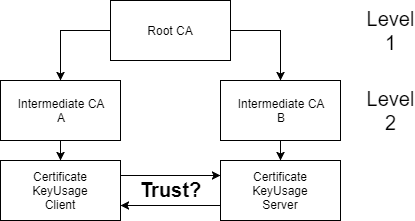
# Як увімкнути взаємний TLS у програмі Spring Boot

Взаємний TLS – це ще один крок під час SSL-рукостискання поверх одностороннього TLS, у якому клієнт вірить, що сервер має дійсний сертифікат. Таким же чином сервер також запитуватиме та перевірятиме сертифікат клієнта, тому обидві сторони довіряють одна одній і з’єднання взаємно перевіряється. Важливою частиною цих переговорів є орган, який підписав сертифікати. Якщо орган публічно довіряє в Інтернеті, то сторони також довіряють сертифікату, підписаному цим органом. Якщо повноваження не є загальнодоступними, сторона, яка їх отримує, має довіряти всьому ланцюжку сертифікатів від кінцевої сутності до кореневого ЦС. Це включає в себе всі проміжні сертифікати, які надходять між кореневим ЦС і кінцевою сутністю.



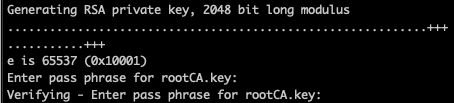
**Примітка.** У цієї інструкції ми використовуватимемо самопідписані сертифікати, щоб захистити нашу програму mTLS. Це означає, що ми самі, у цьому випадку, будемо кореневим центром сертифікації, і ми можемо підписувати сертифікати клієнта та сервера, щоб ми могли використовувати їх для тестування нашої програми. Тепер почнемо з генерації сертифікатів!

**Створення кореневих сертифікатів**

Якщо ***openssl*** CLI вже встановлено, виконайте наведені нижче дії, щоб створити самопідписаний кореневий сертифікат. Ми будемо використовувати його для підписання сертифікатів клієнта та сервера далі в історії.

openssl genrsa -des3 -out rootCA.key 2048

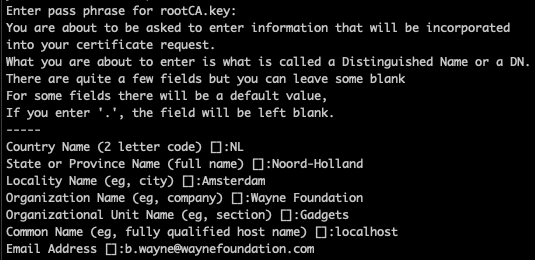
Ця команда генерує закритий ключ для кореневого сертифіката. Буде запропоновано ввести пароль, який буде паролем ключа. Це дуже важливо, і його потрібно надійно зберігати, оскільки його можна використовувати для підписання сертифікатів від імені іншої сутності, якщо його розкриють. Результат буде приблизно таким:



Надішліть запит на сертифікат *openssl* за допомогою ключа, згенерованого на попередньому кроці

openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 365 -out rootCA.pem

***days*** вказує дату закінчення терміну дії сертифіката відтепер. В нашому випадку це 10 років. Ми називаємо файл сертифіката***rootCA.pem.*** Тут вам буде запропоновано ввести певну інформацію в сертифікат для ідентифікації.



Щиро вітаю! Тепер ви є центром сертифікації та можете підписувати сертифікати для інших організацій. Суб’єкт, який запитує сертифікат від центру сертифікації, повинен надати ЦС CSR, який містить інформацію про запит. Відповідно до визначення:

*У системах інфраструктури відкритих ключів запит на підписання сертифіката — це повідомлення, надіслане заявником до органу реєстрації інфраструктури відкритих ключів для того, щоб подати заявку на сертифікат цифрової ідентифікації.*

Продовжуючи наш процес, давайте спочатку створимо сертифікати сервера.

**Підписання сертифіката сервера**

Давайте створимо закритий ключ, а потім CSR для нашого сертифіката сервера.

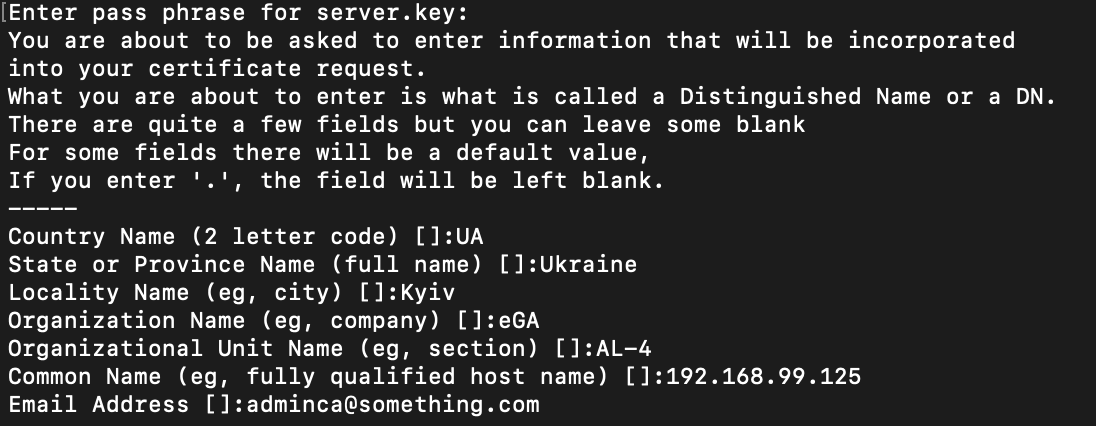
openssl genrsa -des3 -out server.key 2048

Тепер надішліть запит CSR із ключем у якості вхідного ключа:

openssl req -new -sha256 -key server.key -out server.csr

Він запитає ту саму інформацію, яка потрібна для кореневого ЦС.

**ВАЖЛИВО: Поле CN має містити повне ім’я або IP хоста, на якому буде доступний сервер, у цьому випадку 192.168.99.125**

****

CSR має бути надано організацією, яка запитує сертифікат, а потім передано до кореневого ЦС. У цьому випадку, оскільки ми також виконуємо роль кореневого ЦС, давайте підпишемо сертифікат сервера за допомогою даного CSR.

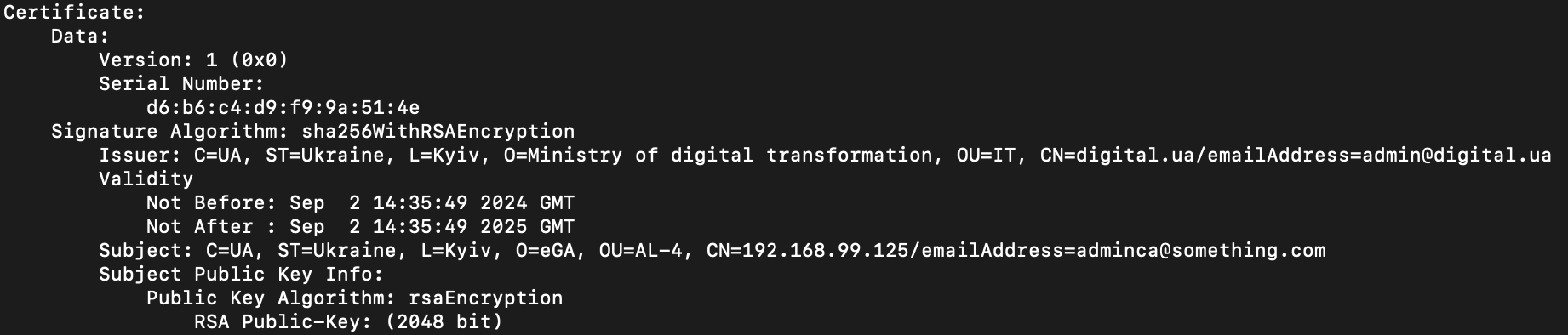
openssl x509 -req -in server.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out server.pem -days 365 -sha256

Щоб продовжити та підписати новий сертифікат, вам буде запропоновано ввести пароль ключа кореневого ЦС. Це має створити сертифікат сервера з назвою ***server.pem***, який разом із ***server.key*** використовуватиметься для налаштування Tomcat для ввімкнення SSL.

Перевіряемо, що усі поля відображаються корректно за допомогою наступної команди:

openssl x509 -in server.pem -noout -text

Ви повинні побачити щось подібне:



Зверніть увагу, що поле CN має вигляд як [CN=192.168.125/emailAddress=adminca@something.com](mailto:CN=192.168.125/emailAddress=adminca@something.com)

Таке може зустрічатися в деяких реалізаціях openssl, та може привести що деякі клієнти не зможуть працювати з полем CN. Якщо у вас відобразилося щось подібне, то рішень два:

1. При генеруванні CSR файла просто не вказувати е-майл адресу.
2. При формуванні PEM файла використовувати додаткове поле subjectAltName

Для цього треба створити конфігураційний файл req.conf з наступним вмістом:

[ v3\_ext ]

subjectAltName = IP:192.168.99.125

(тут повинен бути ваш хост на якому працює сервіс, або його IP адреса)

Або виконати команду:

echo "[ v3\_ext ]

subjectAltName = IP:192.168.99.125" | tee req.conf >/dev/null

це створить конфіг файл з імʼям req.conf та далі формуємо РЕМ файл.

openssl x509 -req -in server.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out server.pem -days 3650 -sha256 -extensions v3\_ext -extfile req.conf

**Підписання сертифіката клієнта**

Як згадувалося вище, взаємний TLS базується на автентифікації обох сторін одна одної. Якби це був односторонній TLS, нам би не потрібен був сертифікат клієнта, оскільки сервер не запитував би його. Однак у цьому випадку ми хочемо, щоб клієнт надав свій сертифікат, а сервер автентифікував його. Давайте створимо клієнтські сертифікати, щоб ми могли використовувати їх для виклику API.

openssl genrsa -des3 -out client.key 2048

Потім таким же чином створіть CSR для клієнта

openssl req -new -sha256 -key client.key -out client.csr

Потім таким же чином підписуємо сертифікат клієнта

openssl x509 -req -in client.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out client.pem -days 365 -sha256

Чудово! Тепер у нас є і клієнтські, і серверні сертифікати. Давайте використаємо ці сертифікати для захисту нашого додатка Spring.

**Увімкнення SSL у прикладі програми Spring**

Тепер запустіть програму та натисніть  
GET [*http://localhost:8080/api/v1/persons*](http://localhost:8080/api/v1/persons/)*/list*

Ми отримали результат. Але наше підключення до сервера недостатньо безпечне. Давайте спочатку захистимо наш сервер за допомогою сертифіката сервера, щоб встановити односторонній TLS, де клієнт зможе перевірити, чи ми маємо дійсний сертифікат.

Spring надає набір конфігурацій, за допомогою яких можна ввести сертифікат. Він очікує сертифікат і відповідний закритий ключ у сховищі **ключів** у форматі JKS або PKCS#12. Хоча ***keytool*** , який є інструментом для створення сховищ ключів, змінив тип сховища за замовчуванням на PKCS12 із JDK 9+, і, очевидно, JKS більше не підтримуватиметься. Дотримуючись того самого шляху, ми також об’єднаємо наш сертифікат сервера та його ключ у сховище ключів PKCS12.

Перейдіть до каталогу, де є сертифікати, і виконайте таку команду, щоб створити сховище ключів із сертифіката сервера та його закритого ключа.

openssl pkcs12 -export -in server.pem -out keystore.p12 -name server -nodes -inkey server.key

Це експортує сертифікат і закритий ключ до сховища ключів у форматі PKCS, яке ми можемо використовувати для налаштування програми Spring. За замовчуванням приватний ключ імпортується в сховище ключів у зашифрованому вигляді, і в цьому випадку нам потрібно буде вказати Spring розшифрувати його за допомогою свого пароля. Для цієї історії ми збираємося розшифрувати закритий ключ перед його імпортом. ***Nodes*** подбають про це у наведеній вище команді, яка запропонує ввести пароль ключа. CLI також запитає пароль для сховища ключів, який нам потрібно буде надати Spring пізніше. Тож запам’ятайте це!

Тепер давайте помістимо згенерований файл ***keystore.p12*** у каталог ***src/main/resources*** нашої програми (хоча це, ймовірно, має бути надано через секрет або якусь зовнішню конфігурацію).

Додайте наступний фрагмент у файл ./config/extended-config.proprties і перезапустіть програму.

server.ssl.enabled=true   
 server.ssl.key-store="classpath:keystore.p12"   
 server.ssl.key-store-password=yourpass   
 server.ssl.key-store-type=PKCS12

**Примітка.** Якщо ключ було імпортовано до сховища ключів у зашифрованому вигляді, тоді для розшифровки закритого ключа слід надати **ключ-пароль .**

**key-store** — це шлях до файлу PKCS12, який у цьому випадку є ресурсом classpath. **key-store-password** буде паролем, який ви ввели під час створення сховища ключів.

Перше, що слід помітити після перезапуску програми, це такий журнал:

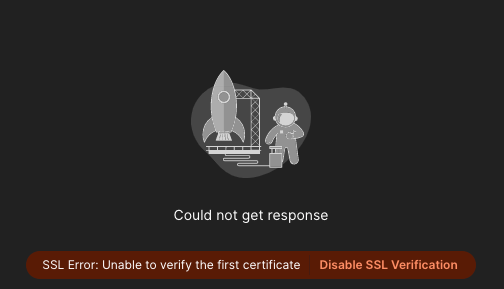
Tomcat started on port(s): 8080 **(https)** with context path ‘’

Це означає, що SSL увімкнено на нашому вбудованому сервері Tomcat. Давайте знову запустимо наш додаток:

GET *<http://localhost:8080/api/v1/persons/list>*

Якщо ви використовуєте такий клієнт API, як Postman, ви отримаєте відповідь на зразок Bad Request. This combination of host and port requires TLS.того, що нам потрібно використовувати ***https*** замість **http.**

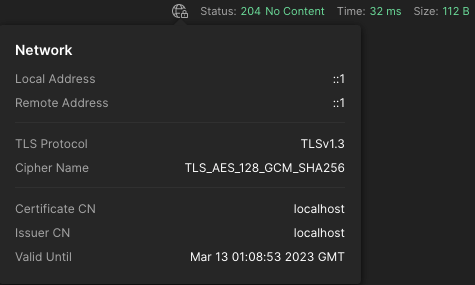
GET [***https****://192.168.99.125:8080/api/v1/persons/list*](https://192.168.99.125:8080/api/v1/persons/list)



Здається, Postman не може перевірити сертифікат нашого сервера під час рукостискання, що має сенс, оскільки Postman не знає, що ми є центром сертифікації. Він уже довіряє іншим публічним ЦС в Інтернеті, але якимось чином нам потрібно змусити Postman довіряти нам як кореневому ЦС.

Перейдіть до налаштувань Postman, а потім вкладку сертифікатів, де ви можете додати сертифікати до своїх запитів на основі імені хоста та номера порту. Угорі є розділ під назвою ***«CA certificates»*** , у якому ви можете вручну налаштувати довіру Центра Сертіфікації (СА). Це один із кроків у налаштуванні TLS, одностороннього чи взаємного.

Увімкніть перемикач і виберіть файл ***rootCA.pem***, щоб Postman міг довіряти CA (нам). Ще раз запустіть запит, і з’єднання має бути перевірено зараз.

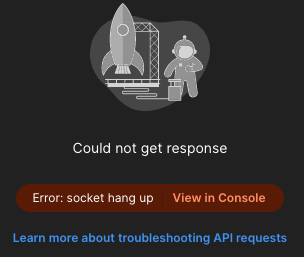


**Увімкнення взаємного TLS у прикладі Spring Boot Application**

Поки що наш сервіс має одностороннє налаштування TLS, і клієнти, які довіряють кореневому ЦС, можуть надсилати запити. Однак, щоб отримати взаємну автентифікацію між клієнтом і сервером, нам потрібно дещо змінити конфігурацію, щоб наш сервер також запитував сертифікат клієнта. Перейдіть до файлу налаштувань (замовчки) ***config/extend-config.properties***і додайте такі властивості:

server.ssl.client-auth=NEED

Цей ключ дозволяє нам налаштувати, чи потрібна нам автентифікація клієнта (така ж взаємна автентифікація) чи ні. Це може бути *NEED* , WANT та NONE. NEED вказує, що сервер повинен перевірити сертифікат клієнта, тоді як WANT також запитує сертифікат клієнта з тією ключовою різницею, що він підтримує з’єднання, якщо не надається автентифікація. Використовуючи NONE, сертифікат клієнта ніколи не запитується. Перезапустіть програму та знову спробуйте кінцеву точку.



Ви побачите, що рукостискання з сервером не вдалося. У цьому випадку сервер не зміг перевірити клієнта, оскільки клієнт не надавав жодних сертифікатів, і з’єднання було розірвано під час рукостискання. Якщо ви ввімкнете журналювання налагодження в програмі Spring, це має відображатися як виняток із повідомленням «**пустий ланцюжок сертифікатів клієнта**». Що нам потрібно зробити, щоб це розібратися, так це надіслати сертифікат клієнта разом із запитом під час виклику API.

Знову перейдіть до вкладки «Сертифікати» в налаштуваннях Postman, натисніть « ***Додати сертифікат»*** і додайте такий запис:

**- host:**192.168.99.125  
**- port:** 8080  
- **CRT File:**/path/to/client.pem  
**- KEY File:**/path/to/client.key

Повторно надішліть запит і Postman скаржиться на ту саму помилку. Хоча клієнт і сервер надають один одному свої сертифікати, з’єднання все одно не вдається встановити. Якщо ви заглянете в журнал свого сервера, ви зможете відстежити, що **SslHandShakeException** виникає щоразу, коли клієнт викликає кінцеву точку, і десь у цих рядках є повідомлення ***«unknown certificate»*** . Цей виняток виникає, оскільки наш сервер не може перевірити сертифікат клієнта, оскільки він не довіряє кореневому **CA** сертифіката клієнта. Він знає, що це дійсний сертифікат, але не знає, де його було підписано, і тому відхиляє запит. (так само, як Postman).

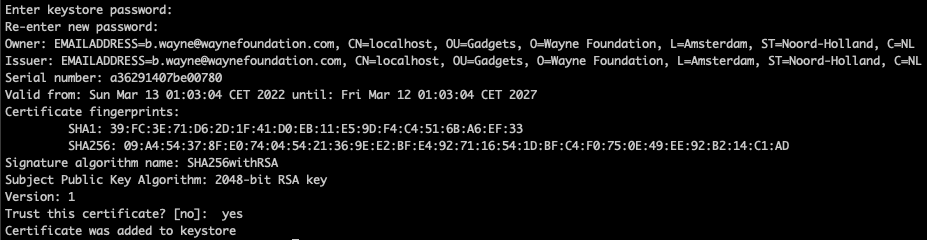
Виявляється, для ввімкнення автентифікації сертифіката клієнта потрібно довіряти кореневому **CA** іншої сторони, якщо тільки він не довірений через Інтернет. Java виконує цей механізм довіри за допомогою пакетів об’єктів під назвою **Trust Store** . Подібно до сховищ ключів, які зберігають сертифікат сервера, довірчі сховища зберігають сертифікати, яким програма повинна довіряти, щоб перевіряти та автентифікувати клієнтів. За замовчуванням Java використовує власне довірче сховище, яке постачається з усіма JRE, однак Spring забезпечує достатню гнучкість, щоб ми могли спрямовувати програму до довіреного сховища так само, як це було зі сховищем ключів.

Давайте створимо надійне сховище, що містить кореневий ЦС, який підписав сертифікат клієнта. Перейдіть до місця, де знаходяться сертифікати для клієнта, і виконайте таку команду:

keytool -import -file rootCA.pem -alias rootCA -keystore truststore.p12

**Примітка** . Для роботи ***keytool*** потрібно встановити Java.

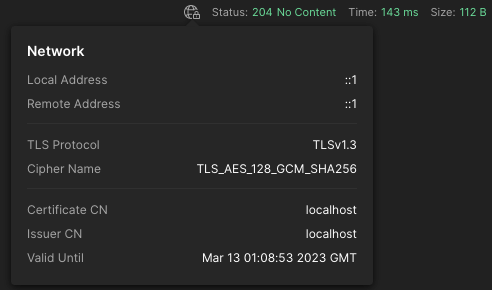
Ця команда використовує згаданий вище CLI ***keytool*** для створення надійного сховища у форматі PKCS12. Він запитує пароль, який нам знадобиться під час налаштування програми. CLI показує інформацію про сертифікат і запитає, чи слід довіряти сертифікату. Запропонуйте **«yes»** і створіть довірче сховище.



Тепер давайте помістимо файл ***truststore.p12*** у папку **src/main/resources** і вкажемо на файл за допомогою таких властивостей:

server.ssl.trust-store=classpath:keystore/truststore.p12   
server.ssl.trust-store-password=yourpass  
server.ssl.trust-store-type=PKCS12

Перезапустіть програму та повторно надішліть запит за допомогою Postman.



Тепер ми бачимо, що сервер також прийняв сертифікат клієнта як **дійсний,** оскільки він довіряє органу, який його підписав.

Чудово! Хоча наш зразок програми лише для тестування, вміст довіреного сховища буде зовсім іншим у реальних ситуаціях. Сертифікати клієнта та сервера для нашого зразка програми були підписані безпосередньо кореневим ЦС, тому не було потреби додавати проміжні сертифікати. У цьому випадку весь ланцюжок сертифікатів (кореневий + проміжний + кінцевий об'єкт) слід імпортувати в довірче сховище, щоб інша сторона могла відстежити сертифікат до кореневого центру, який його підписав.

**Взаємна TLS автентифікація з Трембітою**

Наступний етап, це налаштування взаємодії нашого сервіса та Трембіти.

Для цього у вас повинно бути налаштован ШБО.

Додано хоча б 2 клієнта

Також треба додати наш сервіс на ШБО (докладніше як це робити можна прочитати в документаціі на Трембіту)

Приклад зареєстрованого сервіса показан на рис. 5

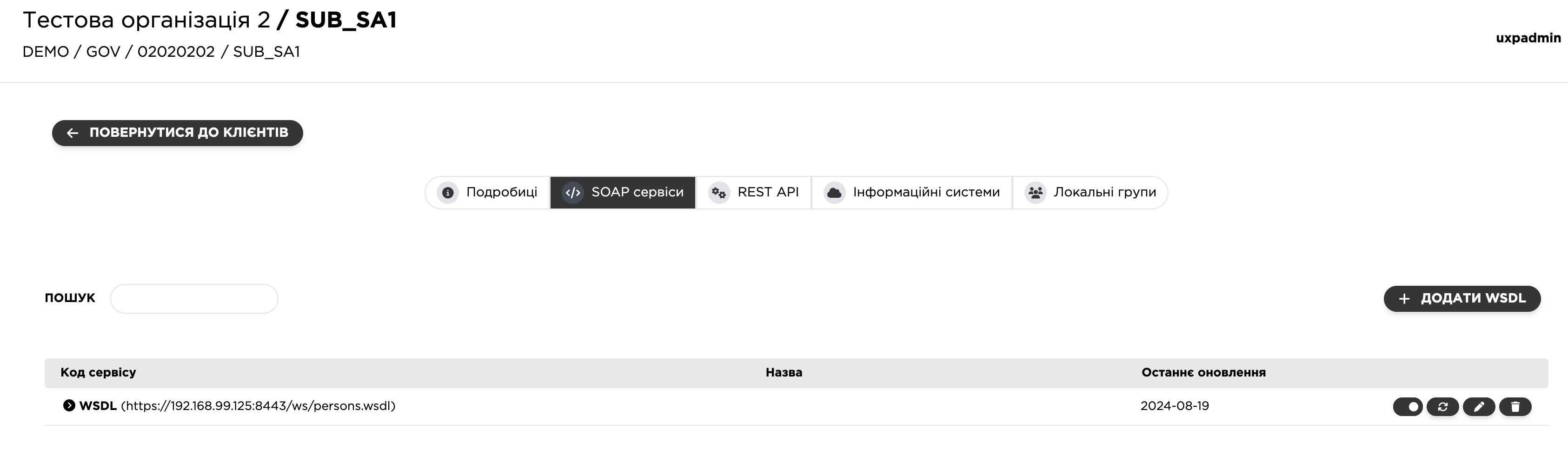


Рис.5

Далі треба зареєструвати на ШБО сертифікати сервера (server.pem та client.pem) сгенерованих раніше.

Це робиться на вкладці Інформаційни системи як показано на рис.6

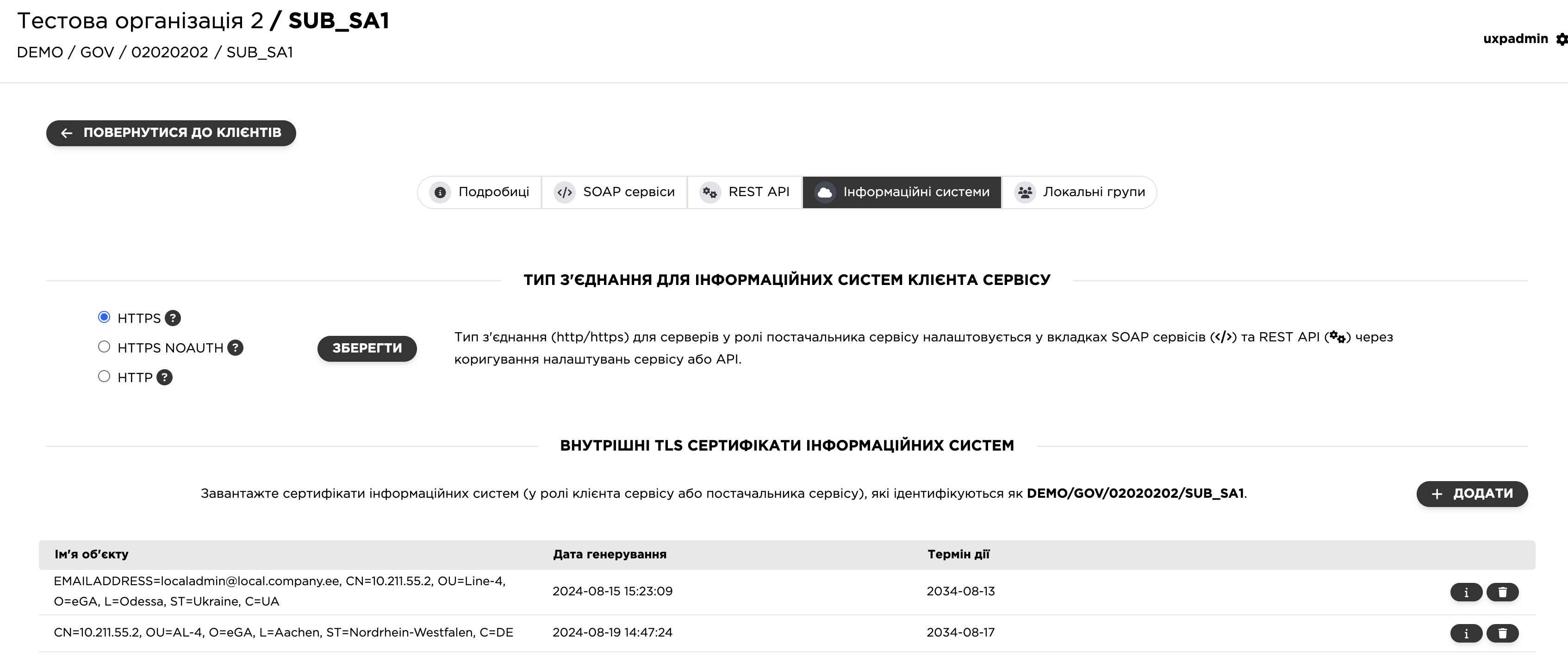


Рис.6

Тиснемо Додати та вказуємо шлях до сертіфікатов server.pem та client.pem

Тем самим ми вказали що для Клієнта з ЄДРПОУ 02020202 підсистеми SUB\_SA1 дозволено працювати з сервісом та клієнтом.

Якщо ви використовуєте інші підсистеми, то для них також треба тим самим чином додати сертифікати.

Далі завантажуємо на диск внутрішний сертифікат ШБО рис.7. Замовчки у нього буде імʼя internal\_auth\_cert.pem

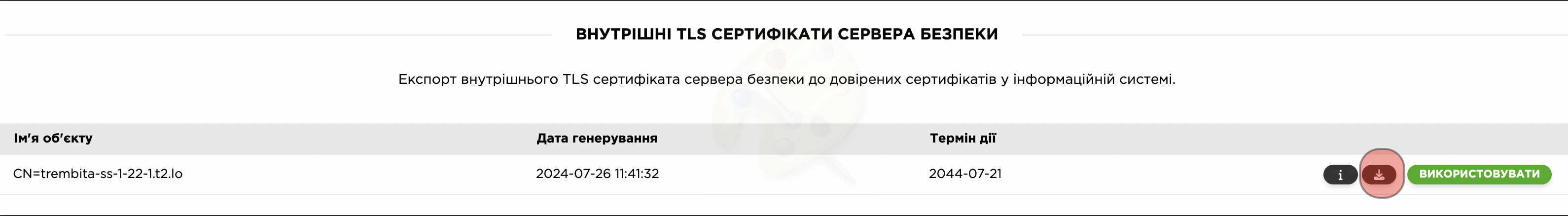
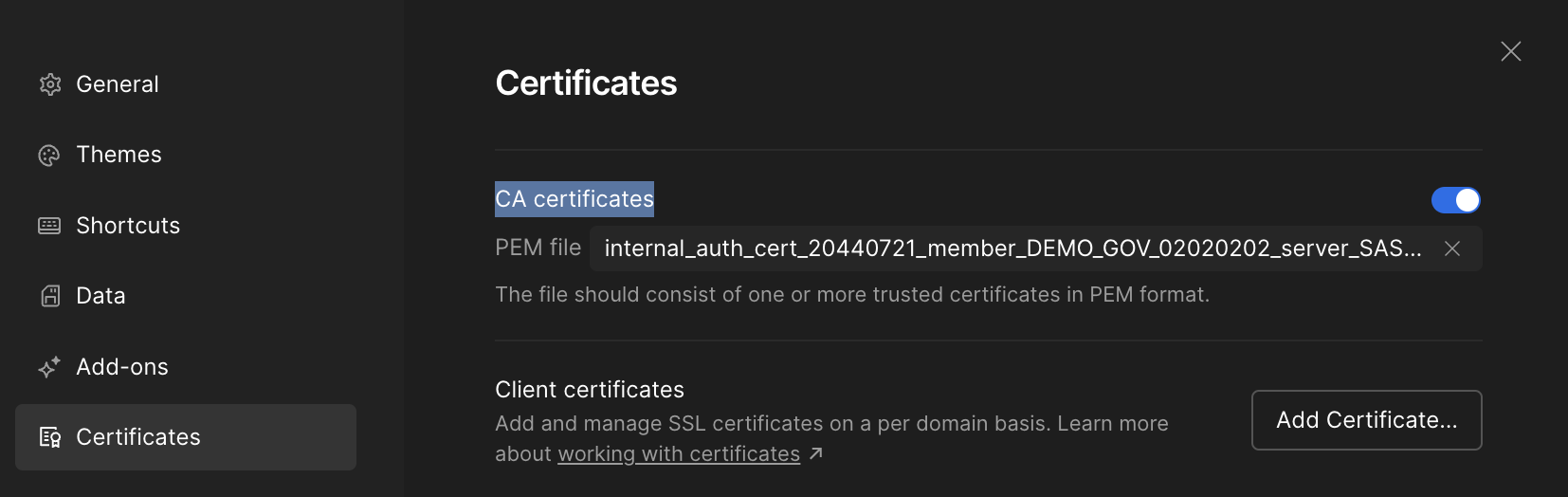


Рис. 7

Далі відкриваємо Postman -> Налаштування -> Сертифікати та додаємо в поле CA certificates наш internal\_auth\_cert.pem як показано на рис.8



Додаємо Клієнтський сертифікат (тиснемо Add Сertificate…) та вказуємо

**HOST**: 192.168.99.120 (це наш сервер ШБО)

**CRT file**: /Volumes/DATA/Exchange/keys/client.pem

**KEY file**: /Volumes/DATA/Exchange/keys/client.key

**Passphrase:** \*\*\*\*\*\*\*\*\* (пароль до client.key)

\*/Volumes/DATA/Exchange/keys/ - ваш шлях до сертифіката та таємного ключа.