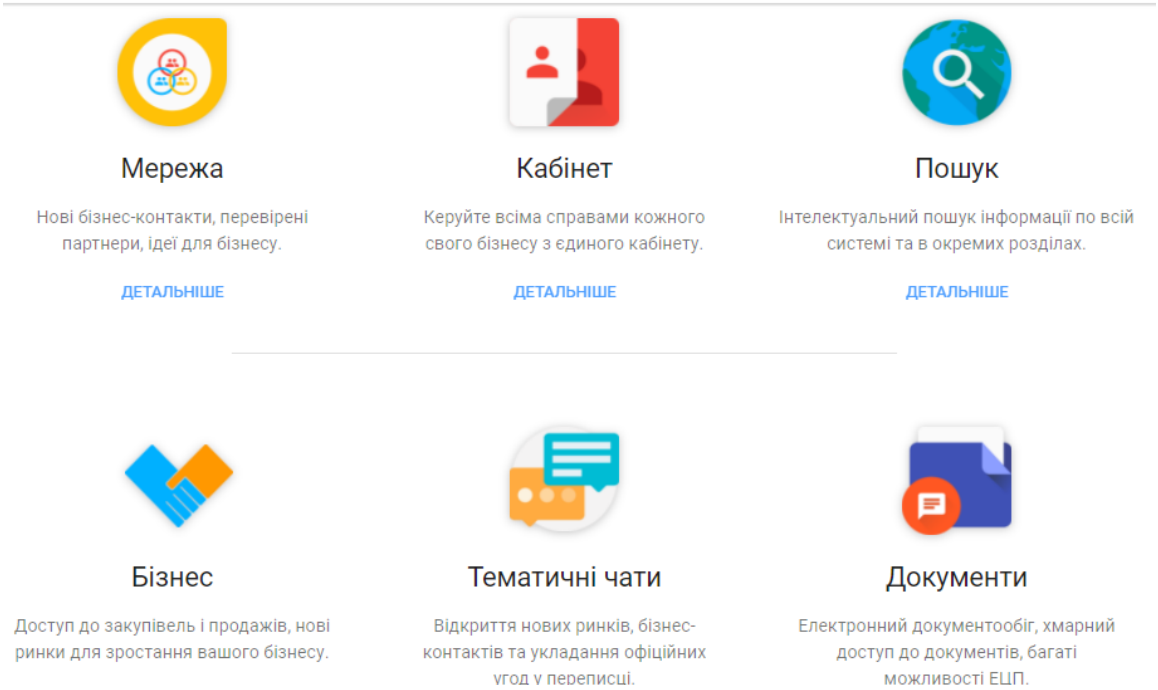


Рисунок 1.5. Додатковий функціонал сервісу «SSOid.net»

Користувачам веб-сервісу відкрита велика кількість додаткових функцій для ведення і розвитку бізнесу, пошуку ідей та партнерів. В сервісі представлена ціла мережа для комфортної роботи в системі. Можна також вести електронний документообіг, здійснювати хмарний доступ до документів, підписувати їх з допомогою ЕЦП та підтверджувати цифровими сертифікатами.

1.2.2. Іноземні сервіси єдиного входу

Серед іноземних рішень на ринку представлені наступні сервіси:

- «One Identity Enterprise Single Sign-on» (https://www.oneidentity.com/)
- «Oracle Enterprise Single Sign-On Suite» (https://www.oracle.com/)
- «Indeed Enterprise Single Sign-on» (https://indeed-id.ru/)

Продукт компанії One Identity забезпечує єдину точку входу в корпоративні додатки, що полегшує процес управління доступом кінцевих користувачів. Система призначена для двофакторної аутентифікації у мережі з використанням USB-токенів. Замість пароля користувач повинен пред'явити електронний USB-ключ і ввести PIN-код. Таким способом можна здійснити автентифікацію у наступні системи:[6]

- Active Directory (сервер Windows 2000/2003);
- SAMBA (сервер Linux);
- Локальні робочі станції Windows NT/2000/XP;
- NDS/eDirectory (сервер Novell NetWare 4.x-6.x);

Роботу системи автентифікації починаємо зі створення профілів користувачів і запису їх на електронний за допомогою модуля адміністрування. Ця операція здійснюється у вікні самостійної реєстрації профілів (Рисунок 1.6.).

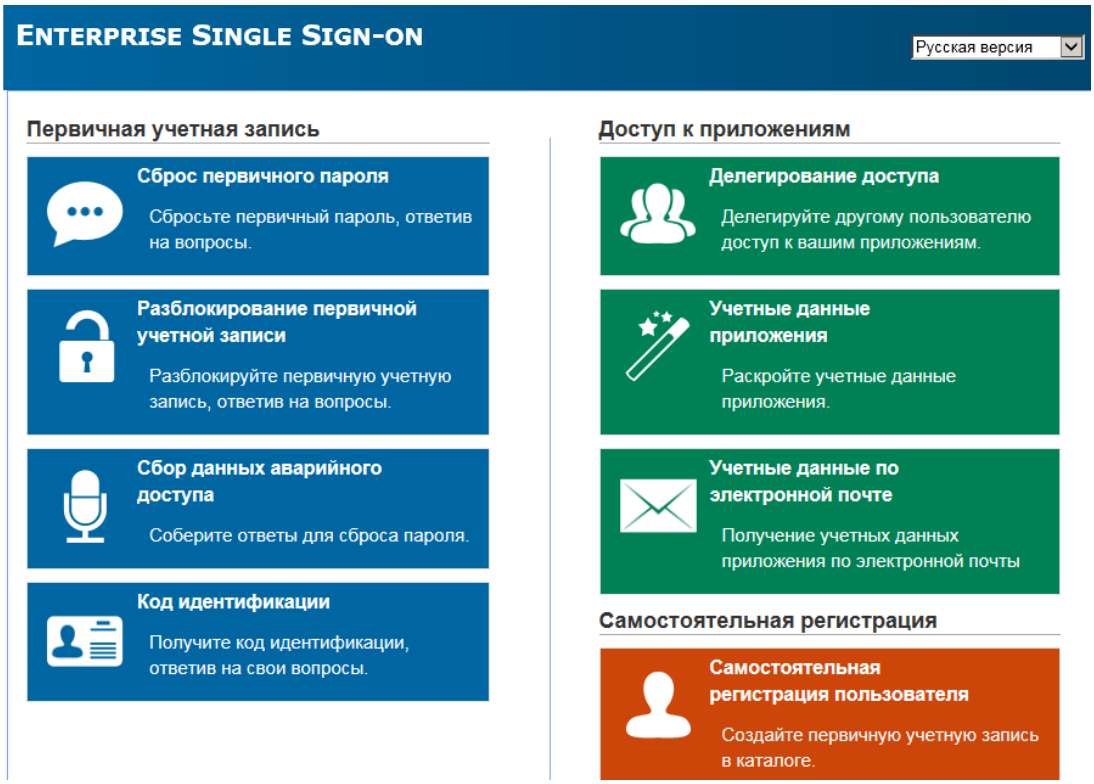


Рисунок 1.6. Web-портал «One Identity Enterprise Single Sign-on»

На першому етапі реєстрації профілів адміністратор мережі вказує облікові записи, які необхідно включити в профіль. Профіль складається з одного облікового запису для аутентифікації в Windows та облікового запису NDS.

На другому етапі відбувається генерація нового пароля для обраного користувача і запис в ключ інформації, необхідної для аутентифікації користувача при вході в мережу, а також додаткової службової та конфігураційної інформації. Після цієї операції користувач не зможе увійти в мережу без ключа, оскільки пароль його облікового запису буде змінено.

Після вставлення ключа і введення правильного PIN-коду система зчитує з електронного ключа профіль користувача і здійснює аутентифікацію. Під час вилучення електронного ключа, в залежності від налаштувань, консоль комп'ютера блокується або здійснюється вихід з мережі (logout). «One Identity Enterprise Single Sign-on» може застосовувати для входу в мережу також метод Universal SmartCard Login і цифрові сертифікати X.509.

Перевагою цієї системи є генерація паролів спеціальним алгоритмом, їх систематична регенерація, а також інтеграція з Novell Certificate Server і інфраструктурою відкритих ключів PKI.

Наступним розглянемо комплексне рішення компанії Oracle, яке забезпечує одноразову аутентифікацію і прозоре підключення користувачів без необхідності внесення змін в існуючі програми. З допомогою модуля ESSO Logon Manager співробітники входять у Windows, а коли вони запускають свої інші програми, ESSO Logon Manager автоматично входить до них із правильним паролем для кожної програми. Система може бути встановлена на корпоративних комп'ютерах або ж доступна на вимогу з будь-якого приватного комп'ютера - де завгодно та будь-коли.[6]

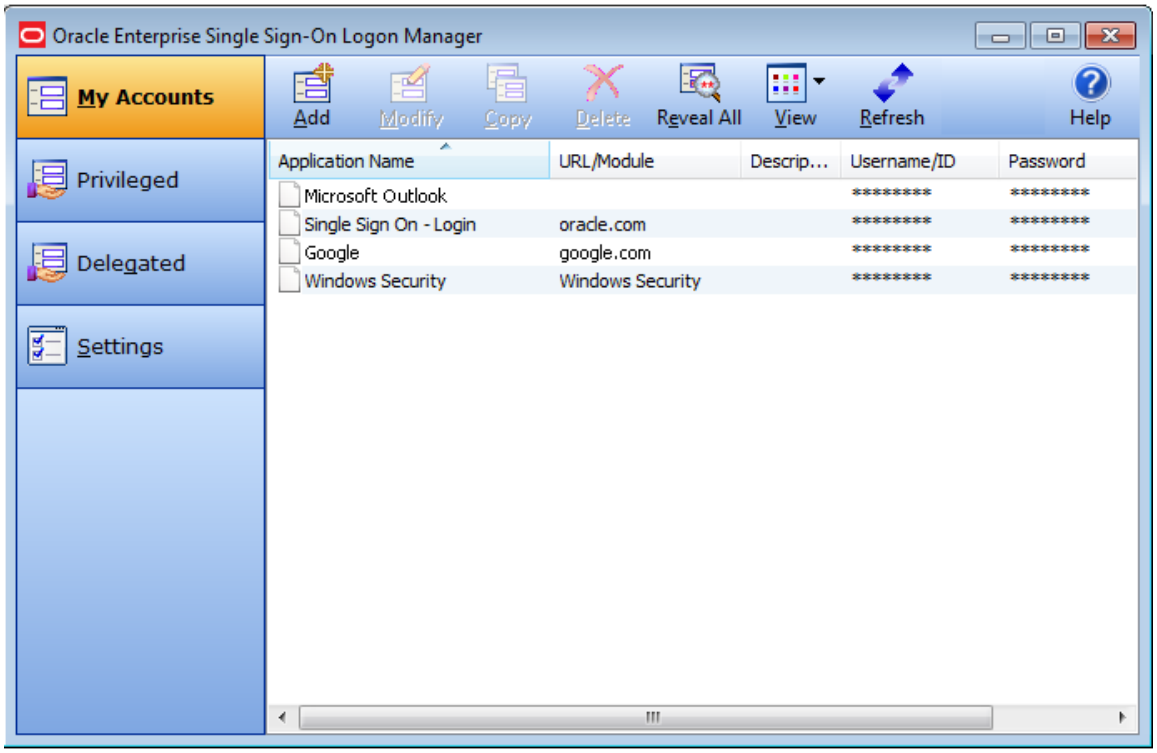


Рисунок 1.7. Інтерфейс «Oracle Enterprise Single Sign-On Logon Manager»

При роботі в корпорації інший модуль – ESSO Provisioning Gateway дозволяє системним адміністраторам автоматично додавати, редагувати або видаляти облікові дані кінцевого користувача. Це значно спрощує роботу:

усувається необхідність користувачам вводити свої паролі, а при необхідності облікові дані можна безпечно видалити.

За автентифікацію відповідає модуль ESSO Universal Authentication Manager. Він дозволяє користувачам входити в Windows за допомогою будь-якого пристрою автентифікації, наприклад RFID мітки. Universal Authentication Manager підтримує стандартний інтерфейс входу в систему Windows, тому цей процес стає інтуїтивно зрозумілим, швидким та безпечним. Це перше рішення надійної автентифікації корпоративного класу без серверної інфраструктури, що робить його недорогим і надійним.

За відновлення пароля відповідає модуль ESSO Password Reset. Він надає користувачам швидкий спосіб відновити доступ до свого комп'ютера за допомогою звичного скидання пароля у Windows.

Загалом, рішення компанії Oracle є дуже вдалим: підтримує велику кількість додатків та, завдяки зрозумілому інтерфейсу, легко адаптує до себе користувача. Єдиний вхід гарантується незалежно від типу додатків, а багатофакторна аутентифікація застосовується для доступу до додатків, які її раніше не підтримували. Ще однією перевагою є зберігання даних у централізованих сховищах Oracle Directory Services, Microsoft Active Directory, а також у БД Oracle та БД SQL.

Недоліком цієї системи є відсутність підтримки для входу цифрових сертифікатів X.509.

Наступне рішення «Indeed Enterprise Single Sign-on» російської компанії Indeed ID призначене організувати доступ до всіх корпоративних IT-систем та спростити процес управління обліковими даними користувачів. Сервіс пропонує такі можливості:

- підтримка будь-яких типів додатків;
- сувора аутентифікація при доступі до додатків;
- наскрізна аутентифікація в додатках;

- підтримка широкого спектру технологій аутентифікації і можливість їх комбінувати;
- журналювання подій в системі й записування дій адміністраторів та працівників;

Управління користувачами і параметрами системи виконується централізовано за допомогою консолі Indeed Enterprise Management Console. Адміністратор отримує повний набір інструментів для роботи з системою.

Indeed Enterprise SSO складається з серверної і клієнтської частин, кожна містить окремі модулі. Робота системи заснована на взаємодії Агента (клієнстька частина) і сервера Indeed (серверна частина). В момент, коли співробітник запускає додаток зі свого профілю доступу, ESSO Агент перехоплює реєстраційне вікно цієї програми, приховує його від користувача, автоматично заповнює необхідними даними, отриманими з сервера (підставляє ім'я облікового запису та пароль). Факт виконання успішної або неуспішної спроби доступу фіксується в журналі подій системи. [6]

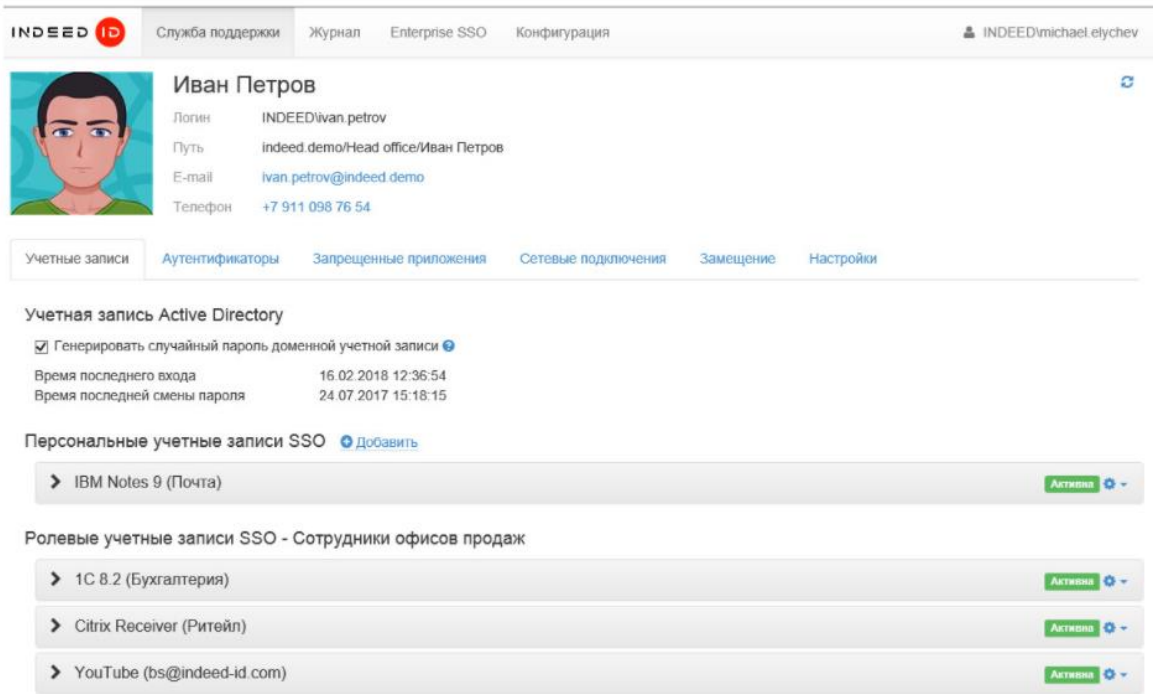


Рисунок 1.8. Картка користувача Indeed Enterprise Management Console

Робота з SSO-параметрами користувачів виконується в так званій картці користувача, яка є сторінкою, яка показує інформацію про співробітника, його аутентифікатор і облікові записи SSO.

Розглянувши даний сервіс, можна зазначити, що він реалізує задачі, яких немає у іноземних аналогів, а саме:

- сувора і наскрізна аутентифікація в додатках в термінальному режимі роботи;
- журнал роботи (запис успішних та неуспішних автентифікацій);
- запис дій адміністраторів і користувачів.

Із недоліків можна відмітити незручне налаштування серверної частини через командний рядок.

1.3. Концепція системи SSO автентифікації на базі цифрових сертифікатів

Технологія цифрових сертифікатів має широку сферу застосування не тільки в Інфраструктурі відкритих ключів (PKI). Вона дозволяє шифрувати та підписувати електронну пошту, засвідчує чинність відкритого ключа, підтримує механізм електронного підпису, а також забезпечує автентифікацію клієнта.[3]

При використанні асиметричних методів шифрування ідентифікаційних даних користувача (паролів і, зокрема, електронного цифрового підпису) необхідно мати гарантію автентичності пари (ім'я, відкритий ключ). Для вирішення цього завдання в специфікаціях X.509 вводяться поняття цифрового сертифікату та засвідчувального центру.[1]

Засвідчувальний центр - це компонент глобальної служби каталогів, що керує криптографічними ключами користувачів. Вся інформація про користувачів зберігається засвідчувальними центрами у вигляді **цифрових сертифікатів**, що мають таку структуру:

- ім'я центра, що засвідчує;

- порядковий номер сертифіката;
- ім'я власника сертифіката (кому належить сертифікат);
- ідентифікатор алгоритму електронного підпису;
- відкриті ключі власника сертифіката (ключів може бути декілька);
- термін придатності;
- ідентифікатори алгоритмів, асоційованих з відкритими ключами власника сертифіката;
- електронний підпис, згенерований з використанням секретного ключа (підписується результат хешування інформації, що зберігається в сертифікаті).

Сертифікати клієнтів часто використовуються при двофакторній аутентифікації. Розглянемо, як працюватиме таке рішення:

1. Користувач в системі SSO обирає додаток, до якого хоче доступитися .
2. У вікні авторизації обирає спосіб «Використати цифровий сертифікат».
3. Користувач завантажує файл сертифіката.
4. Сервер перевіряє достовірність сертифіката клієнта
5. Сервер надає користувачеві доступ до додатку.

Дана концепція робить систему єдиного входу ще простішою, адже користувачу не доведеться вставляти USB-токен чи вводити пін-код, щоб отримати доступ на будь-який сайт або програму в корпорації. Клієнтські сертифікати дозволяють захистити мережу, додатки і інтерфейси, дозволяючи доступ тільки уповноваженим особам. Такий спосіб задовольняє необхідність аутентифікації пристроїв, так як сертифікати клієнтів захищають пристрої і контролюють доступ, тому їх можуть використовувати тільки уповноважені особи.

ВИСНОВКИ ДО РОЗДІЛУ

Незважаючи на існування великої кількості корпоративних та Web-SSO сервісів, вони все ж мають недоліки в кількості підтримуваних способів автентифікації або в зручності їх використання. Варіанти застосування технології єдиного входу, які існують на українському ринку, потребують більше часу, щоб перейти із нововведень до сталого довершеного продукту. Існує великий розрив між іноземними застосунками. Лідери закордонних ринків системного забезпечення готові для постійного використання, проте є дорого вартісними для звичайного користувача чи невеликої фірми .

Для покращення практик усіх проаналізованих SSO-систем необхідно обрати таку, яка була б зручною у використанні, простою у встановленні і маловартісною для небагатого українського користувача. Такою є система на базі цифрових сертифікатів, концепція яких була розглянута.

РОЗДІЛ 2. ВИЗНАЧЕННЯ ВИМОГ ТА ЗАДАЧ ПРОЕКТУВАННЯ

2.1. Функціональні задачі системи SSO автентифікації

Принцип роботи SSO-сервісів полягає у керуванню процесом аутентифікації користувачів, що являє собою наступну послідовність:

1. Користувач входить в систему і вводить облікові дані один раз.
2. Намагається доступитися до додатків, де вимагається авторизація.
3. Система перехоплює запит авторизації і передає на SSO-сервіс.
4. Проводиться автентифікація обраним способом(у нашому випадку – сертифікатом X.509) та формуються спеціальні “тікети” успішної аутентифікації.
5. Довірений сервіс перевіряє істинність тікетів.
6. Надається доступу за допомогою тікетів до цільових додатків.

Щоб задовільнити всі пункти, оптимально застосувати технологію «SSO із коробки», яку реалізовує протокол автентифікації Kerberos. [7]

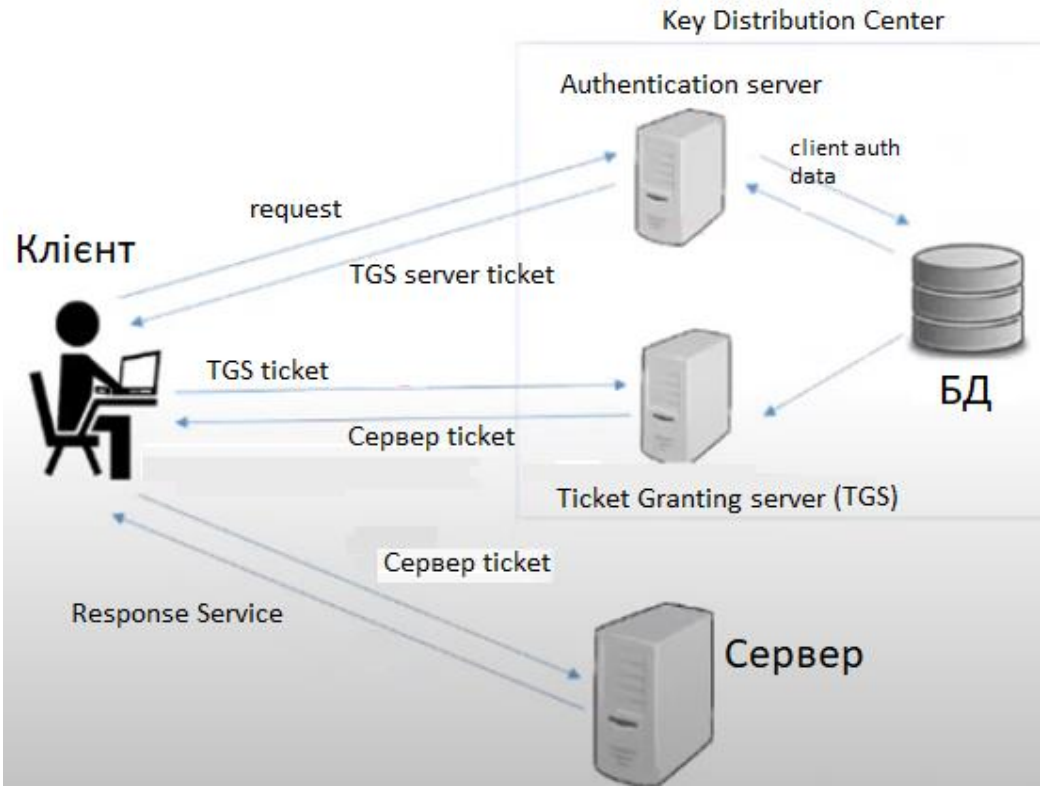


Рисунок 2.1. Структура протоколу Kerberos

Учасниками процесу є:

- Клієнт (довільний користувач);
- Сервер (де хоститься захищений протоколом сервіс);
- Key Distribution Center (KDC) – автентифікаційний third-party, якому довіряють клієнт і сервер. У свою чергу поділяється на:
    - Authentication Server (ініціалізує автентифікацію та видає TGS-тікети користувачеві);
    - Ticket-Granting Server (на основі TGS формує та видає service тікети);

Kerberos починає працювати з моменту входу в Windows, звертаючись до KDC. Опишемо алгоритм роботи протоколу[10]:

1. Центр видає TGT-тікет, який засвідчує аутентифікацію в системі.
2. Токен зберігається в кеші операційної системи і всі клієнти, які хочуть працювати із Kerberos, мають до нього доступ.
3. При роботі із довільними додатками на запит авторизаційних даних Kerberos-клієнт звертається до кешу операційної системи по TGT-тікет.
4. Із цим тікетом у KDC видається інший для роботи із конкретним додатком.
5. Виданий service ticket у авторизаційному хедері передається на сервер.
6. Сервер приймає, переглядає, верифікує і, якщо операція успішна, надає доступ в додаток без ніяких логінів і паролів.

Функція єдиного входу може коригуватися постачальником і середовищем. Наприклад, в одному SSO-додатку може комбінуватися Kerberos у корпоративній мережі і файли cookie браузера для автентифікації через інтернет. У сучасних протоколах ідентифікаційні дані клієнтів можуть бути подані довільним способом. Основним критерієм оцінки систем єдиного входу є безпека ідентифікаційних даних. Для забезпечення високої якості продукту, застосовані протоколи підкріплюються багатофакторною

аутентифікацією, сертифікатами на пристрій для ідентифікації відомих пристроїв.

2.2. Автентифікація з використанням цифрових сертифікатів

Функції SSO-сервісу, який розробляється, полягають у перевірці цифрових підписів сертифікатів X.509, згенерованих довіреним Центром сертифікації та затверджених (підписаних) Certification Authority. При успішному проходженні всіх процесів, якщо сертифікат правильний, користувач безпечно авторизується.[3]

Технологія цифрових сертифікатів винайдена для безпечного використання передуючій їй технології відкритих ключів (Public Key Infrastructure), тому для кращого розуміння слід спочатку розглянути принцип роботи PKI яку засновано на асиметричному шифруванні.[11]

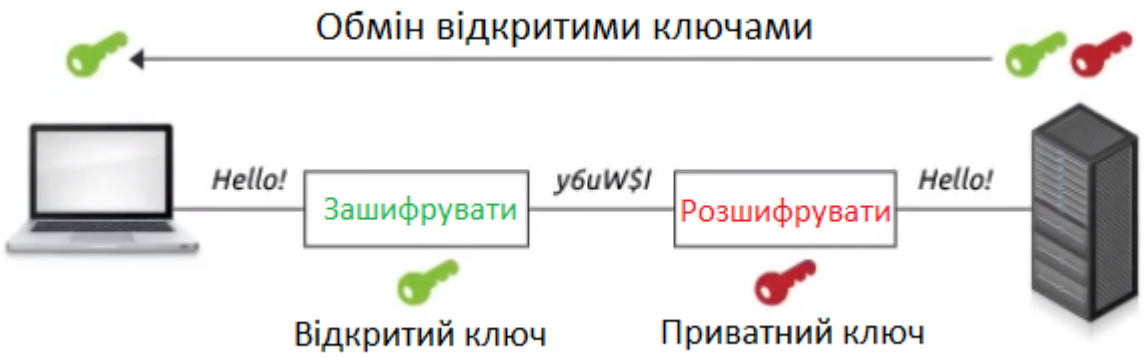


Рисунок 2.2. Принцип роботи Public Key Infrastructure

Алгоритм обміну:

1. Кінцевий користувач розсилає свій відкритий ключ всім пристроям, які хочуть налагодити комунікацію.
2. Вони в свою чергу використовують цей ключ, щоб зашифрувати надіслану інформацію.
3. Кінцевий користувач розшифровує отримані дані своїм приватним

ключем.

Асиметричний тип шифрування використовує важкі 2048-бітні окремі ключі, на відміну від симетричного, де інформація шифрується і розшифровується одним 256-бітним ключем. Процес шифрування важкими ключами довший, але безпечніший.

Незважаючи на це, в асиметричному шифруванні велика ймовірність, що відкритий ключ сервера перехопить третя сторона. Вона надсилає клієнту взамін свій відкритий ключ, який сприймається як ключ сервера, адже немає прив'язки до власників. Браузер клієнта шифрує симетричний ключ сесії з отриманого і надсилає перехоплювачу, помилково вважаючи, що це сервер. Оскільки ключ був зашифрований перехоплювачем, то він може розшифрувати його своїм приватним ключем. Таким чином, перехоплювач створює незахищене з'єднання і може розшифровувати, читати і навіть модифікувати дані, а сервер і користувач не знатимуть про це.

Цифрові сертифікати зберігають довіру між сторонами, які аутентифікуються. Вони прив'язують публічні ключі до їх власників, тим самим убезпечуючи передачу даних між користувачами. Розглянемо детальніше, яким способом можна отримати затверджені сертифікати. [1]

Користувач надає наступну інформацію:

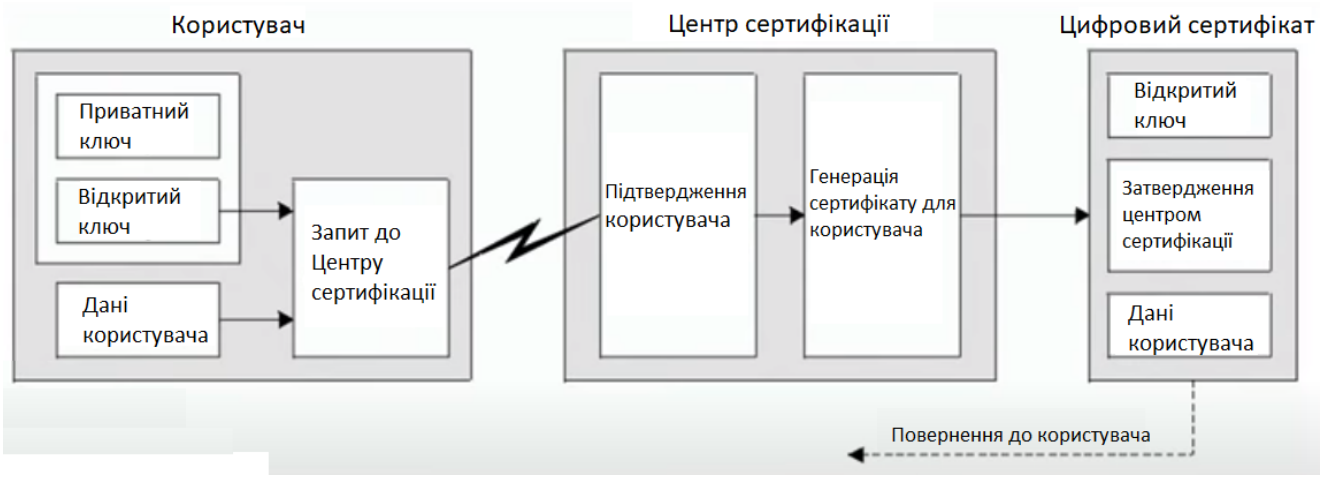


Рисунок 2.3. Спосіб отримання цифрових сертифікатів

- згенерований відкритий ключ (складова структури PKI, яка є

невід’ємною частиною захищених комунікацій у мережі);

- ім’я;
- адресу;
- назву компанії, тощо.

Формується запит у центр сертифікації, де він валідується, підтверджується і створюється. Користувач отримує файл сертифікату, на якому записані його дані, відкритий ключ, а також інформація про того, хто засвідчив цей сертифікат.

Розглянемо принцип генерації цифрових сертифікатів і їх стандарт X.509.[1]

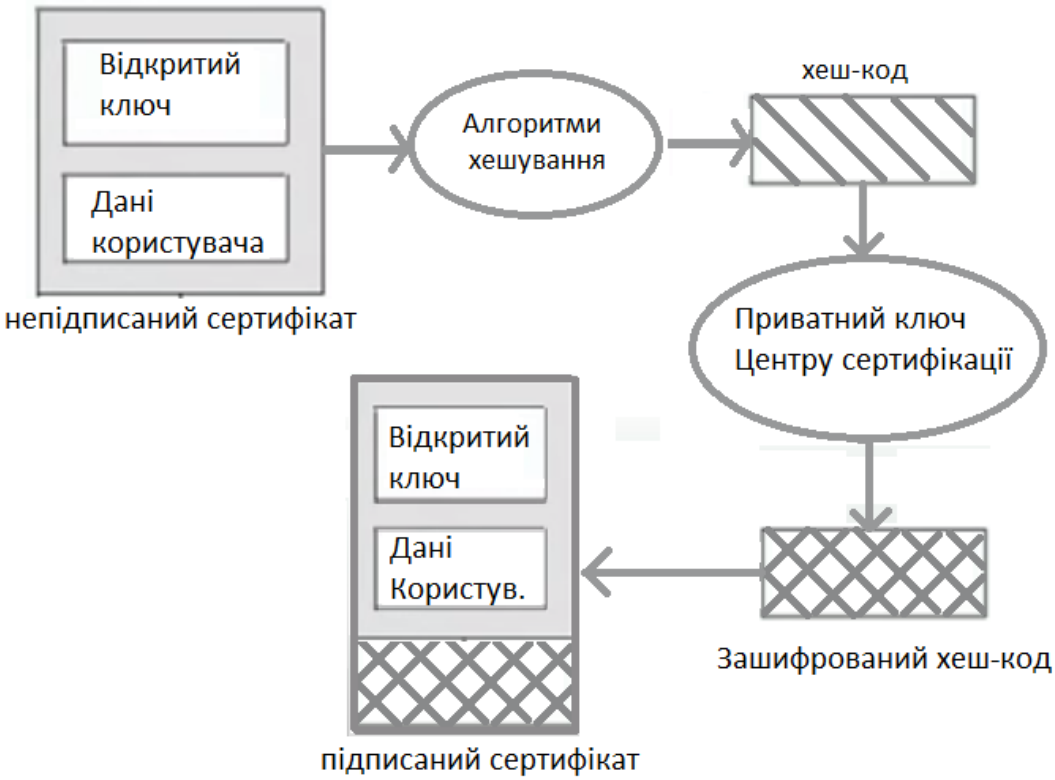


Рисунок 2.4. Схема генерації цифрового сертифікату

На вході подається структура із відкритого користувацького ключа й ідентифікаційних даних – це так званий «непідписаний сертифікат». До цієї структури застосовуються алгоритми хешування та генерується хеш-код. Цей хеш-код зашифровується приватним ключем Центру сертифікації.

Зашифрований хеш додається до початкової структури і отримуємо готовий «підписаний сертифікат», в якому «підписом» вважається зашифрований хеш-код. Розшифрувати підпис може тільки той користувач, який має відкритий ключ Центру сертифікації.

Центр сертифікації не відповідає за створення відкритих ключів для кожного клієнта, але відповідає за генерацію підпису для існуючих відкритих ключів. Процес аутентифікації полягає у розшифруванні хеш-кодів підписів та пошуку спільного між ними. Автентифікація успішна, якщо знайдені відповідності між розшифрованими хеш-кодами.

Виходячи з цього, має існувати місце, де зберігатимуться цифрові сертифікати, таким став єдиний каталог сертифікації X.509. [1]

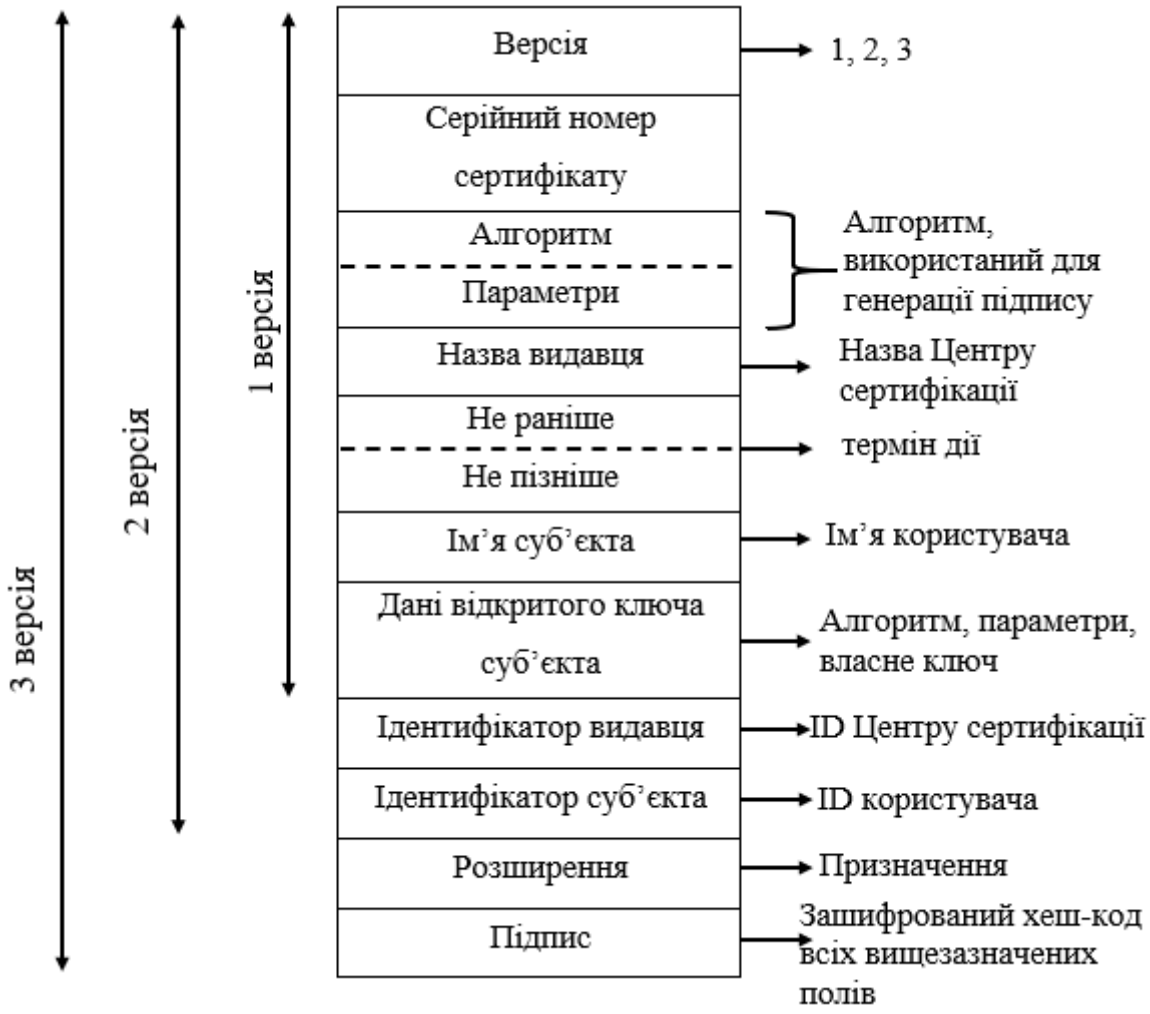


Рисунок 2.5. Структура сертифікату X.509

Стандартизовані сертифікати X.509 давно використовуються. Функції органів їх видачі полягають у тестуванні на захищеність, тобто безпекою передачі та зберігання персональних ключів користувачів. Всі сертифікати генеруються з обмеженим періодом, і після його закінчення вони більше не вважаються дійсними. Терміни дії можуть становити один або два роки, але сертифікати повинні бути замінені до цієї дати, щоб уникнути попереджень або порушень безпеки. Хорошим підходом вважають підтримку життєдіяльності (продовження, оновлення) сертифікатів, термін дії яких менший, адже чим довший термін дії сертифіката, тим більша імовірність його компрометації через втрату персонального ключа користувачем.[11]

Це породжує ще одну функціональну задачу розроблюваної системи – **управління життєвим циклом сертифіката**. Вона повинна підтримувати наступні етапи життєдіяльності:

- **Реєстрація SSL-сертифіката**

  Це спільний процес між ЦС і користувачем, який виконується тільки після запиту користувача у довірений ЦС. Сформований запит на реєстрацію (CSR) містить відкритий ключ й інформацію про реєстрацію. Як тільки користувач запитує сертифікат, ЦС перевіряє подану інформацію. Згідно встановлених правил політики ЦС створює і публікує сертифікат і відправляє користувачеві.

- **Перевірка сертифіката**

  Статус SSL сертифікату перевіряється кожен раз, коли використовується, щоб дізнатися, чи дійсний сертифікат. ЦС перевіряє стан сертифіката в ході процесу валідації, щоб переконатися, що його немає в його списку відкликаних сертифікатів (Revocation List).

- **Оновлення сертифікатів**

  Сертифікат повинен оновлюватися користувачами автоматично або вручну, як тільки його термін дії закінчується. Крім того, при оновленні

сертифіката можна обрати, чи створювати нові закриті та відкриті ключі.

- **Скасування сертифікатів**

  У випадку, якщо сертифікат більше не потрібен, його і будь-які резервні копії, пов'язані з цим сертифікатом, повинні бути скасовані. Це дає гарантію, що сертифікат ніким не використовується і ніким не скомпрометований.

- **Перевірка сертифіката**

Цей етап включає в себе створення, відстеження, відкликання та закінчення терміну дії сертифікатів. Можливо також відстежувати кожне успішне використання сертифіката.

- **Відкликання сертифікату**

Всі видані Центром сертифікації сертифікати мають термін дії. Для безпеки сертифікат повинен бути відкликаний до закінчення цього терміну. ЦС додає його у свій список відкликаних сертифікатів. Також сертифікат може бути відкликаний, якщо особа, якій він належить, залишає компанію або секретний ключ або сертифікат скомпрометовані.

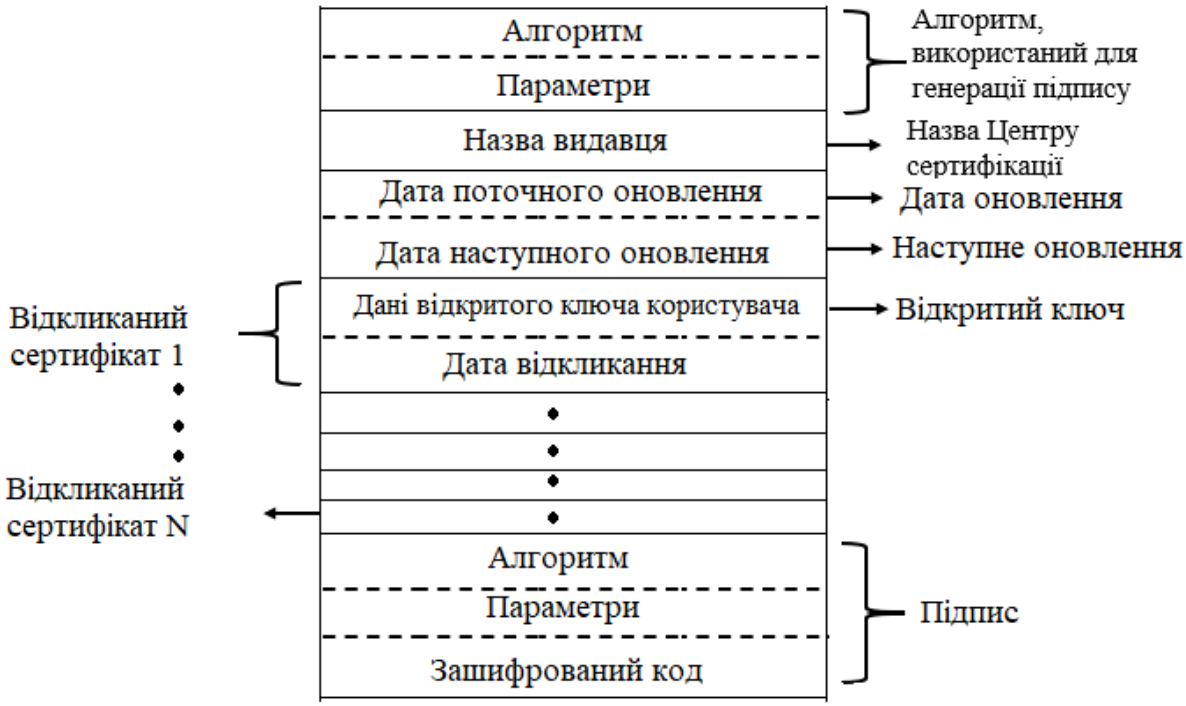


Рисунок 2.6. Структура X.509 Revocation List

X.509 Revocation List – це колекція відкликаних сертифікатів. Сертифікат, у якого спливає термін дії, відкликається і додається у Revocation List.[1]

Передача даних між кінцевими користувачами із різними доменами може не відбутися, якщо їх сертифікати підписували різні Центри сертифікації. Виникає задача обміну сертифікатами між самими ЦС. Для її вирішення вводиться інший ЦС, який видає свої сертифікати підлеглим ЦС. Тепер кінцеві користувачі можуть автентифікуватися між собою, обмінюючись сертифікатами на шляху до іншого по ієрархічному дереву.

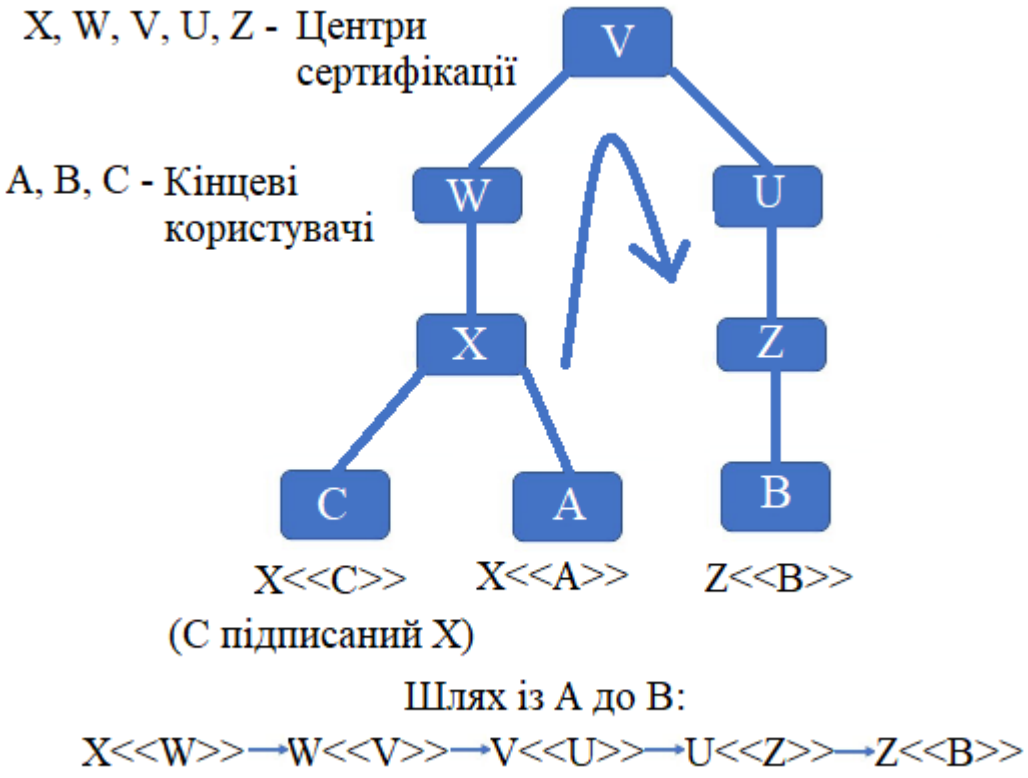


Рисунок 2.7. Принцип комунікації між кінцевими користувачами різних Центрів сертифікації

Ієрархія PKI може мати один або кілька рівнів. У однорівневому середовищі, кореневим ЦС буде єдиний наявний. Якщо середовище складається з декількох рівнів, кореневий центр сертифікації видаватиме сертифікати підлеглого ЦС прямо під кореневим каталогом. Таким чином,

зникає необхідність постійно звертатися до сервера кореневого центру сертифікації. Видані сертифікати запитуються користувачами прямо із найближчого підлеглого ЦС, дозволяючи кореневому ЦС відключатися. Таке відключення підвищує безпеку середовища PKI, оскільки ніхто не доступається напряму до сервера. Кількість рівнів для використання залежить від призначення середовища PKI, вимог безпеки або довіри, які ви хочете встановити у середовищі.[11]

На наведеному Рисунку 2.7 видно, що кожен ЦС надає права суб-ЦС на підписання цифрових сертифікатів. Всі сертифікати розміщені в кінці ієрархії. Вони приймаються пристроями і затверджуються підрозділом ЦС, який і генерує їх підписує.

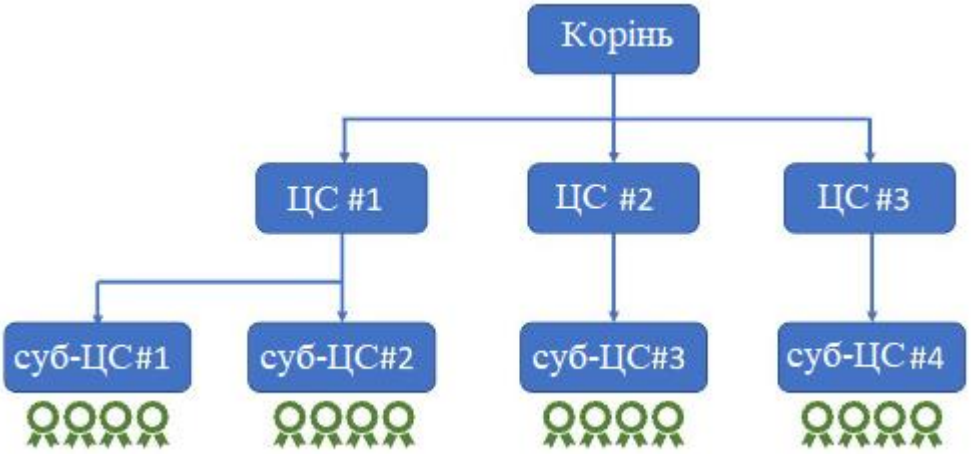


Рисунок 2.7. Ієрархія сертифікації X.509

Суб-ЦС, мають свій сертифікат, який затверджується цифровим підписом вищого на рівень центру сертифікації. На вершині PKI знаходиться головний ЦС, що є основою і коренем цієї ієрархії. Крім цього, причинами того, що середовище PKI повинно бути організоване в ієрархії, також визначаються, що вона дозволяє заборонити доступ до вибраних рівнів у разі витоку або компрометації закритого ключа.

Основною перевагою PKI реалізованої за допомогою ієрархій деревовидних типів є те, що вона дозволяє творцю керувати

скомпрометованою подією. Це також одна з причин, по якій сертифікати видаються не безпосередньо через кореневий ЦС, а через підлеглі ЦС, які знаходяться на рівень нижче, оскільки в разі аварійної ситуації всім пристроям в цій області потрібно бути відкликаними. Одна із основних переваг полягає в тому, що один суб-ЦС здатний генерувати більше мільйона сертифікатів, що використовуються для аутентифікації мільйонів пристроїв тільки з одним відкритим ключем суб-ЦС.[11]

2.3. Стек технологій для реалізації SSO системи

Не зважаючи на те, що проект SSO сервісу не є масштабним, головна його перевага – це безпека при комунікації між кінцевими користувачами. Важливо вибрати мову програмування, яка забезпечить стабільну роботу системи.

Розглянемо наступні варіанти мов програмування для реалізації сервісу:

- PHP (Фреймворк Laravel)
- Java (Фреймворк Spring)
- Python (Фреймворк Django)
- C# (Фреймворк .NET Core)
- JavaScript (Платформа Node.js)

Відкинемо вузькопрофільну та багату на недоліки мову програмування PHP. Заточений під програмування для Інтернету PHP не володіє можливостями для розвитку прогресивної системи єдиного входу, яка в майбутньому може розвинутись до корпоративної. Така перспектива є несумісною із обмеженим PHP.

Значно серйознішою вадою є безпека, а точніше її відсутність у добре вивченому недобросовісними програмістами PHP. Засобів безпеки недостатньо, адже зловмисники завжди попереду і знаходять вразливості

швидше, ніж розробники встигають їх виправити.

Через брак продуктивності, в наступну чергу відкинемо Python. Більшість розробників сходяться на думці, що ця мова не така швидкодійна, якою могла б бути. Для сервісу, завдяки якому користувач має доступатися до сторінок із авторизацією, не вводячи логіну та паролю, потрібно ввійти на сторінку настільки швидко, щоб зробити враження невимушеного переходу. Якщо користувачу доведеться чекати, поки програма на Пайтоні зреагує, то це нівелює переваги безпарольного входу.

Мови програмування C# та Java зручні у випадку розширення проекту та його підтримки. Вони також надають великий функціонал, а різноманітні фреймворки дозволяють швидко реалізовувати поставлені задачі. Попри всі переваги, в Java зможемо написати лише серверну частину проекту, а для фронтенду доведеться використовувати іншу мову. Оскільки обидві частини сервісу розробляються самостійно, а не командою, простіше використовувати одну мову, ніж функціонал декількох.

Мінуси C# в обмеженій кількості літератури для вивчення. Ця мова програмування досить легко дизасемблюється. Це означає, що програмний код може бути отриманий і вивчений конкурентами. Не так багато літератури можна знайти також для сучасних версій фреймворка .NET Core, який може зможе знадобитися для проекту. Це робить вивчення мови тривалим, що не вписується у короткі строки розробки.

Залишилось розглянути просту та популярну мову – JavaScript. ЇЇ оптимально використовувати у розроблюваному проекті з багатьох причин.

По-перше, JavaScript може бути дуже швидким, так як здебільшого запускається в браузері клієнта. Немає необхідності компілювати код перед його запуском, тому JavaScript не сповільнюється через виклики до сервера.

По-друге, JavaScript можна вставити на будь-яку веб-сторінку або використовувати в багатьох різних видах програм, на відміну від PHP чи

інших скриптових мов.

По-третє, JavaScript має велику підтримку з боку платформи Node.js. Це дає можливість застосовувати одну мову на клієнті і сервері. Зменшується об’єм коду і на клієнті, і на стороні сервера. Важливо пам’ятати, що часто об'єкти з однаковими назвами можуть виконувати абсолютно різні функції в браузері і на бекенді.

Для клієнтської частини буде обрано фреймворк Angular та бібліотеку React. Вони використовуватимуться у візуальній частині проекту.

Що стосується бази даних, то серед можливих варіантів, таких як MS SQL, MySQL, PostgreSQL, обрано PostgreSQL. Це безплатна, широко розповсюджена open-source база даних з функціоналом, ідентичним до платних аналогів. Створена з використанням об'єктно-реляційної моделі, база підтримує широкий спектр типів даних. Перевагою також є багата документація, а доступні книги, статті та уроки полегшать ознайомлення із PostgreSQL.

ВИСНОВКИ ДО РОЗДІЛУ

У даному розділі було розглянуто функціональні задачі SSO на основі протоколу автентифікації Kerberos, а також принципи генерації, підпису та обміну цифровими сертифікатами X.509 (mutual authentication). Детально проаналізовано структуру та життєвий цикл сертифіката, який включає в себе реєстрацію, валідацію, оновлення, відстеження й відкликання в разі закінчення терміну дії. Останній фактор прийнято вважати найважливішим для встановлення безпечної передачі ідентифікаційних даних між кінцевими користувачами.

Ієрархічна модель засвідчення сертифікатів дозволяє відкликати або забороняти доступ до вибраних рівнів в разі витоку або компрометації закритого ключа.

Автентифікація сертифікатами убезпечує користувача у разі, якщо його сертифікатом заволодіють зловмисники, вони не матимуть змоги розшифрувати його без приватного ключа. Це швидка та легка в застосуванні технологія для безпарольного входу.

В останньому пункті розділу було обрано мову програмування та фреймворки, для комфортної розробки проекту, спираючись на основні функціональні задачі розроблюваного сервісу.

РОЗДІЛ 3. СИСТЕМА ЄДИНОГО ВХОДУ НА БАЗІ ЦИФРОВИХ СЕРТИФІКАТІВ

3.1. Опис запропонованого сервісу

Пропонується сервіс, який реалізовує легке безпарольне користування сайтами та додатками для одного користувача або деякої кількості працівників компанії. В якості ідентифікаційних даних для входу (логіна чи електронної пошти та паролю) використовуються короткострокові цифрові сертифікати стандарту X.509, які мають автентифікацію Kerberos. Також даний сервіс надає функціонал, який потрібен для контролю облікового запису, підключених програм та моніторингу життєвого циклу сертифікатів, які використовуються.

Завдяки застосуванню стандарту X.509, розроблюваний сервіс ніколи не потребує паролів у процесі користування. Натомість, потрібно використати власний пароль при первинній авторизації, закріплюванні сертифікату за користувачем, отриманні квитків Kerberos. Генерація цифрових сертифікатів та запит на їх підписання проводяться за допомогою іншого системного програмного забезпечення.

Коли термін дії сертифіката закінчується, X.509 намагається автоматично використовувати поточні квитки Kerberos для придбання нового.

Отриманий цифровий сертифікат автоматично видаляється, коли виходить із нього.

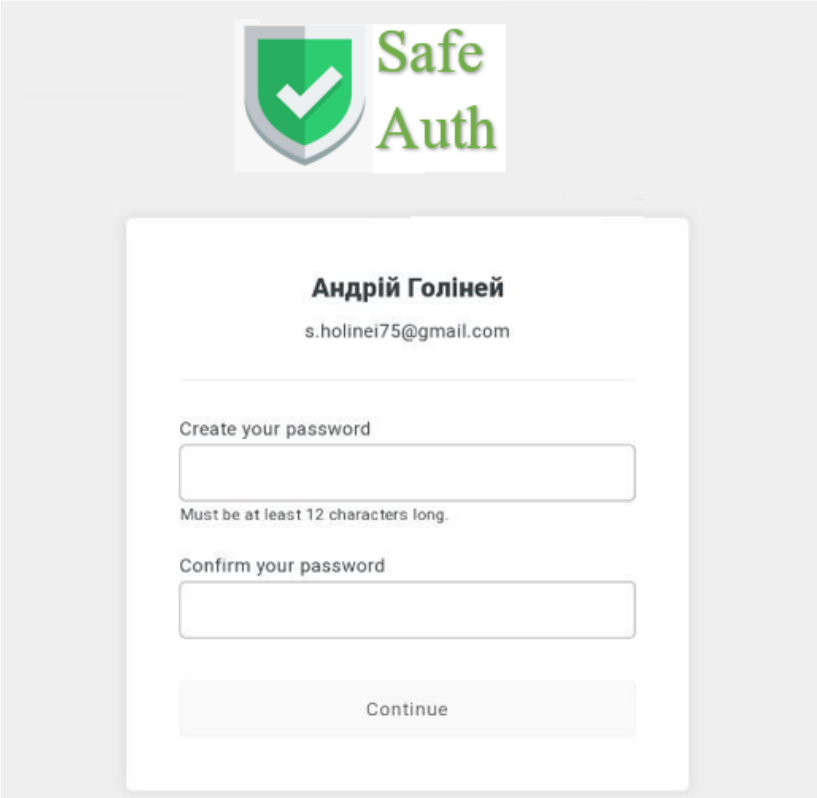


Рисунок 3.1. Первинна авторизація у SafeAuth із використанням паролю

Користувачам потрібна автентифікація лише один раз для входу в сервіс, щоб мати змогу аутентифікуватися на веб-серверах, які використовують аутентифікацію X.509, усуваючи необхідність власноруч вводити пароль при запиті веб-сайтами.

Оскільки пароль для них ніколи не пропонується, компрометація веб-серверів, для яких він пройшов автентифікацію, не порушує паролю.

Для сервісу передбачається дві ролі:

- Адміністратор сервісу
- Звичайний користувач

Роль адміністратора використовується у корпоративному режимі, коли треба делегувати багатьма факторами (працівниками, додатками, правами користувачів, сертифікатами). Адміністратор у свою чергу виконує функції власника, модератора, менеджера, технічної підтримки. Він має повний доступ для створення, оновлення та видалення користувачів, додатків, налаштувань,

прав та сертифікатів.

Адміністратор в якості технічної підтримки може створювати, оновлювати та видаляти користувацькі телефони, змінювати деяку інформацію про користувача, наприклад, повні імена, псевдоніми та адреси електронної пошти, змінювати статус користувача у випадку, якщо він покине компанію, а також переглядати звіти про автентифікацію.

Користувачеві, у свою чергу, доступний особистий кабінет, де можна додавати та переглядати додатки, змінювати інформацію про себе, керувати сертифікатами та персональними параметрами безпеки. Після первинного вводу паролю та підтвердження електронної адреси, користувачеві буде запропоновано ввести основну інформацію, таку як:

- ПІБ
- Номер телефону
- Електронну адресу
- Назву компанії або ім’я в мережі

Ці дані необхідні для подальшого заповнення реєстраційних форм системою в додатках, до яких буде доступатися користувач.

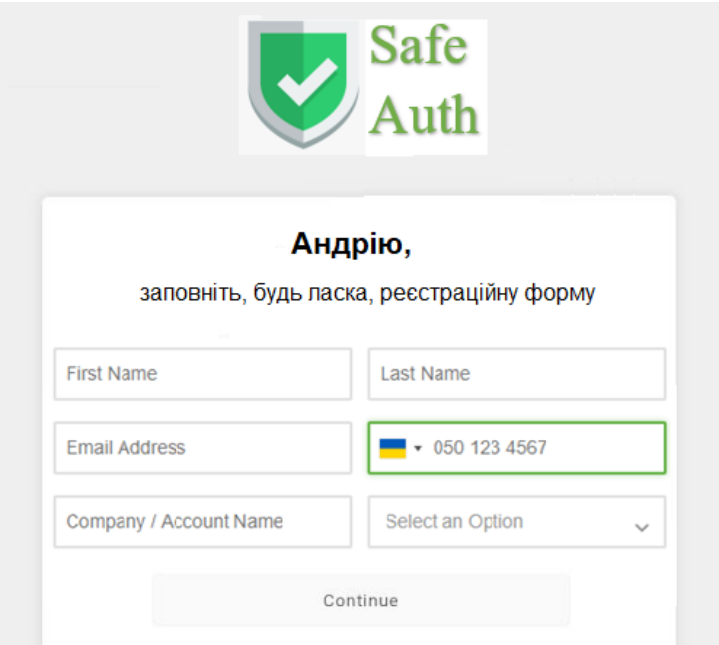


Рисунок 3.2. Запит основної інформації

3.2. Опис реалізації основних функцій

Система єдиного входу не потребує багатого інтерфейсу, так як більшість роботи виконується без прямої присутності користувача, а програмне забезпечення, відповідальне за генерацію сертифікатів, запити на підписання та звернення до Центрів сертифікації підключаються до системи аутентифікації через інтеграційні інтерфейси – HTTP REST API.

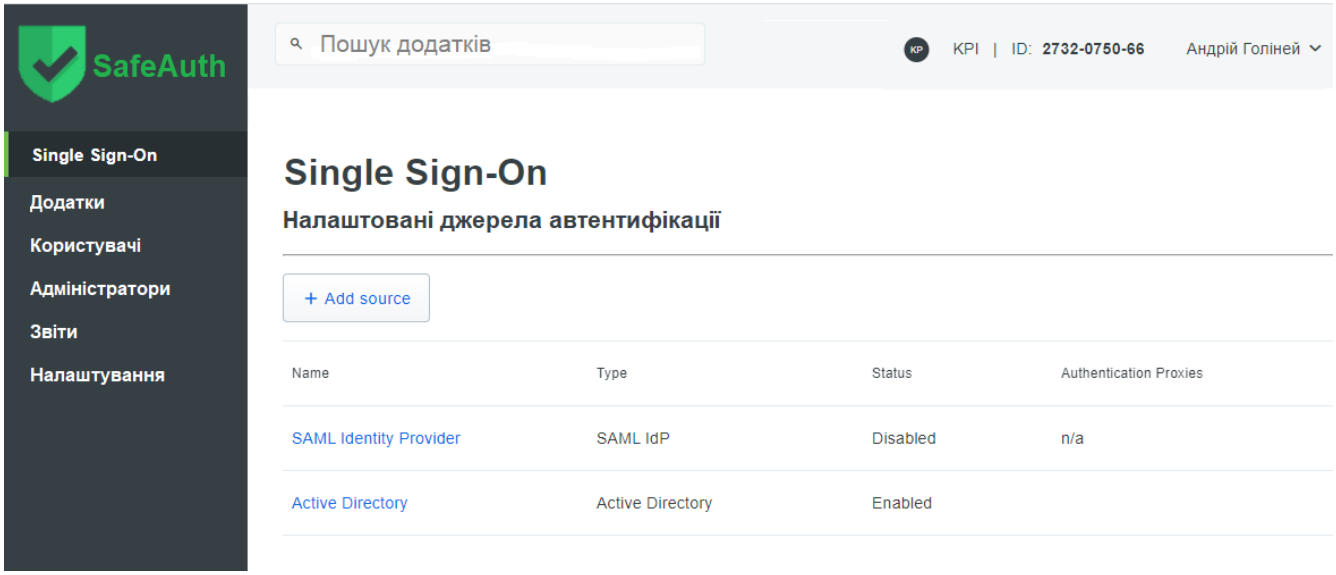


Рисунок 3.3. Інтерфейс головної сторінки сервісу

Для реалізації основних функцій керування SSO було обрано наступні пункти:

- Single Sign-on – налаштовані джерела автентифікації
- Додатки – підключені взаємною автентифікацією із використанням сертифікатів застосунки
- Користувачі – список підключених користувачів системи
- Адміністратори – перехід до керування обліковими записами в режимі адміністратора
- Звіти – журнал здійснених входів
- Налаштування – додаткові налаштування системи, не пов’язані із основними функціями роботи

Для швидкого доступу до потрібних пунктів використовується поле

пошуку. Воно дозволяє користувачеві швидко знаходити підключені до єдиного входу додатки. Адміністратор цим же полем може шукати профілі працівників, щоб швидко коригувати потрібну інформацію.

Робота сервісу починається із первинного налаштування служби єдиного входу за протоколом аутентифікації Kerberos. Розглянемо процес налаштування SSO на основній вкладці.

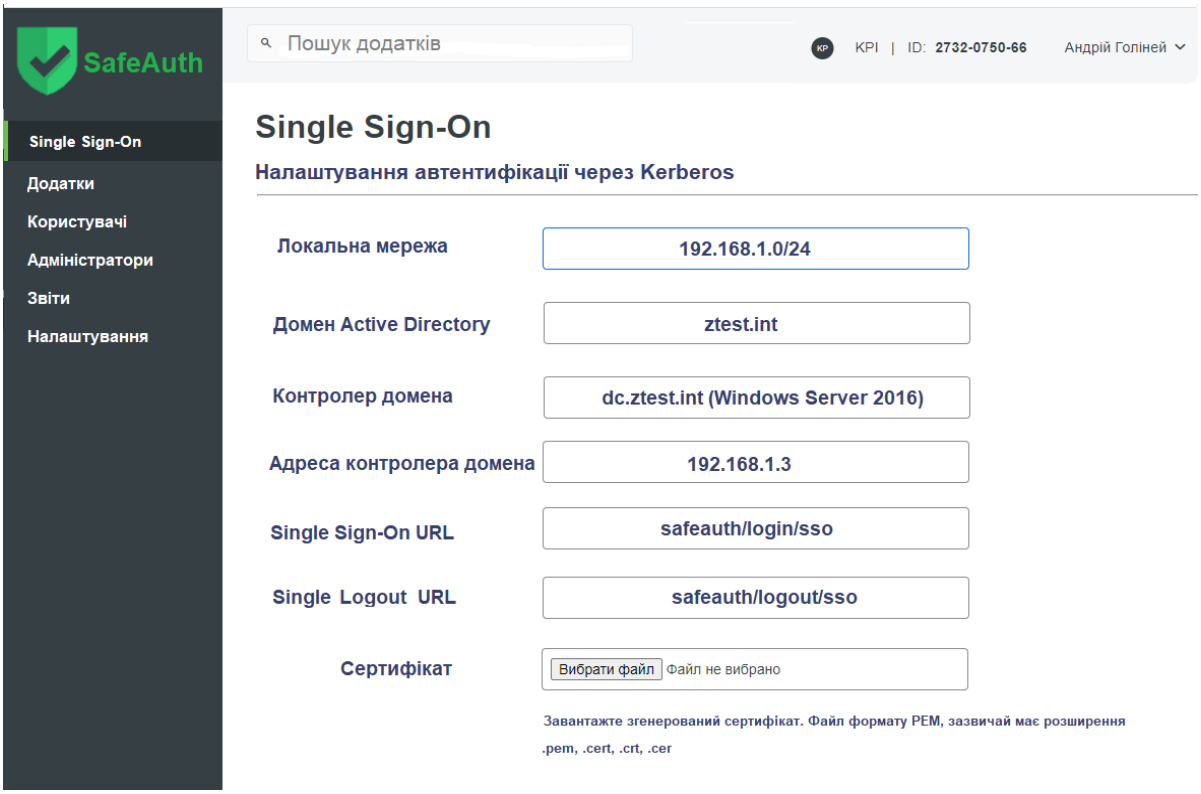


Рисунок 3.4. Налаштування параметрів автентифікації

Користувач працює через протокол Kerberos, попередньо налаштувавши доменне середовище Active Directory (AD). Функціонал середовища дозволяє реалізувати аутентифікацію в стилі Single Sign On. Таким чином, доменний користувач звільняється від повторних запитів на проходження аутентифікації.

Користувач вводить логін/пароль всього один раз - при вході в систему. В даному вікні він має змогу налаштувати середовище, посилання, за якими переходитиме система, а основне – прикріпити сертифікат. При наступному зверненні в Інтернет аутентифікація відбувається прозоро і автоматично.

3.3. Принцип створення сертифікатів

Процедура генерації сертифікатів, запити на підписання та звернення до Центрів сертифікації здійснюється окремо із застосуванням незалежного програмного забезпечення – X.509 Certificate Generator. Це багатоцільовий застосунок. Він може бути використаний для створення сертифікатів X.509 на файлах смарт-карт або PFX, попереднього перегляду сертифікатів або зміни розширень використання ключів.[?] https://x509-certificate-generator.software.informer.com/

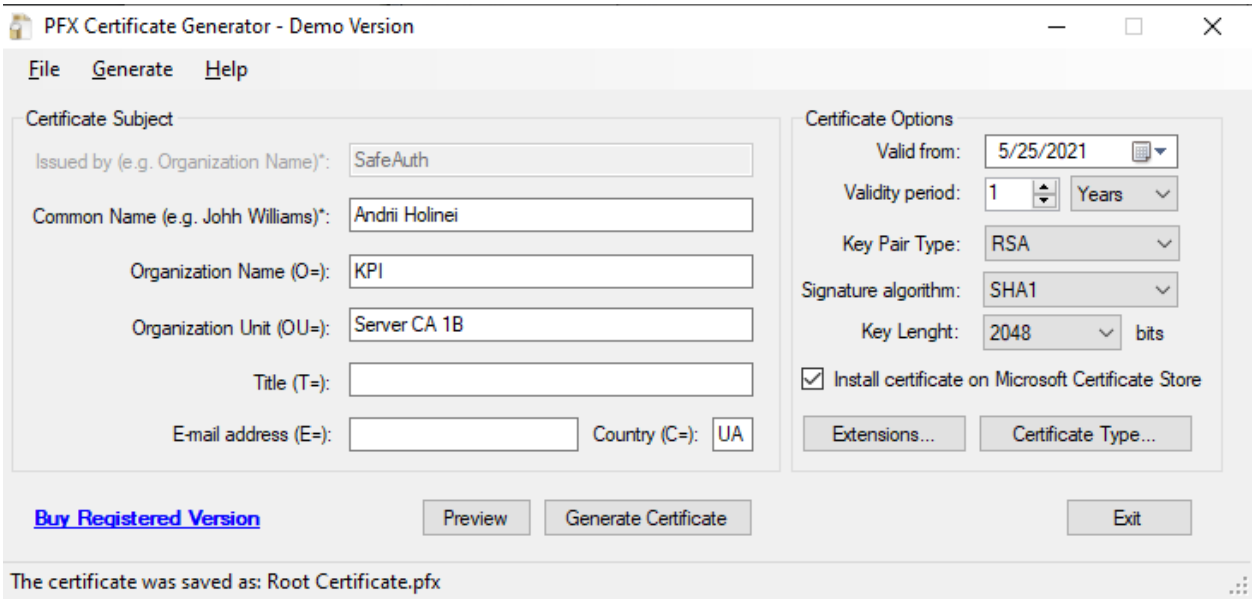


Рисунок 3.5. Основний інтерфейс X.509 Certificate Generator

Користувач заповнює поля назвами Центру сертифікації, власної організації, суб-ЦС, за бажанням вказує заголовок, ел. пошту та код країни. Програма дозволяє повністю налаштовувати такі параметри сертифікатів:

- Тип;
- Термін придатності;
- Алгоритм шифрування;
- Довжину ключа;
- Призначення ключа;

Запит на підписання здійснюється попередньо створеним кореневим сертифікатом. Обрати тип підписання можна на вкладці Generate. Згенерований та підписаний сертифікат зберігається в Microsoft Certificate

Store, якщо відмічений відповідний прапорець.[10]

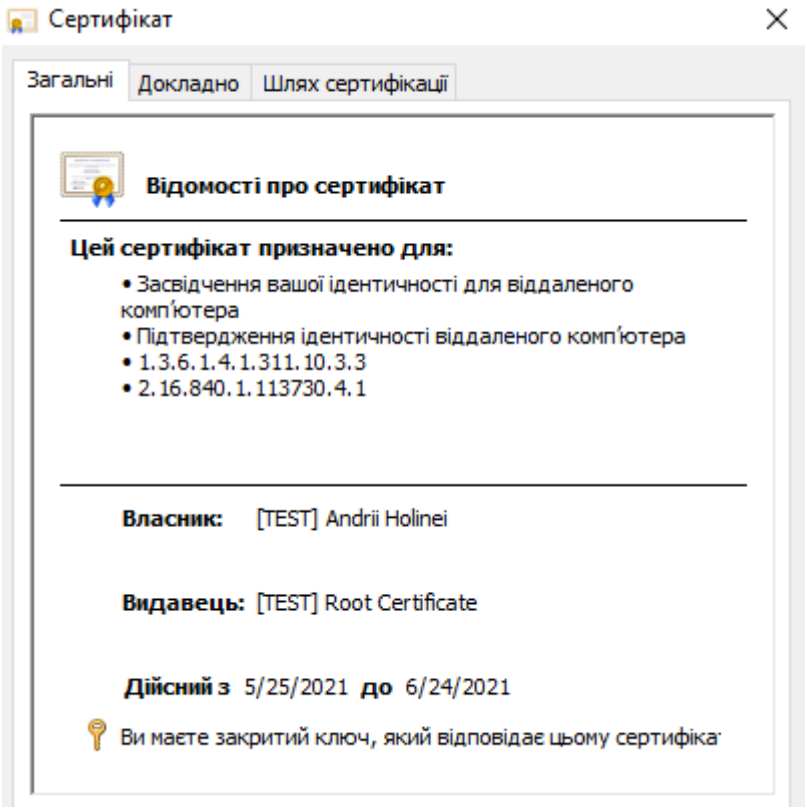


Рисунок 3.6. Підписаний цифровий сертифікат

Шлях сертифікації даного документу містить вказані користувачем Центри сертифікації, суб-ЦС та Кореневий сертифікат в ієрархічному порядку. Згенерований документ завантажується в систему та використовується для безпарольного входу.

3.4. Автентифікація у сторонніх застосунках

Після налаштування параметрів єдиного входу, користувач переходить на вкладку «Додатки», звідки починається робота із сервісами, які підтримують взаємну автентифікацію.

Механізм підключення додатків до розроблюваної системи SafeAuth розглянемо на прикладі сервісу Grafana. Це система візуалізації даних у вигляді панелі моніторингу із графіками, таблицями й іншими подібними елементами. Grafana підтримує технологію автентифікації через протокол

Kerberos, тому сумісна із розроблюваною системою.

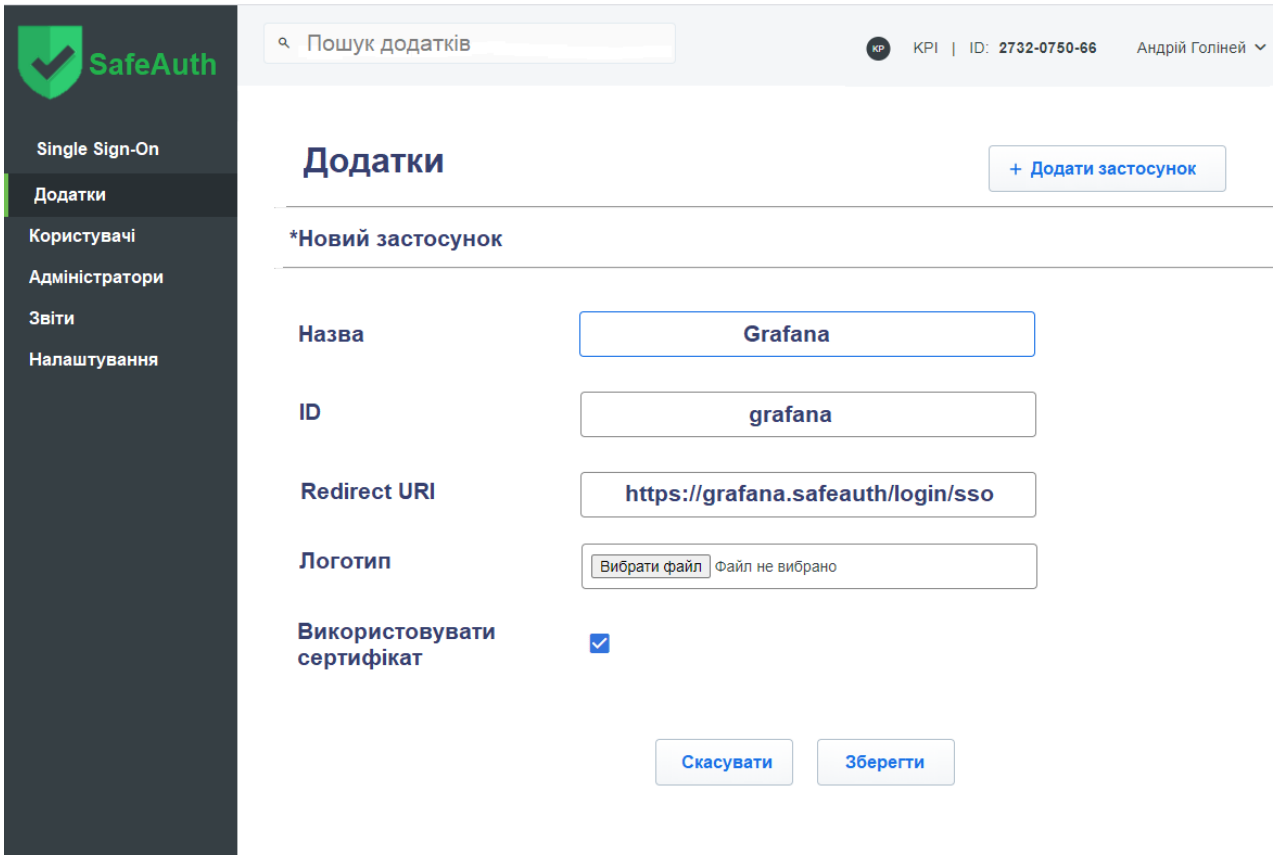


Рисунок 3.7. Налаштування параметрів для підключення застосунку

Для налаштування роботи користувач вказує ID застосунку – по ньому буде відбуватися ідентифікація запиту, а також посилання для перенаправлення (зазвичай складається із доменного імені з приставкою /login/sso, але краще переглянути документацію доданого сервісу).

Система дає можливість прикріпити зображення для додатку, яке можуть бачити користувачі в особистому кабінеті та адміністратори.

Для встановлення безпечного з’єднання слід проставити прапорець «Використовувати сертифікат». В протилежному випадку сервіс намагатиметься аутентифікувати користувача, використовуючи тільки облікові дані. Такий сценарій працює не зовсім коректно і не убезпечує з’єднання.

Наступний крок – налаштування параметрів на стороні Grafana.

Необхідно задати конфігурацію її сервера в частині секції аутентифікації.

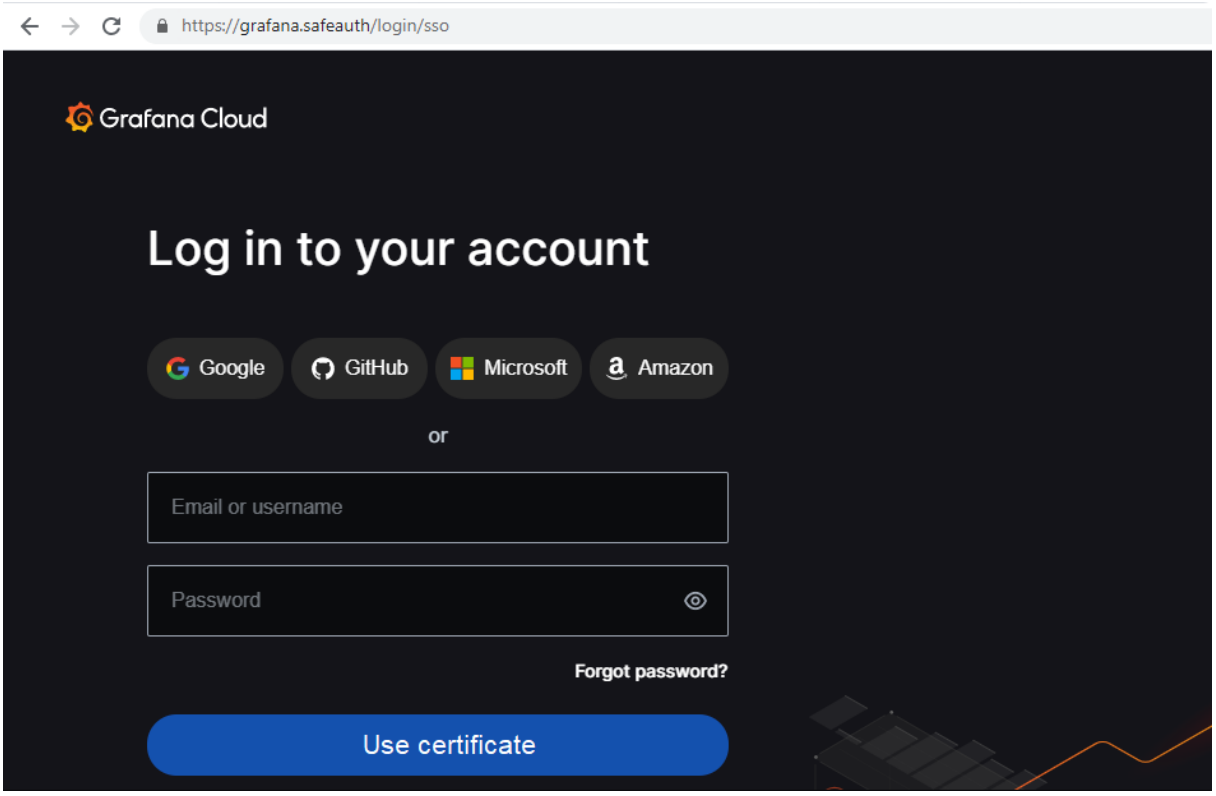


Рисунок 3.8. Інтерфейс автентифікації Grafana

Завдяки здійсненим налаштуванням, на сторінці сервісу Grafana з'являється можливість застосувати цифровий сертифікат (кнопка «Use certificate») для автентифікації. Система перенаправляє користувача в інтерфейс SafeAuth, де реєстраційні поля заповнюються введеними на етапі налаштування SSO даними. Після успішної аутентифікації користувач потрапляє в інтерфейс Grafana.

Сервіс дозволяє також додати в систему єдиного входу застосунки із десктопним вікном автентифікації. Цей функціонал потребує додаткових налаштувань та підтримується обмеженою кількістю застосунків. Виставляються параметри автентифікації, а у спеціальний шаблон записуються команди натиснення клавіш. Ідентифікатором такого застосунку буде назва процесу із диспетчера завдань, а назвою застосунку – назва вікна входу. В поле «Redirect URI» записується розташування шаблону

автентифікації, із попередньо складеного набору кроків.

3.5. Робота із системою з точки зору адміністратора

Адміністратор системи керує всіма параметрами, налаштовує їх для продуктивної роботи. Вхід адміністратора в систему ідентичний із входом звичайного користувача, відмінності тільки в доступі до певних вкладок. Користувач не може переглядати вкладку «Адміністратори», а на вкладці «Користувачі» не має права змінювати інформацію про інших.

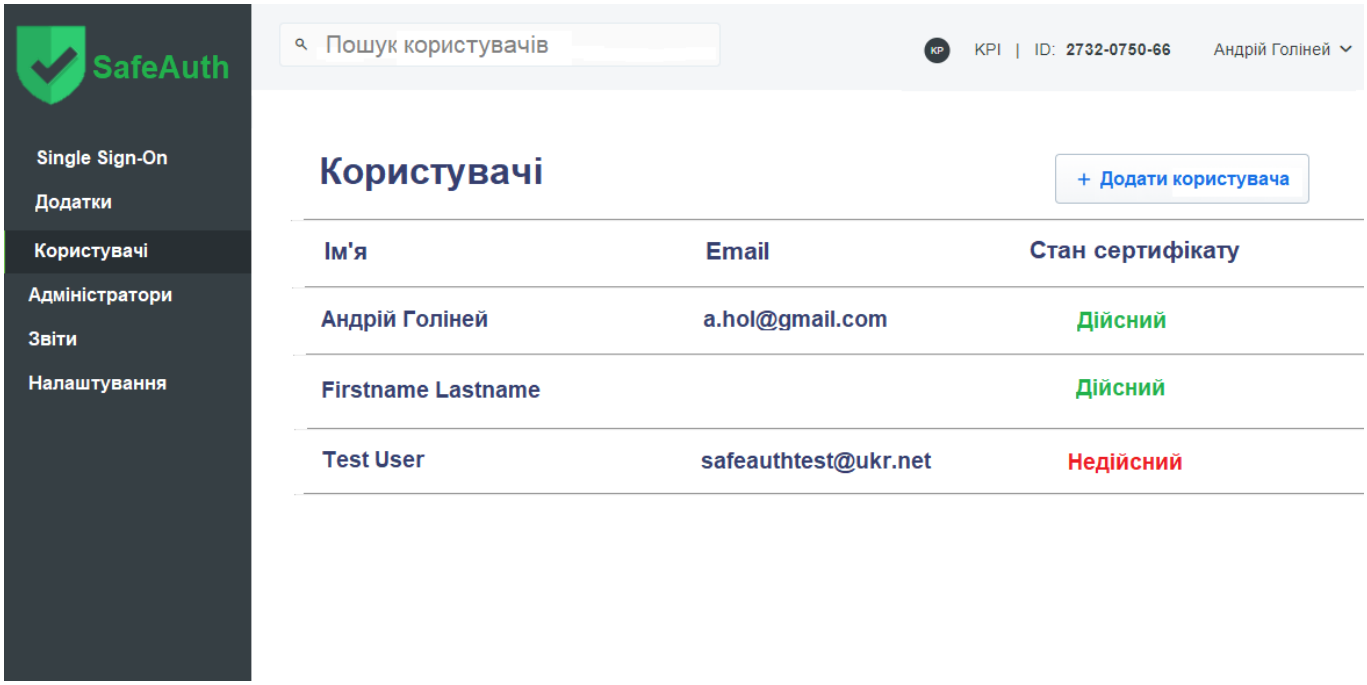


Рисунок 3.9. Вкладка «Користувачі» з інтерфейсу адміністратора

На даній вкладці адміністратору доступна інформація по кожному співробітнику: ім’я, логін, адреса електронної пошти і статус закріпленого за кожним користувачем сертифікату. Для перегляду детальної інформації та редагування профілю користувача необхідно натиснути на його ім’я в загальному списку.

У картці користувача адміністратору доступні такі операції:

- Реєстрація нового облікового запису;
- Призначення доступних співробітнику додатків;

- Перегляд і редагування інформації про користувача;
- Зміна облікових даних;
- Блокування / розблокування користувача, якщо обліковий запис загрожує безпеці системи;
- Управління цифровими сертифікатами, прив'язаними до профілю;
- Видалення користувача.

Стан прикріпленого сертифікату має два значення: дійсний та недійсний. Це корисна властивість, завдяки якій можна слідкувати за життєвим циклом сертифіката і замінити його у випадку закінчення терміну дії. Зняттям прапорця «прикріплений» адміністратор може відв'язати сертифікат від облікового запису в разі його втрати або компрометації.

ВИСНОВКИ ДО РОЗДІЛУ

Запропонована система єдиного входу, яка базується на ідентифікації особистості користувачів за допомогою цифрових сертифікатів. Розроблюваний застосунок має простий для щоденного застосування інтерфейс, інтуїтивно зрозумілий в освоєнні. Пропонуються нововведення відносно аналогів по типу індикації стану сертифікату, попереднього та єдиного заповнення форми обліковими даними. Таким чином сервіс має потенціал стати конкурентом системам єдиного входу на українському ринку та надати можливість користувачам безпечно входити в різноманітні сервіси, пам’ятаючи всього один пароль.

РОЗДІЛ 4. ПРОГРАМНЕ РІШЕННЯ ТА ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ СИСТЕМИ

4.1. Опис програмного рішення системи

Обравши мову програмування в попередньому розділі, опишемо доступні програмні технології реалізації системи єдиного входу, пояснимо взаємодію її компонентів один між одним.

Для того, щоб зекономити час і сили на вивчення різних мов програмування досягти бажаного результату була вибрана технологія NW.js. Це сукупність Node.js та Webkit, яка дозволить застосунок на JavaScript перетворити у настільний додаток для будь-якої операційної системи: Windows, Linux чи MacOS. Таким чином задовольняється потреба у веб-інтерфейсі сервісу для окремого користувача та десктопного продукту для компанії. Розробникові достатньо створити прогресивний веб-додаток, який використовуватимуть за обома призначеннями. [4]

Node.js транслюватиме написаний скрипт у машинний код, що необхідно при роботі з бекендом, а движок Webkit дозволить поєднати написаний код із браузерними API. Коду, написаному на JS, будуть доступні як Node.js модулі, так і стандартний браузерні API.

Прогресивний веб-додаток на NW.js має такі переваги:

- один інтерфейс на сайт та настільний додатки
- можливість використання в автономному режимі
- можливість використання характерного веб-додаткам адаптивного дизайну[5]

Розробку сервісу починаємо із встановлення основної платформи Node.js, а головним IDE слугуватиме продукт компанії JetBrains – WebStorm.

Завантаження здійснимо із офіційного сайту https://nodejs.org/en/.

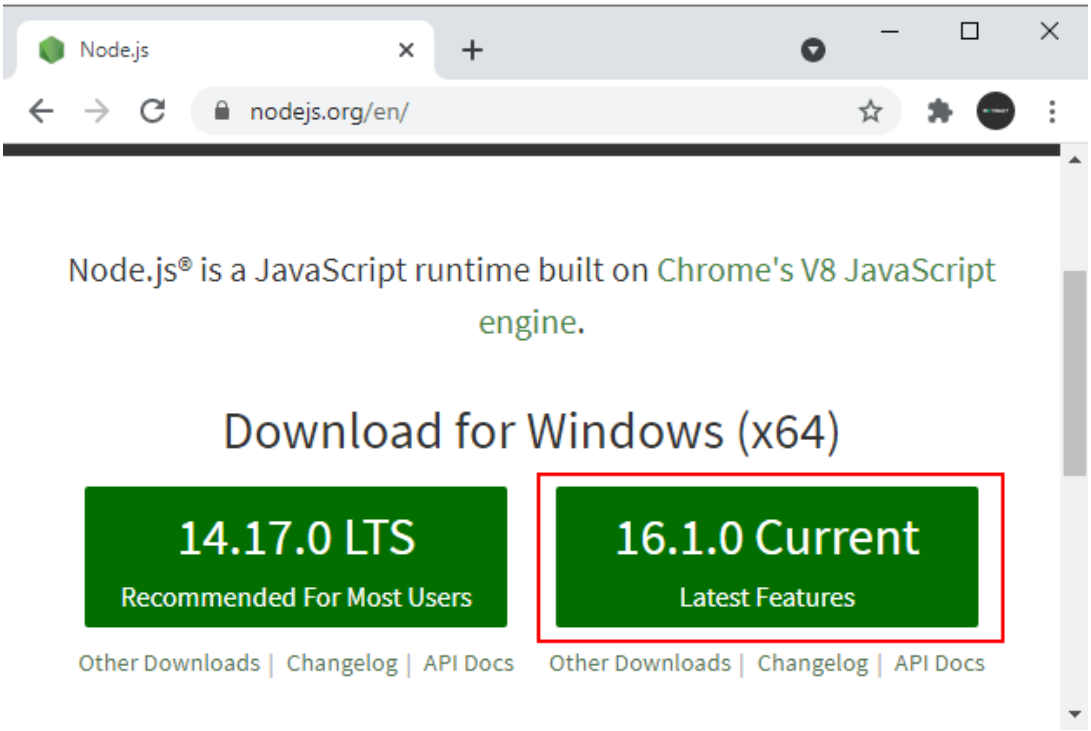


Рисунок 4.1 Вікно завантаження платформи Node.js

Для встановлення веб-сайту як додатку на ПК, потрібно попередньо завантажити із бібліотеки відкритого коду github.com репозиторій з вихідним кодом для конкретної ОС (у нашому випадку Windows). За допомогою вбудованого в Node.js менеджера пакетів npm, встановлюємо готову бібліотеку nw-builder. Для цього в командному рядку пропишемо:

```
> npm install nw-builder -g
```

Прапорець `-g` вказує, що збирач для NW.js встановлено глобально.

Для того, щоб зібрати застосунок виконуємо команду:

```
> nwbuild -p [назва платформи] -o [шлях до папки для зібраної версії] [шлях до програми]
```

В назві платформи можуть бути наступні значення: linux32, linux64, win32, win64, osx32, osx64.

Інформація для побудови застосунку береться із файлу package.json:

```
{
        "name": "hello-world",
        "version": "1.0.0",
        "description": "First application",
        "main": "index.html",
         "author": "Developer",
        "window": {
                 "toolbar": false,
                "width": 500,
                 "height": 200
         }
}
```

Рисунок 4.2. Вміст файлу package.json

В main записуємо файл з розміткою index.html, який буде точкою входу в додаток.

4.2. Розробка серверної частини системи SSO

Серверна частина системи єдиного входу буде реалізована у вигляді API (прикладний програмний інтерфейс), але необхідно попередньо налаштувати обраний протокол автентифікації – Kerberos. Для Windows модуль Kerberos надає окремий API до якого будуть надходити запити на взаємну аутентифікацію (перевірка дійсності ключів, видача токенів). [13]

Перевагою використання даного протоколу є те, що він встановлений за замовчуванням у операційній системі та підтримується більшістю популярних браузерів, тому не потребує налаштувань зі сторони клієнта.

У режимі адміністратора здійснюються наступні дії:

1. В Active Directory задається SPN-назва сервісу (наприклад HTTP/webservice.example.com@EXAMPLE.COM). Ця назва передається клієнтськими браузерами в HTTP-заголовку і дозволяє запитувати токен для даного сервісу, а Kerberos-клієнту через даний сервіс на основі сертифікату перевіряти справжність користувачів.

2. Kerberos-клієнту надається сертифікат.

```
        //cut phrase "Negotiate "
        var ticket = req.headers.authorization.substring(10);

        //init context
        kerberos.authGSSServerInit("HTTP", function(err, context) {
            //check ticket
            kerberos.authGSSServerStep(context, ticket, function(err) {
                //in success context contains username
                res.set( 'WWW-Authenticate', 'Negotiate ' + context.respons
e);

                res.send(context.username);
            });
        });
```

Рисунок 4.3. Код перевірки отриманого токену

Отримавши на свій GET-запит код відповіді 401 («Not Authorized»), а в заголовку «WWW-Authenticate: Negotiate», браузер робить запит до Key Distribution Centre та очікує отримати спеціальній токен сервісу. Метод «authGSSServerStep» відправляє токен на перевірку і якщо він містить назву клієнта, то він автентифікується у сервісі.[13]

4.2.1. Опис всіх запитів до API

Опишемо основні запити обміну даними між інтерфейсом системи у веб або настільному додатку та сервером:

1. GET /safeauth/login/sso – запит на вхід у застосунок. Принцип даного запиту являється основним у роботі системи, використовується при налаштуванні входу, що був описаний в попередньому розділі.
2. GET /safeauth/logout – запит на вихід користувача із сервісу.
3. GET /safeauth/application/{id} – запит на отримання застосунку із наявних на вкладці «Додатки». Параметром id вказаний ідентифікатор самого застосунку.
4. GET /safeauth/user/{id} – запит на отримання картки користувача із

наявних на вкладці «Користувачі». Параметром id вказаний ідентифікатор самого користувача.

5. POST /login/sso/callback– запит на отримання відповіді від сервера. Повертає спеціальний токен сервісу, який пройшов перевірку сервером на співпадіння із ім’ям клієнта.
6. PUT /safeauth/updateuser/{id} – запит на зміну інформації про користувача.
7. PUT /safeauth/deactivateuser/{id} – запит на відключення користувача у разі виникнення загрози компрометації його облікових даних. Використовується для того, щоб у разі звільнення або тимчасової відсутності (відпустки, декрету) співробітника відсторонити від будь-яких дій у системі.
8. DELETE /safeauth/user/{id} – запит на видалення неактивного, звільненого або становлячого загрозу безпеці системи користувача.
9. DELETE /safeauth/application/{id} – запит на видалення неактивного, мало використовуваного або становлячого загрозу безпеці системи застосунку.

Вказані запити аналогічні як для звичайного користувача, так і для адміністратора, зважаючи на реалізацію у режимі адміністратора CRUD моделі запитів(create – створення, read – читання, update – оновлення, delete – видалення). Шлях запитів однаковий, різниця тільки в застосованому типі методу. Для отримання інформації використовується метод GET. Запит на створення потрібного об’єкту здійснює метод POST, на видалення – DELETE, запит на оновлення – PUT.

4.2.2. Документація API проекту

Фреймворк Swagger має функціонал для аналізу коду проекту та документації усіх запитів до прикладного інтерфейсу системи. Фреймворк

візуалізує усі http запити, які можуть бути надіслані до API.

Рисунок 4.4. Візуалізація запитів до API

Уся інформація про запити у відповідному форматі записується в додатково згенерований json файл.

Застосований фреймворк дозволяє не тільки переглянути шляхи до запитів, але й протестувати роботу API. При тестуванні розробник передає потрібні параметри та здійснює запит, у відповідь отримавши статус код.

Таким чином, реалізована достатня документація для серверної частини системи єдиного входу, що реалізовує основний функціонал завдяки представленим запитам та описаним технологіям.

4.2.3. Опис взаємодії системи із базою даних

Щоб додати функціональну можливість підключення бази даних до розроблюваної системи, необхідно завантажити для Node.js відповідний драйвер обраної бази даних – PostgreSQL. Як один із найкращих драйверів було обрано бібліотеку pg. Node-postgres - це колекція модулів node.js для

взаємодії із PostgreSQL. Вона підтримує всі необхідні в роботі компоненти: зворотні виклики, проміси, async/await, пулінги, потокові результати[8]. Також було використано додатковий фреймворк Express, який розширює можливості попередньо названої бібліотеки та надає можливість легко та швидко використовувати його у взаємодії з об'єктно-реляційною системою керування базами даних PostgreSQL.[12]

Роботу з БД було спроектовано через функції та запити, які звертаються до відповідних інтерфейсів. Такий спосіб дозволяє легко змінити систему управління або проектування запитів до БД, не змінюючи інтерфейси роботи з ними. Принцип створення коротких функцій та запитів дозволить застосовувати фреймворк Moq, щоб тестувати роботу сервісів без будь-якого впливу на систему управління базами даних. Також це зручніше при виконанні unit-тестів.

Опишемо приклад взаємодії для класу UserController, який реалізовує додавання користувача. Інтерфейс такого класу описує принцип CRUD, тобто його методи виконують створення картки користувача, отримання інформації про нього, редагування картки та видалення.

Додавання користувача здійснюється функцією query(), в яку напряму прописуємо запит на створення нового поля у БД, параметрами вказуємо список із імені та прізвища. Дана функція стандартна для всіх запитів, змінюємо тільки тип запиту до БД.

Редагування картки реалізоване подібним чином: ключове поле в таблиці збережеться, а редаговані поля будуть замінені на інформацію відповідного типу.

Видалення відбувається за ідентифікаційним номером користувача (id), розміщеним у ключовому полі, так само як і отримання інформації про нього.

Кожна функція приймає два значення: запит та відповідь («req» та «res»). Окремо створені файли із маршрутами та запитами. Щоб

використовувати створений контролер в маршрутах, необхідно експортувати об’єкт класу.

4.3. Проектування бази даних

Зважаючи на опис запропонованої системи, необхідно створити в базі даних таблиці для збереження інформації про користувачів, додатки, авторизацію, сертифікати, звернення в центр підтримки, історію успішних та неуспішних входів.

Інформацію про авторизацію для забезпечення високого рівня захисту можна зберігати в платних базах даних на віддалених захищених серверах, але PostgreSQL пропонує різноманітні параметри безпеки, що дозволить вийти на один рівень із платними аналогами. Ручне налаштування розробником наявного функціоналу дозволить підвищити рівень захисту, що є принциповим для системи єдиного входу. [12]

Обрана СУБД пропонує наступні можливості запобігання несанкціонованого доступу:

1. Обмеження кількості можливих підключень.

Явно вказуємо IP-адреси користувачів, які можуть робити SQL запити, у нашому випадку вказуємо тільки адміністраторів. Для цього встановлюємо наступні параметри в конфігураційному файлі:

```
listen_addresses = 'ip_1, ip_2, ip_3'.
```

2. Обмеження кількості одночасних підключень.

Змінюємо вказане за замовчуванням число 100, враховуючи максимальну можливу кількість одночасних підключень (+1 для адміністратора)

3. Зміна типу аутентифікації.

До бази даних також можемо застосувати згенеровані раніше сертифікати, додавши у полі METHOD параметр cert.

4. Обмеження часу аутентифікації в СУБД.

5. Шифрування каналу передачі даних.

Підключені застосунки розділено на декілька таблиць для швидкого пошуку. Також буде створена таблиця сертифікатів, яка буде реалізовувати зв’язок ‘один до одного’ з таблицею користувачів. Таблиця адміністраторів матиме зв'язок ‘один до багатьох’ із таблицею додатків і таблицею користувачів. Між таблицями користувачів і додатків зв'язок ‘багато до багатьох’.

Таблиці, які зберігають інформацію звернень до технічної підтримки, спроектовані таким чином, щоб запобігти конфліктним ситуаціям в базі даних. Було створено додаткову таблицю, щоб реалізувати відношення багато звернень від багатьох користувачів. Таблиця з такими зверненнями містить інформацію про текст звернення, час відправлення та самого відправника.

Окремо створена таблиця для звітів аутентифікації, вона містить інформацію про усі успішні та неуспішні підтвердження ідентичності.

ВИСНОВКИ ДО РОЗДІЛУ

В даному розділі було розглянуто програмне рішення системи єдиного входу. Описано усі додаткові фреймворки та технології, необхідні в розробці системи подібного типу. Інтерфейс сервісу розроблено таким чином, щоб легко адаптуватися під різні цілі використання: веб-сервіс для безпарольного входу між сторінками та настільний додаток для компанії. Адаптацію реалізовано за допомогою бібліотеки NW.js для платформи Node.js.

Проаналізовано принцип роботи протоколу Kerberos, а також функцію цифрових сертифікатів у процесі підтвердження ідентичності. Встановлено можливості клієнтів у ролі адміністратора і в ролі звичайного користувача із різними особливостями та обмеженнями для двох ролей.

Розглянуто основні API запити системи до сервера. Проведено документацію запитів, що реалізовують основний функціонал.

Спроектовано базу даних, описано принцип її взаємодії із системою та способи підвищення рівня безпеки БД. Обґрунтовано всі таблиці та зв’язки між ними згідно потреб системи.

ЗАГАЛЬНІ ВИСНОВКИ

Під час виконання дипломного проєкту розроблено систему єдиного входу, яка вирішує проблему багатопарольності та безпечної аутентифікації особистостей під час роботи в корпоративній або локальній мережі. Для порівняння були проаналізовані можливі системи аналоги, враховані до уваги їх недоліки та переваги. Головною особливістю розробленої системи відносно аналогів є використання цифрових сертифікатів для автентифікації особистості. Також наявні інноваційні рішення, такі як формування звіту здійснених входів, індикація стану прикріпленого сертифікату.

Описано основний алгоритми безпарольного входу та реалізації необхідного для сервісу функціоналу, такого як: додавання застосунків і налаштування безпарольного входу між ними, пошук по додатках, завантаження згенерованих сертифікатів і закріплення їх за користувачем, редагування інформації про користувача, його деактивація або видалення, моніторинг виконаних дій, комунікація із технічною підтримкою.

Технологія SSO (Single Sign-On) реалізована через протокол аутентифікації Kerberos. Розглянуті основні принципи роботи даного протоколу, інфраструктуру відкритих ключів (PKI), обмін відкритими ключами та їх недоліки перед цифровими сертифікатами. Розглянуто способи генерації, запити на підписання, затвердження сертифікатів X.509, а також ієрархію Центрів сертифікації (CA).

Для реалізації проекту обрано мову програмування, яка найкраще задовольняла потреби системи такого класу – JavaScript. Особливості даної мови дозволили використати її для розробки обидвох частин сервісу: клієнтської та серверної. Для зручності розробки використано відповідний до JavaScript перелік фреймворків.

Проект був оптимізований із дотриманням усіх параметрів безпеки,

необхідних системам такого класу. Враховуючи особливості застосованого протоколу, забезпечено шифроване з’єднання між обома сторонами для взаємної аутентифікації. Передачу за зберігання цифрових сертифікатів захищено додатковими параметрами доступу до БД. Для полегшення розробки сервісу під настільну та веб-версію, всі запити до API було задокументовано за допомогою фреймворку Swagger.

Розроблена система безпарольного входу є безпечною та зручною реалізацією поставленої задачі розробки системи SSO аутентифікації, яка може конкурувати з її іноземними та українськими аналогами.

ПЕРЕЛІК ПОСИЛАНЬ

1. The KX.509 Protocol [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/2377000_The_KX509_Protocol
2. Інтегрована система електронної ідентифікації [Електронний ресурс] – Режим доступу до ресурсу: https://id.gov.ua/
3. Алгоритми та моделі організації доступу до веб-ресурсів на основі систем одноразової аутентифікації користувачів [Електронний ресурс] – Режим доступу до ресурсу: http://elartu.tntu.edu.ua/bitstream/lib/27591/2/TSTUSJ_2007v12n4_Karpinskyy_M-Algorithms_and_models_115-126.pdf
4. Best Single Sign-On (SSO) Software [Електронний ресурс] – Режим доступу до ресурсу: https://www.g2.com/categories/single-sign-on-sso
5. Аутентифікація і авторизація в мікросервісних додатках [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/dataart/blog/311376/
6. Порівняння і вибір системи єдиного входу [Електронний ресурс] – Режим доступу до ресурсу: https://www.anti-malware.ru/security/single-sign-on
7. Вступ до функції єдиного входу Kerberos [Електронний ресурс] – Режим доступу до ресурсу: https://support.apple.com/uk-ua/guide/deployment-reference-ios/apdf5b35aad2/web
8. Розбираємо токени авторизації у MS SSO [Електронний ресурс] – Режим доступу до ресурсу: https://blog.uaid.net.ua/sso/
9. Single Sign-On for Web Services [Електронний ресурс] – Режим доступу до ресурсу: https://help.sap.com/saphelp_nw73/helpdata/en
10. Microsoft Active Directory [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/azure/active-directory/manage-apps/what-is-single-sign-on
11. Що таке і як працює PKI Public Key Infrastructure [Електронний ресурс] –

Режим доступу до ресурсу: https://pki.com.ua/pki_help.html

12. Node + PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://node-postgres.com/
13. HTTP-Based Cross-Platform Authentication by Using the Negotiate Protocol [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/previous-versions/ms995329
14. Інструкція щодо авторизації користувачів для використання електронних сервісів [Електронний ресурс] – Режим доступу до ресурсу: https://www.city-adm.lviv.ua/lmr/lmrdownloads/forms/8_-Avtoryzatsiia-korystuvacha_1.pdf

ДОДАТОК 1

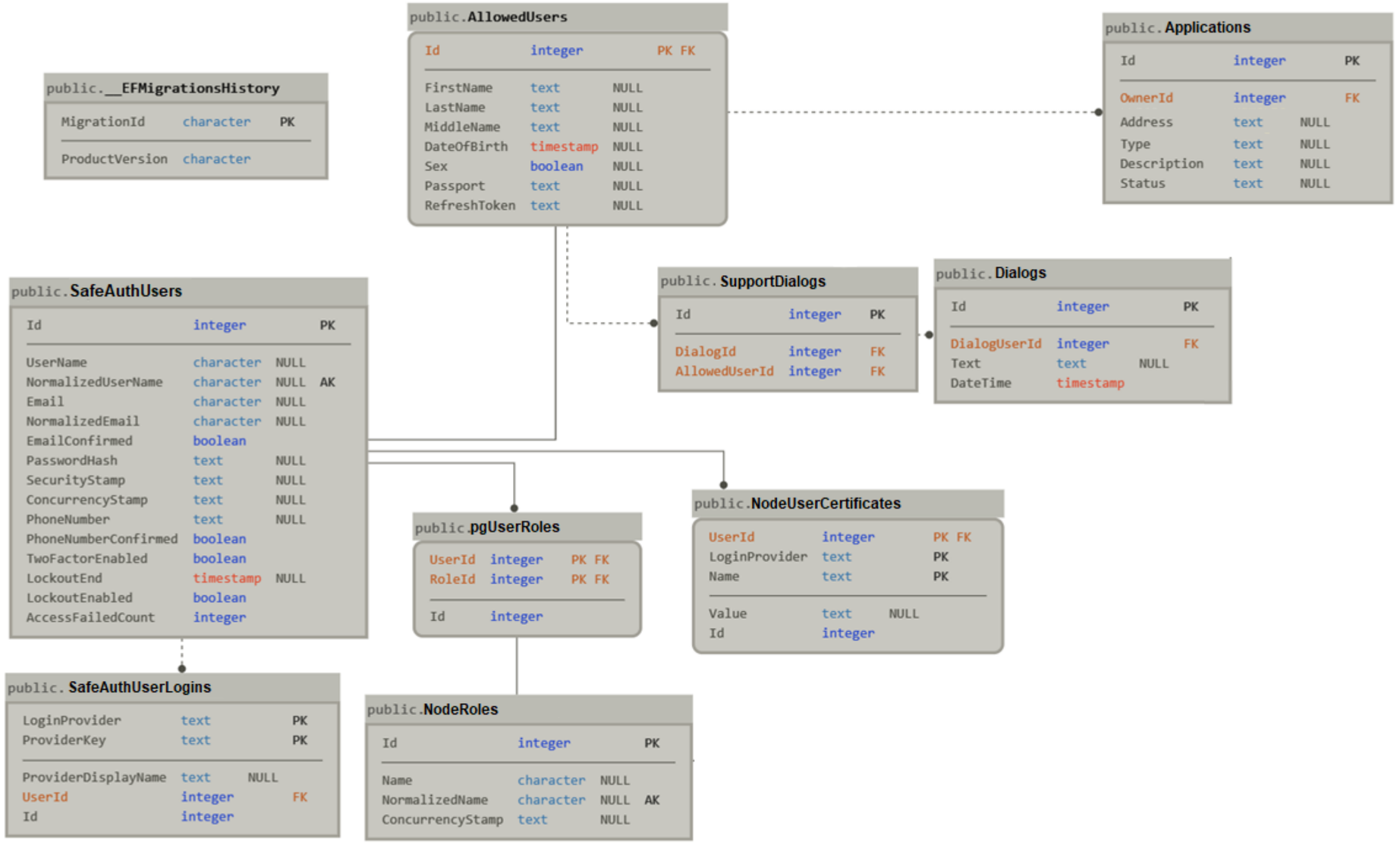
Система SSO автентифікації на базі цифрових сертифікатів

Схема бази даних

ІАЛЦ.467800.004 Д1

Аркушів 1

Київ 2021 р.



					ІАЛЦ.467100.004 Д1	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Голіней А.М.			Схема бази даних	Літ. / Маса / Масштаб
Перевір.		Роковий О.П.				Аркуш 1 / Аркушів 1
Н. Контр.		Сімоненко В.П.			Дипломна робота	НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ ІВ-72
Затв.						

Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № орис.

ДОДАТОК 2

Система SSO автентифікації на базі цифрових сертифікатів

Принципова схема безпарольного входу
ІАЛЦ.467100.005 Д2

Аркушів 1

Київ 2021 р.

Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № оригі.

					ІАЛЦ.467100.005 Д2	
Зм.	Арк.	№ докум.	Підпис	Дата	Принципова схема безпарольного входу	Літ. / Маса / Масштаб
Розроб.		Голіней А.М.				
Перевір.		Роковий О.П.				Аркуш 1 / Аркушів 1
Н. Контр.		Сімоненко В.П.			Дипломна робота	НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ ІВ-72
Затв.						

ДОДАТОК З

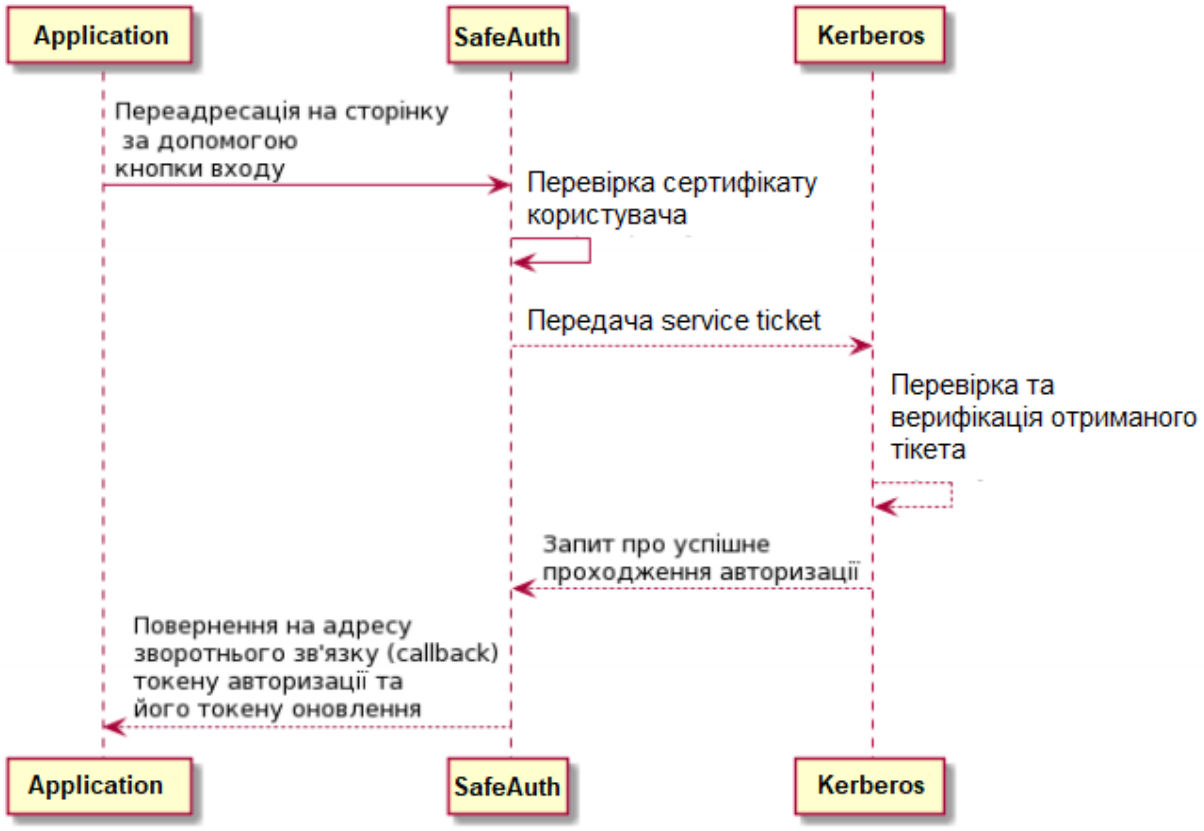
Система SSO автентифікації на базі цифрових сертифікатів

Функціональна схема авторизації

ІАЛЦ.467100.006 ДЗ

Аркушів 1

Київ 2021 р.



					ІАЛЦ.467100.006 ДЗ	
Змн.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Голіней А.М.			Функціональна схема авторизації	Літ. / Арк. 1 / Аркушів 1
Перевір.		Роковий О.П.				
Н. Контр.		Сімоненко В.П.				НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ ІВ-72
Затверд.						

ДОДАТОК 4

Система SSO автентифікації на базі цифрових сертифікатів

Текст програми

ІАЛЦ.467100.007 Д4

Аркушів 17

Київ 2021 р.

Index.js

```
const initPassport = require('./init');
var express = require('express');
var app = express();
var router = express.Router();

app.get('/', function (req, res) {

    //
    console.log('-----request-----');
    console.log(req.headers);


    if (!req.headers.authorization) {
        res.set( 'WWW-Authenticate', 'Negotiate' );

        //
        console.log('-----response-----');
        console.log(res._headers);

        res.status(401).send();

    } else {


        var KerberosNative = require('kerberos').Kerberos;
        var kerberos = new KerberosNative();
        var ActiveDirectory = require('activedirectory');
        var ad = new ActiveDirectory({
            "url": "ldap://ztest.com",
            "baseDN": "dc=ztest,dc=com",
            "username": "safeauthtest@ukr.net",
            "password": "qwerty1234"});
        //cut phrase "Negotiate "
        var ticket = req.headers.authorization.substring(10);

        //init context
        kerberos.authGSSServerInit("HTTP", function(err, context) {
            //check ticket
            kerberos.authGSSServerStep(context, ticket, function(err) {
                //in success context contains username
                ad.findUser(context.username, function(err, user) {
                    //get user groups
                    //if need filter by group name
                    // var opts = {filter: '&(member:1.2.840.113556.1.4.1941:=' + user.dn +
')(sAMAccountName=Ex*)'};
                    var opts = {filter: '&(member:1.2.840.113556.1.4.1941:=' + user.dn +
')'};
                    ad.find(opts, function(err, result) {
                        res.set( 'WWW-Authenticate', 'Negotiate ' + context.response);

                        //
                        console.log('-----response-----');
                        console.log(res._headers);
```

```
  var response = '<p>Ім’я користувача: '+ user.cn + '</p><p>Входить в групу:</p><ul>';
                          for (var i in result.groups) {response += '<li>' +
 result.groups[i].cn + '</li>';}
                          res.send(response);
                      })
                  });
              });
          });




      }
 });


      module.exports = router;
      app.use(router);
      app.listen(5000);
      module.exports = {initPassport};
```

Init.js

```
const passport = require('passport');
const passportSaml = require('passport-saml');
const {readFileSync} = require('fs');
const path = require('path');


passport.serializeUser((user, done) => {
    done(null, user);
})

passport.deserializeUser((user, done) => {
    done(null, user);
})

const cert = readFileSync(path.join(__dirname, 'onelogin.pem'), 'utf-8');

const strategy = new passportSaml.Strategy({
    entryPoint: '',
    cert
}, (profile, done) => {
    return done(null, profile)
})

function initPassport() {
    passport.use(strategy);
}

module.exports = initPassport;
```

Routes.js

```
const express = require('express');
const router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  const hasAuth = req.user !== undefined;
```

```
  const name = hasAuth && req.user.nameID;
  res.render('index', {hasAuth, name});
});

module.exports = router;
```

AuthRouter.js

```
const router = require('express').Router()
const { check } = require('express-validator')
const authController = require('../controllers/authController')

const roleMiddleware = require('../middlewares/roleMiddleware')

router.post('/registration', [
  check('username', 'The name cannot be empty').trim().escape().notEmpty(),
  check('password', 'Password length must be between 4 and 10 characters').isLength({ min: 4,
max: 10 })
], authController.registration)

router.post('/login', authController.login)

// router.get('/users', authMiddleware, authController.getUsers)
router.get('/users', roleMiddleware([ 'ADMIN' ]), authController.getUsers)

module.exports = router
```

AuthController.js

```
const User = require('../models/User')
const Role = require('../models/Role')
const { validationResult } = require('express-validator')
const jwt = require('jsonwebtoken')
const bcrypt = require('bcryptjs')
const { secretKey } = require('../config')

function generateAccessToken(id, roles) {
  const payload = { id, roles }
  return jwt.sign(payload, secretKey, {expiresIn: "8h"})
}

class AuthController {
  // ************* REGISTRATION
  async registration(req, res) {
    try {
      const validationErrors = validationResult(req)
      if (!validationErrors.isEmpty()) {
        return res.status(400).json({ message: 'Validation error', validationErrors })
      }

      const { username, password } = req.body
      const candidate = await User.findOne({ username })
      if (candidate) {
        res.status(400).json({ message: 'Such Username already exist (in Controller)' })
      }

      const hashPassword = bcrypt.hashSync(password, bcrypt.genSaltSync(5))
      const userRole = await Role.findOne({ value: 'USER' })
      const user = new User({ username, password: hashPassword, roles: [ userRole.value ] })
```

```
      await user.save()
      return res.json({ message: 'User created successfully' })

    } catch (e) {
      console.error(e)
      res.status(400).json({ message: 'Registration error (in Controller)' })
    }
  }

  // ************* LOGIN
  async login(req, res) {
    try {
      const { username, password } = req.body
      const user = await User.findOne({ username })
      if (!user) {
        return res.status(400).json(`User "${ username }" not found`)
      }

      const isPasswordValid = bcrypt.compareSync(password, user.password)
      if (!isPasswordValid) {
        return res.status(400).json(`Password is incorrect`)
      }

      const token = generateAccessToken(user._id, user.roles)
      return res.json({token})

    } catch (e) {
      console.error(e)
      res.status(400).json({ message: 'Login error (in Controller)' })
    }
  }

  // ************* GET USERS (for test purpose)
  async getUsers(req, res) {
    try {
      const users = await User.find()
      return res.json(users)

    } catch (e) {
      console.error(e)
    }
  }
}

module.exports = new AuthController()
```

authMiddleware.js

```
const jwt = require('jsonwebtoken')
const { secretKey } = require('../config')

module.exports = function (req, res, next) {
  if (req.method === 'OPTIONS') next()

  try {
    const token = req.headers.authorization.split(' ')[1]
    if (!token) {
      return res.status(403).json({ message: 'Not authorized user' })
    }
    req.user = jwt.verify(token, secretKey)
```

```
    next()
  } catch (e) {
    console.error(e)
    return res.status(403).json({ message: 'Not authorized user' })
  }
```

roleMiddleware.js

```
const jwt = require('jsonwebtoken')
const { secretKey } = require('../config')

module.exports = function (roles) {
  return function (req, res, next) {
    if (req.method === 'OPTIONS') next()

    try {
      const token = req.headers.authorization.split(' ')[1]
      if (!token) {
        return res.status(403).json({ message: 'Not authorized user' })
      }
      // FOR AUTHORIZATION (any) CHECK
      // Put into request 'user' object with payload (user ID & roles) from JWT
      // req.user = jwt.verify(token, secretKey)

      // FOR SPECIFIC ROLES (param 'roles') CHECK
      const {roles: userRoles} = jwt.verify(token, secretKey)

      let hasRole = false
      userRoles.forEach(role => {
        if (roles.includes(role)) hasRole = true
      })
      if (!hasRole) {
        return res.status(403).json({message: 'Access is denied'})
      }

      next()
    } catch (e) {
      console.error(e)
      return res.status(403).json({ message: 'Not authorized user' })
    }
  }
```

Users.js

```
const express = require('express');
const router = express.Router();
const passport = require('passport');

router.get('/login/sso',
    passport.authenticate('saml', {
        successRedirect: '/'
    }),
    (req, res) => {
        res.redirect('/');
    }
)

router.post('/login/sso/callback',
    passport.authenticate('saml', {
        successRedirect: '/'
```

```
    }),
    (req, res) => {
        res.redirect('/');
    }
)

router.get('/logout/', (req, res) => {
    req.logout();
    res.redirect('/');
})

module.exports = router;
```

App.js

```
const createError = require('http-errors');
const express = require('express');
const path = require('path');
const cookieParser = require('cookie-parser');
const logger = require('morgan');
const passport = require('passport')
const bodyParser = require('body-parser');
const cookieSession = require('cookie-session');

const indexRouter = require('./routes/index');
const userRouter = require('./routes/user');

const app = express();

require('./authentication').initPassport(app);

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.use(cookieSession({
  secret: 'my-secret'
}))

app.use(bodyParser.urlencoded({extended: true}));
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));
app.use(passport.initialize());
app.use(passport.session());

app.use('/', indexRouter);
app.use('/user', userRouter);


// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
```

```
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

package-lock.json

```
{
  "name": "sso-saml",
  "version": "0.0.0",
  "lockfileVersion": 1,
  "requires": true,
  "dependencies": {
    "accepts": {
      "version": "1.3.7",
      "resolved": "https://registry.npmjs.org/accepts/-/accepts-1.3.7.tgz",
      "integrity": "sha512-
Il80Qs2WjYlJIBNzNkK6KYqlVMTbZLXgHx2oT0pU/fjRHyEp+PEfEPY0R3WCwAGVOtauxh1hOxNgIf5bv7dQpA==",
      "requires": {
        "mime-types": "~2.1.24",
        "negotiator": "0.6.2"
      }
    },
    "array-flatten": {
      "version": "1.1.1",
      "resolved": "https://registry.npmjs.org/array-flatten/-/array-flatten-1.1.1.tgz",
      "integrity": "sha1-ml9pkFGx5wczKPKgCJaLZOopVdI="
    },
    "basic-auth": {
      "version": "2.0.1",
      "resolved": "https://registry.npmjs.org/basic-auth/-/basic-auth-2.0.1.tgz",
      "integrity": "sha512-
NF+epuEdnUYVlGuhaxbbq+dvJttwLnGY+YixlXlME5KpQ5W3CnXA5cVTneY3SPbPDRkcjMbifrwmFYcClgOZeg==",
      "requires": {
        "safe-buffer": "5.1.2"
      }
    },
    "body-parser": {
      "version": "1.18.3",
      "resolved": "https://registry.npmjs.org/body-parser/-/body-parser-1.18.3.tgz",
      "integrity": "sha1-WykhmP/dVTs6DyDe0FkrlWlVyLQ=",
      "requires": {
        "bytes": "3.0.0",
        "content-type": "~1.0.4",
        "debug": "2.6.9",
        "depd": "~1.1.2",
        "http-errors": "~1.6.3",
        "iconv-lite": "0.4.23",
        "on-finished": "~2.3.0",
        "qs": "6.5.2",
        "raw-body": "2.3.3",
        "type-is": "~1.6.16"
      }
    },
    "bytes": {
      "version": "3.0.0",
      "resolved": "https://registry.npmjs.org/bytes/-/bytes-3.0.0.tgz",
```

```
      "integrity": "sha1-0ygVQE1olpn4Wk6k+odV3ROpYEg="
    },
    "content-disposition": {
      "version": "0.5.2",
      "resolved": "https://registry.npmjs.org/content-disposition/-/content-disposition-
0.5.2.tgz",
      "integrity": "sha1-DPaLud318r55YcOoUXjLhdunjLQ="
    },
    "content-type": {
      "version": "1.0.4",
      "resolved": "https://registry.npmjs.org/content-type/-/content-type-1.0.4.tgz",
      "integrity": "sha512-
hIP3EEPs8tB9AT1L+NUqtwOAps4mk2Zob89MWXMHjHWg9milF/j4osnnQLXBCBFBk/tvIG/tUc9mOUJiPBhPXA=="
    },
    "cookie": {
      "version": "0.4.0",
      "resolved": "https://registry.npmjs.org/cookie/-/cookie-0.4.0.tgz",
      "integrity": "sha512-
+Hp8fLp57wnUSt0tY0tHEXh4voZRDnoIrZPqlo3DPiI4y9lwg/jqx+1Om94/W6ZaPDOUbnjOt/99w66zk+l1Xg=="
    },
    "cookie-parser": {
      "version": "1.4.5",
      "resolved": "https://registry.npmjs.org/cookie-parser/-/cookie-parser-1.4.5.tgz",
      "integrity": "sha512-
f13bPUj/gG/5mDr+xLmSxxDsB9DQiTIfhJS/sqjrmfAWiAN+x2O4i/XguTL9yDZ+/IFDanJ+5x7hC4CXT9Tdzw==",
      "requires": {
        "cookie": "0.4.0",
        "cookie-signature": "1.0.6"
      }
    },
    "cookie-session": {
      "version": "1.4.0",
      "resolved": "https://registry.npmjs.org/cookie-session/-/cookie-session-1.4.0.tgz",
      "integrity": "sha512-
0hhwD+BUIwMXQraiZP/J7VP2YFzqo6g4WqZlWHtEHQ22t0MeZZrNBSCxC1zcaLAs8ApT3BzAKizx9gW/AP9vNA==",
      "requires": {
        "cookies": "0.8.0",
        "debug": "2.6.9",
        "on-headers": "~1.0.2"
      }
    },
    "cookie-signature": {
      "version": "1.0.6",
      "resolved": "https://registry.npmjs.org/cookie-signature/-/cookie-signature-1.0.6.tgz",
      "integrity": "sha1-4wOogrNCzD7oylE6eZmXNNqzriw="
    },
    "cookies": {
      "version": "0.8.0",
      "resolved": "https://registry.npmjs.org/cookies/-/cookies-0.8.0.tgz",
      "integrity": "sha512-
8aPsApQfebXnuI+537McwYsDtjVxGm8gTIzQI3FDW6t5t/DAhERxtnbEPN/8RX+uZthoz4eCOgloXaE5cYyNow==",
      "requires": {
        "depd": "~2.0.0",
        "keygrip": "~1.1.0"
      },
      "dependencies": {
        "depd": {
          "version": "2.0.0",
          "resolved": "https://registry.npmjs.org/depd/-/depd-2.0.0.tgz",
          "integrity": "sha512-
g7nH6P6dyDioJogAAGprGpCtVImJhpPk/roCzdb3fIh61/s/nPsfR6onyMwkCAR/OlC3yBC0lESvUoQEAssIrw=="
```

```
        }
      }
    },
    "debug": {
      "version": "2.6.9",
      "resolved": "https://registry.npmjs.org/debug/-/debug-2.6.9.tgz",
      "integrity": "sha512-
bC7ElrdJaJnPbAP+1EotYvqZsb3ecl5wi6Bfi6BJTUcNowp6cvspg0jXznRTKDjm/E7AdgFBVeAPVMNcKGsHMA==",
      "requires": {
        "ms": "2.0.0"
      }
    },
    "depd": {
      "version": "1.1.2",
      "resolved": "https://registry.npmjs.org/depd/-/depd-1.1.2.tgz",
      "integrity": "sha1-m81S4UwJd2PnSbJ0xDRu0uVgtak="
    },
    "destroy": {
      "version": "1.0.4",
      "resolved": "https://registry.npmjs.org/destroy/-/destroy-1.0.4.tgz",
      "integrity": "sha1-l4hXRCxEdJ5CBmE+N5RiBYJqvYA="
    },
    "ee-first": {
      "version": "1.1.1",
      "resolved": "https://registry.npmjs.org/ee-first/-/ee-first-1.1.1.tgz",
      "integrity": "sha1-WQxhFWsK4vTwJVcyoViyZrxWsh0="
    },
    "ejs": {
      "version": "2.6.2",
      "resolved": "https://registry.npmjs.org/ejs/-/ejs-2.6.2.tgz",
      "integrity": "sha512-
PcW2a0tyTuPHz3tWyYqtK6r1fZ3gp+3Sop8Ph+ZYN81Ob5rwmbHEzaqs10N3BEsaGTkh/ooniXK+WwszGlc2+Q=="
    },
    "encodeurl": {
      "version": "1.0.2",
      "resolved": "https://registry.npmjs.org/encodeurl/-/encodeurl-1.0.2.tgz",
      "integrity": "sha1-rT/0yG7C0CkyL1oCw6mmBslbP1k="
    },
    "escape-html": {
      "version": "1.0.3",
      "resolved": "https://registry.npmjs.org/escape-html/-/escape-html-1.0.3.tgz",
      "integrity": "sha1-Aljq5NPQwJdN4cFpGI7wBR0dGYg="
    },
    "etag": {
      "version": "1.8.1",
      "resolved": "https://registry.npmjs.org/etag/-/etag-1.8.1.tgz",
      "integrity": "sha1-Qa4u62XvpiJorr/qg6x9eSmbCIc="
    },
    "express": {
      "version": "4.16.4",
      "resolved": "https://registry.npmjs.org/express/-/express-4.16.4.tgz",
      "integrity": "sha512-
j12Uuyb4FMrd/qQAm6uCHAkPtO8FDTRJZBDd5D2KOL2eLaz1yUNdUB/NOIyq0iU4q4cFarsUCrnFDPBcnksuOg==",
      "requires": {
        "accepts": "~1.3.5",
        "array-flatten": "1.1.1",
        "body-parser": "1.18.3",
        "content-disposition": "0.5.2",
        "content-type": "~1.0.4",
        "cookie": "0.3.1",
        "cookie-signature": "1.0.6",
```

```
        "debug": "2.6.9",
        "depd": "~1.1.2",
        "encodeurl": "~1.0.2",
        "escape-html": "~1.0.3",
        "etag": "~1.8.1",
        "finalhandler": "1.1.1",
        "fresh": "0.5.2",
        "merge-descriptors": "1.0.1",
        "methods": "~1.1.2",
        "on-finished": "~2.3.0",
        "parseurl": "~1.3.2",
        "path-to-regexp": "0.1.7",
        "proxy-addr": "~2.0.4",
        "qs": "6.5.2",
        "range-parser": "~1.2.0",
        "safe-buffer": "5.1.2",
        "send": "0.16.2",
        "serve-static": "1.13.2",
        "setprototypeof": "1.1.0",
        "statuses": "~1.4.0",
        "type-is": "~1.6.16",
        "utils-merge": "1.0.1",
        "vary": "~1.1.2"
      },
      "dependencies": {
        "cookie": {
          "version": "0.3.1",
          "resolved": "https://registry.npmjs.org/cookie/-/cookie-0.3.1.tgz",
          "integrity": "sha1-5+Ch+e9DtMi6klxcWpboBtFoc7s="
        }
      }
    },
    "finalhandler": {
      "version": "1.1.1",
      "resolved": "https://registry.npmjs.org/finalhandler/-/finalhandler-1.1.1.tgz",
      "integrity": "sha512-
Y1GUDo39ez4aHAw7MysnUD5JzYX+WaIj8I57kO3aEPT1fFRL4sr7mjei97FgnwhAyyzRYmQZaTHb2+9uZ1dPtg==",
      "requires": {
        "debug": "2.6.9",
        "encodeurl": "~1.0.2",
        "escape-html": "~1.0.3",
        "on-finished": "~2.3.0",
        "parseurl": "~1.3.2",
        "statuses": "~1.4.0",
        "unpipe": "~1.0.0"
      }
    },
    "forwarded": {
      "version": "0.1.2",
      "resolved": "https://registry.npmjs.org/forwarded/-/forwarded-0.1.2.tgz",
      "integrity": "sha1-mMI9qxF1ZXuMBXPozszZGw/xjIQ="
    },
    "fresh": {
      "version": "0.5.2",
      "resolved": "https://registry.npmjs.org/fresh/-/fresh-0.5.2.tgz",
      "integrity": "sha1-PYyt2Q2XZWn6g1qx+OSyOhBWBac="
    },
    "http-errors": {
      "version": "1.6.3",
      "resolved": "https://registry.npmjs.org/http-errors/-/http-errors-1.6.3.tgz",
      "integrity": "sha1-i1VoC7S+KDoLW/TqLjhYC+HZMg0=",
```

```
      "requires": {
        "depd": "~1.1.2",
        "inherits": "2.0.3",
        "setprototypeof": "1.1.0",
        "statuses": ">= 1.4.0 < 2"
      }
    },
    "iconv-lite": {
      "version": "0.4.23",
      "resolved": "https://registry.npmjs.org/iconv-lite/-/iconv-lite-0.4.23.tgz",
      "integrity": "sha512-
neyTUVFtahjf0mB3dZT77u+8O0QB89jFdnBkd5P1JgYPbPaia3gXXOVL2fq8VyU2gMMD7SaN7QukTB/pmXYvDA==",
      "requires": {
        "safer-buffer": ">= 2.1.2 < 3"
      }
    },
    "inherits": {
      "version": "2.0.3",
      "resolved": "https://registry.npmjs.org/inherits/-/inherits-2.0.3.tgz",
      "integrity": "sha1-Yzwsg+PaQqUC9SRmAiSA9CCCYd4="
    },
    "ipaddr.js": {
      "version": "1.9.1",
      "resolved": "https://registry.npmjs.org/ipaddr.js/-/ipaddr.js-1.9.1.tgz",
      "integrity": "sha512-
0KI/607xoxSToH7GjN1FfSbLoU0+btTicjsQSWQlh/hZykN8KpmMf7uYwPW3R+akZ6R/w18ZlXSHBYXiYUPO3g=="
    },
    "keygrip": {
      "version": "1.1.0",
      "resolved": "https://registry.npmjs.org/keygrip/-/keygrip-1.1.0.tgz",
      "integrity": "sha512-
iYSchDJ+liQ8iwbSI2QqsQOvqv58eJCEanyJPJi+Khyu8smkcKSFUCbPwzFcL7YVtZ6eONjqRX/38caJ7QjRAQ==",
      "requires": {
        "tsscmp": "1.0.6"
      }
    },
    "media-typer": {
      "version": "0.3.0",
      "resolved": "https://registry.npmjs.org/media-typer/-/media-typer-0.3.0.tgz",
      "integrity": "sha1-hxDXrwqmJvj/+hzgAWhUUmMlV0g="
    },
    "merge-descriptors": {
      "version": "1.0.1",
      "resolved": "https://registry.npmjs.org/merge-descriptors/-/merge-descriptors-
1.0.1.tgz",
      "integrity": "sha1-sAqqVW3YtEVoFQ7J0blT8/kMu2E="
    },
    "methods": {
      "version": "1.1.2",
      "resolved": "https://registry.npmjs.org/methods/-/methods-1.1.2.tgz",
      "integrity": "sha1-VSmk1nZUE07cxSZmVoNbD4Ua/O4="
    },
    "mime": {
      "version": "1.4.1",
      "resolved": "https://registry.npmjs.org/mime/-/mime-1.4.1.tgz",
      "integrity": "sha512-
KI1+qOZu5DcW6wayYHSzR/tXKCDC5Om4s1z2QJjDULzLcmf3DvzS7oluY4HCTrc+9FiKmWUgeNLg7W3uIQvxtQ=="
    },
    "mime-db": {
      "version": "1.44.0",
      "resolved": "https://registry.npmjs.org/mime-db/-/mime-db-1.44.0.tgz",
```

```
      "integrity": "sha512-
/NOTfLrsPBVeH7YtFPgsVWveuL+4SjjYxaQ1xtM1KMFj7HdxlBlxeyNLzhyJVx7r4rZGJAZ/6lkKCitSc/Nmpg=="
    },
    "mime-types": {
      "version": "2.1.27",
      "resolved": "https://registry.npmjs.org/mime-types/-/mime-types-2.1.27.tgz",
      "integrity": "sha512-
JIhqnCasI9yD+SsmkquHBxTSEuZdQX5BuQnS2Vc7puQQQ+8yiP5AY5uWhpdv4YL4VM5c6iliiYWPgJ/nJQLp7w==",
      "requires": {
        "mime-db": "1.44.0"
      }
    },
    "morgan": {
      "version": "1.9.1",
      "resolved": "https://registry.npmjs.org/morgan/-/morgan-1.9.1.tgz",
      "integrity": "sha512-
HQStPIV4y3afTiCYVxirakhlCfGkI161c76kKFca7Fk1JusM//Qeo1ej2XaMniiNeaZklMVrh3vTtIzpzwbpmA==",
      "requires": {
        "basic-auth": "~2.0.0",
        "debug": "2.6.9",
        "depd": "~1.1.2",
        "on-finished": "~2.3.0",
        "on-headers": "~1.0.1"
      }
    },
    "ms": {
      "version": "2.0.0",
      "resolved": "https://registry.npmjs.org/ms/-/ms-2.0.0.tgz",
      "integrity": "sha1-VgiurfwAvmwpAd9fmGF4jeDVl8g="
    },
    "negotiator": {
      "version": "0.6.2",
      "resolved": "https://registry.npmjs.org/negotiator/-/negotiator-0.6.2.tgz",
      "integrity": "sha512-
hZXc7K2e+PgeI1eDBe/10Ard4ekbfrrqG8Ep+8Jmf4JID2bNg7NvCPOZN+kfF574pFQI7mum2AUqDidoKqcTOw=="
    },
    "node-forge": {
      "version": "0.10.0",
      "resolved": "https://registry.npmjs.org/node-forge/-/node-forge-0.10.0.tgz",
      "integrity": "sha512-
PPmu8eEeG9saEUvI97fm4OYxXVB6bFvyNTyiUOBichBpFG8A1Ljw3bY62+5oOjDEMHRnd0Y7HQ+x7uzxOzC6JA=="
    },
    "on-finished": {
      "version": "2.3.0",
      "resolved": "https://registry.npmjs.org/on-finished/-/on-finished-2.3.0.tgz",
      "integrity": "sha1-IPEzZIGwg811M3mSoWlxqi2QaUc=",
      "requires": {
        "ee-first": "1.1.1"
      }
    },
    "on-headers": {
      "version": "1.0.2",
      "resolved": "https://registry.npmjs.org/on-headers/-/on-headers-1.0.2.tgz",
      "integrity": "sha512-
pZAE+FJLoyITytdqK0U5s+FIpjN0JP3OzFi/u8Rx+EV5/W+JTWGXG8xFzevE7AjBfDqHv/8vL8qQsIhHnqRkrA=="
    },
    "parseurl": {
      "version": "1.3.3",
      "resolved": "https://registry.npmjs.org/parseurl/-/parseurl-1.3.3.tgz",
      "integrity": "sha512-
CiyeOxFT/JZyN5m0z9PfXw4SCBJ6Sygz1Dpl0wqjlhDEGGBP1GnsUVEL0p63hoG1fcj3fHynXi9NYO4nWOL+qQ=="
```

```
    },
    "passport": {
      "version": "0.4.1",
      "resolved": "https://registry.npmjs.org/passport/-/passport-0.4.1.tgz",
      "integrity": "sha512-
IxXgZZs8d7uFSt3eqNjM9NQ3g3uQCW5avD8mRNoXV99Yig50vjuaez6dQK2qC0kVWPRTujxY0dWgGfT09adjYg==",
      "requires": {
        "passport-strategy": "1.x.x",
        "pause": "0.0.1"
      }
    },
    "passport-saml": {
      "version": "1.3.4",
      "resolved": "https://registry.npmjs.org/passport-saml/-/passport-saml-1.3.4.tgz",
      "integrity": "sha512-
FYkRAt3pbmUpm/FsH97U/xYDnd1+z3FuGYe3IUOQPRgyoDg2ub8LYofst4UU49u65aAWHHSfXk6S3CdSMllq3A==",
      "requires": {
        "debug": "^3.1.0",
        "passport-strategy": "*",
        "q": "^1.5.0",
        "xml-crypto": "^1.4.0",
        "xml-encryption": "^1.0.0",
        "xml2js": "0.4.x",
        "xmlbuilder": "^11.0.0",
        "xmldom": "0.1.x"
      },
      "dependencies": {
        "debug": {
          "version": "3.2.6",
          "resolved": "https://registry.npmjs.org/debug/-/debug-3.2.6.tgz",
          "integrity": "sha512-
mel+jf7nrtEl5Pn1Qx46zARXKDpBbvzezse7p7LqINmdoIk8PYP5SySaxEmYv6TZ0JyEKA1hsCId6DIhgITtWQ==",
          "requires": {
            "ms": "^2.1.1"
          }
        },
        "ms": {
          "version": "2.1.2",
          "resolved": "https://registry.npmjs.org/ms/-/ms-2.1.2.tgz",
          "integrity": "sha512-
sGkPx+VjMtmA6MX27oA4FBFELFCZZ4S4XqeGOXCv68tT+jb3vk/RyaKWP0PTKyWtmLSM0b+adUTEvbs1PEaH2w=="
        }
      }
    },
    "passport-strategy": {
      "version": "1.0.0",
      "resolved": "https://registry.npmjs.org/passport-strategy/-/passport-strategy-
1.0.0.tgz",
      "integrity": "sha1-tVOaqPwiWj0a0XlHbd8ja0QPUuQ="
    },
    "path-to-regexp": {
      "version": "0.1.7",
      "resolved": "https://registry.npmjs.org/path-to-regexp/-/path-to-regexp-0.1.7.tgz",
      "integrity": "sha1-32BBeABfUi8V60SQ5yR6G/qmf4w="
    },
    "pause": {
      "version": "0.0.1",
      "resolved": "https://registry.npmjs.org/pause/-/pause-0.0.1.tgz",
      "integrity": "sha1-HUCLP9t2kjuVQ9lvtMnf1TXZy10="
    },
    "proxy-addr": {
```

```
      "version": "2.0.6",
      "resolved": "https://registry.npmjs.org/proxy-addr/-/proxy-addr-2.0.6.tgz",
      "integrity": "sha512-
dh/frvCBVmSsDYzw6n926jv974gddhkFPfiN8hPOi30Wax25QZyZEGveluCgliBnqmuM+UJmBErbAUFIoDbjOw==",
      "requires": {
        "forwarded": "~0.1.2",
        "ipaddr.js": "1.9.1"
      }
    },
    "q": {
      "version": "1.5.1",
      "resolved": "https://registry.npmjs.org/q/-/q-1.5.1.tgz",
      "integrity": "sha1-fjL3W0E4EpHQRhHxvxQQmsAGUdc="
    },
    "qs": {
      "version": "6.5.2",
      "resolved": "https://registry.npmjs.org/qs/-/qs-6.5.2.tgz",
      "integrity": "sha512-
N5ZAX4/LxJmF+7wN74pUD6qAh9/wnvdQcjq9TZjevvXzSUo7bfmw91saqMjzGS2xq91/odN2dW/WOl7qQHNDGA=="
    },
    "range-parser": {
      "version": "1.2.1",
      "resolved": "https://registry.npmjs.org/range-parser/-/range-parser-1.2.1.tgz",
      "integrity": "sha512-
Hrgsx+orqoygnmhFbKaHE6c296J+HTAQXoxEF6gNupROmmGJRoyzfG3ccAveqCBrwr/2yxQ5BVd/GTl5agOwSg=="
    },
    "raw-body": {
      "version": "2.3.3",
      "resolved": "https://registry.npmjs.org/raw-body/-/raw-body-2.3.3.tgz",
      "integrity": "sha512-
9esiElv1BrZoI3rCDuOuKCBRbuApGGaDPQfjSflGxdy4oyzqghxu6klEkkVIvBje+FF0BX9coEv8KqW6X/7njw==",
      "requires": {
        "bytes": "3.0.0",
        "http-errors": "1.6.3",
        "iconv-lite": "0.4.23",
        "unpipe": "1.0.0"
      }
    },
    "safe-buffer": {
      "version": "5.1.2",
      "resolved": "https://registry.npmjs.org/safe-buffer/-/safe-buffer-5.1.2.tgz",
      "integrity": "sha512-
Gd2UZBJDkXlY7GbJxfsE8/nvKkUEU1G38c1siN6QP6a9PT9MmHB8GnpscSmMJSoF8LOIrt8ud/wPtojys4G6+g=="
    },
    "safer-buffer": {
      "version": "2.1.2",
      "resolved": "https://registry.npmjs.org/safer-buffer/-/safer-buffer-2.1.2.tgz",
      "integrity": "sha512-
YZo3K82SD7Riyi0E1EQPojLz7kpepnSQI9IyPbHHg1XXXevb5dJI7tpyN2ADxGcQbHG7vcyRHk0cbwqcQriUtg=="
    },
    "sax": {
      "version": "1.2.4",
      "resolved": "https://registry.npmjs.org/sax/-/sax-1.2.4.tgz",
      "integrity": "sha512-
NqVDv9TpANUjFm0N8uM5GxL36UgKi9/atZw+x7YFnQ8ckwFGKrl4xX4yWtrey3UJm5nP1kUbnYgLopqWNSRhWw=="
    },
    "send": {
      "version": "0.16.2",
      "resolved": "https://registry.npmjs.org/send/-/send-0.16.2.tgz",
      "integrity": "sha512-
E64YFPUssFHEFBvpbbjr44NCLtI1AohxQ8ZSiJjQLskAdKuriYEP6VyGEsRDH8ScozGpkaX1BGvhanqCwkcEZw==",
```

```
      "requires": {
        "debug": "2.6.9",
        "depd": "~1.1.2",
        "destroy": "~1.0.4",
        "encodeurl": "~1.0.2",
        "escape-html": "~1.0.3",
        "etag": "~1.8.1",
        "fresh": "0.5.2",
        "http-errors": "~1.6.2",
        "mime": "1.4.1",
        "ms": "2.0.0",
        "on-finished": "~2.3.0",
        "range-parser": "~1.2.0",
        "statuses": "~1.4.0"
      }
    },
    "serve-static": {
      "version": "1.13.2",
      "resolved": "https://registry.npmjs.org/serve-static/-/serve-static-1.13.2.tgz",
      "integrity": "sha512-
p/tdJrO4U387R9oMjb1oj7qSMaMfmOyd4j9hOFoxZe2baQszgHcSWjuya/CiT5kgZZKRudHNOA0pYXOl8rQ5nw==",
      "requires": {
        "encodeurl": "~1.0.2",
        "escape-html": "~1.0.3",
        "parseurl": "~1.3.2",
        "send": "0.16.2"
      }
    },
    "setprototypeof": {
      "version": "1.1.0",
      "resolved": "https://registry.npmjs.org/setprototypeof/-/setprototypeof-1.1.0.tgz",
      "integrity": "sha512-
BvE/TwpZX4FXExxOxZyRGQQv651MSwmWKZGqvmPcRIjDqWub67kTKuIMx43cZZrS/cBBzwBcNDWoFxt2XEFIpQ=="
    },
    "statuses": {
      "version": "1.4.0",
      "resolved": "https://registry.npmjs.org/statuses/-/statuses-1.4.0.tgz",
      "integrity": "sha512-
zhSCtt8v2NDrRlPQpCNtw/heZLtfUDqxBM1udqikb/Hbk52LK4nQSwr10u77iopCW5LsyHpuXS0GnEc48mLeew=="
    },
    "tsscmp": {
      "version": "1.0.6",
      "resolved": "https://registry.npmjs.org/tsscmp/-/tsscmp-1.0.6.tgz",
      "integrity": "sha512-
LxhtAkPDTkVCMQjt2h6eBVY28KCjikZqZfMcC15YBeNjkgUpdCfBu5HoiOTDu86v6smE8yOjyEktJ8hlbANHQA=="
    },
    "type-is": {
      "version": "1.6.18",
      "resolved": "https://registry.npmjs.org/type-is/-/type-is-1.6.18.tgz",
      "integrity": "sha512-
TkRKr9sUTxEH8MdfuCSP7VizJyzRNMjj2J2do2Jr3Kym598JVdEksuzPQCnlFPW4ky9Q+iA+ma9BGm06XQBy8g==",
      "requires": {
        "media-typer": "0.3.0",
        "mime-types": "~2.1.24"
      }
    },
    "unpipe": {
      "version": "1.0.0",
      "resolved": "https://registry.npmjs.org/unpipe/-/unpipe-1.0.0.tgz",
      "integrity": "sha1-sr9O6FFKrmFltIF4KdIbLvSZBOw="
    },
```

```
    "utils-merge": {
      "version": "1.0.1",
      "resolved": "https://registry.npmjs.org/utils-merge/-/utils-merge-1.0.1.tgz",
      "integrity": "sha1-n5VxD1CiZ5R7LMwSR0HBAoQn5xM="
    },
    "vary": {
      "version": "1.1.2",
      "resolved": "https://registry.npmjs.org/vary/-/vary-1.1.2.tgz",
      "integrity": "sha1-IpnwLG3tMNSllhsLn3RSShj2NPw="
    },
    "xml-crypto": {
      "version": "1.5.3",
      "resolved": "https://registry.npmjs.org/xml-crypto/-/xml-crypto-1.5.3.tgz",
      "integrity": "sha512-
uHkmpUtX15xExe5iimPmakAZN+6CqIvjmaJTy4FwqGzaTjrKRBNeqMh8zGEzVNgW0dk6beFYpyQSgqV/J6C5xA==",
      "requires": {
        "xmldom": "0.1.27",
        "xpath": "0.0.27"
      },
      "dependencies": {
        "xmldom": {
          "version": "0.1.27",
          "resolved": "https://registry.npmjs.org/xmldom/-/xmldom-0.1.27.tgz",
          "integrity": "sha1-1QH5ezvbQDr4757MIFcxh6rawOk="
        }
      }
    },
    "xml-encryption": {
      "version": "1.2.1",
      "resolved": "https://registry.npmjs.org/xml-encryption/-/xml-encryption-1.2.1.tgz",
      "integrity": "sha512-
hn5w3l5p2+nGjlmM0CAhMChDzVGhW+M37jH35Z+GJIipXbn9PUlAIRZ6I5Wm7ynlqZjFrMAr83d/CIp9VZJMTA==",
      "requires": {
        "escape-html": "^1.0.3",
        "node-forge": "^0.10.0",
        "xmldom": "~0.1.15",
        "xpath": "0.0.27"
      }
    },
    "xml2js": {
      "version": "0.4.23",
      "resolved": "https://registry.npmjs.org/xml2js/-/xml2js-0.4.23.tgz",
      "integrity": "sha512-
ySPiMjM0+pLDftHgXY4By0uswI3SPKLDw/i3UXbnO8M/p28zqexCUoPmQFrYD+/1BzhGJSs2i1ERWKJAtiLrug==",
      "requires": {
        "sax": ">=0.6.0",
        "xmlbuilder": "~11.0.0"
      }
    },
    "xmlbuilder": {
      "version": "11.0.1",
      "resolved": "https://registry.npmjs.org/xmlbuilder/-/xmlbuilder-11.0.1.tgz",
      "integrity": "sha512-
fDlsI/kFEx7gLvbecc0/ohLG50fugQp8ryHzMTuW9vSa1GJ0XYWKnhsUx7oie3G98+r56aTQIUB4kht42R3JvA=="
    },
    "xmldom": {
      "version": "0.1.31",
      "resolved": "https://registry.npmjs.org/xmldom/-/xmldom-0.1.31.tgz",
      "integrity": "sha512-
yS2uJflVQs6n+CyjHoaBmVSqIDevTAWrzMmjG1Gc7h1qQ7uVozNhEPJAwZXWyGQ/Gafo3fCwrcaokezLPupVyQ=="
    },
```

```
    "xpath": {
      "version": "0.0.27",
      "resolved": "https://registry.npmjs.org/xpath/-/xpath-0.0.27.tgz",
      "integrity": "sha512-
fg03WRxtkCV6ohClePNAECYsmpKKTv5L8y/X3Dn1hQrec3POx2jHZ/0P2qQ6HvsrU1BmeqXcof3NGGueG6LxwQ=="
    }
  }
}
```