НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО ТЕХНІЧНИЙ ІНСТИТУТ
КАФЕДРА МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ ТА АНАЛІЗУ ДАНИХ

Лабораторна робота №4

З дисципліни «Методи реалізації криптографічних механізмів»

Студента групи ФІ-21мн
Прохоренко О.С.

Викладач:
Селюх П.В.

Київ-2023

ХІД РОБОТИ

1. Завдання

Розробка реалізацій IT-систем у відповідності до стандартних вимог Crypto API. Підгрупа 3А. Реалізація Web-сервісу електронного цифрового підпису.

2. Стандарти та конвенції Crypto APIs

https://developers.cryptoapis.io/technical-documentation/api/general-information/standards-and-conventions

1. Структура та стандарти найменування URI:
   - Переконайтеся, що всі URI у вашому API написані маленькими літерами та використовують спінальний випадок (розділені дефісом) для параметрів шляху.
   - URI має включати в себе версію API (/v2/).
   - Максимальна довжина URI – це 2000 символів

URI Structure

The Crypto APIs and endpoint URIs follow the RFC 3986 specification and are divided into three main parts, applicable to all Crypto APIs products (Blockchain Events, Blockchain Data, etc.):

1. The domain and subdomain in the URI always display as: **rest.cryptoapis.io**
2. Next is displayed the version of the product, e.g. currently it is: **/v2/**
3. Next is displayed the product name, where words are separated with spinal-case: **/blockchain-data/**

Example:

rest.cryptoapis.io/v2/blockchain-data/..

URI Maximum Length

The entire URI cannot exceed 2000 characters in length including any punctuation marks, e.g. commas, hyphens, question marks, pluses or forward slashes.

2. Тільки HTTPS:
   - Це важливо для безпеки, особливо при роботі з криптографічними операціями.

3. Стандарти запитів та відповідей:
   - Запити та відповіді повинні використовувати camelCase для публічних атрибутів.
   - Включіть apiVersion у ваші запити та відповіді.
   - Використовуйте JSON для тіл запитів та відповідей.
   - Якщо ваш API повертає великі набори даних, реалізуйте пагінацію для обмеження потоку даних.

Basic API Request

Main prerequisites for requests to the Crypto APIs include:

- requests are sent through HTTPS only to the domain **rest.cryptoapis.io**;
- headers must by default incorporate the JSON content type **application/json**;
- all custom for Crypto APIs attributes are indicated in the beginning with an `x-`, e.g. `x-attribute`;
- request public attributes must be all camelCase, e.g. `"apiVersion": 2`, `"attributeName": "attributeValue"`;
- we enable CORS (Cross-Origin Resource Sharing), for which the API responds with an `Access-Control-Allow-Origin:` header. **Nevertheless**, your users **shouldn't make** direct API requests from a web application that you are building, as our CORS policy may change at some point without warning and any such requests could be then rejected;
- no random unspecified keys can be added to the Crypto APIs URI, e.g. `?randomKey=randomUnspecifiedValue`, as that will result in a 400 error;
- all requests to the Crypto APIs **must** be authenticated with an API key. Clients can generate API keys through their dedicated Dashboards only after product subscription. Multiple API keys can be generated for a single user. To see more information on Authentication, please see the respective article;
- API keys must be kept secure and private by the users who own them. API keys **must not** be uploaded to a frontend of a mobile or web application or in any open source code, as API keys represent access to the user's account and the data inside.

4. Обробка помилок:
   - Узгодьте ваші відповіді на помилки з HTTP-кодами статусів (4xx для помилок клієнта, 5xx для помилок сервера).
   - Структуруйте відповіді на помилки, щоб включати apiVersion, requestId (унікальний ідентифікатор запиту) та об'єкт помилки, що містить код, повідомлення та необов'язкові деталі.

Error and HTTP Code Standards

The full list of Errors you can see on our respective page.

The errors we use follow the HTTP Error Codes Standard.

HTTP Status Code	Error it represents
3xx	Redirection Error
4xx	Client Error
5xx	Server Error

The structure of the error **always** returns the following values, as listed and described in the example:

Example:

```
{
    "apiVersion": "", // The current API Version.
    "requestId": "", // Each request has a unique ID, for which the Support team
    "context": "", // Optional: In case you send this value in your request.
    "error": {
        "code": "", // Error Code, please check our [Errors page](https://develop
        "message": "", // Is the human readable error message.
        "details": "" // Optional: Some errors may need more details.
    }
}
```

5. Формати дати та часу:
    - Якщо вашому API потрібні параметри дати та часу, використовуйте час UNIX Epoch (формат цілого числа).

Date and Time Formats

For all required endpoints that use a time parameter, the accepted value is `timestamp` which is always UTC-based by definition. The human date which corresponds to the `timestamp` is always converted by the system to UNIX Epoch time and returned as an integer.

Example:

Human time: 09:12:41 AM 28th January 2021

Which corresponds to:

UNIX Epoch time: 1611825161

6. Параметр контексту:
    - Розгляньте можливість додавання необов'язкового параметра context, щоб допомогти користувачам відстежувати відповіді на масові запити.

3. Результати

```
❯ python3 main.py
INFO:     Started server process [1319052]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:     127.0.0.1:54382 - "POST /v1/generate-keys/ HTTP/1.1" 200 OK
INFO:     127.0.0.1:43730 - "POST /v1/sign-message/ HTTP/1.1" 200 OK
INFO:     127.0.0.1:46296 - "POST /v1/verify-signature/ HTTP/1.1" 200 OK
INFO:     127.0.0.1:53586 - "POST /v1/verify-signature/ HTTP/1.1" 400 Bad Request
```

```
❯ curl -X 'POST' \
  'http://127.0.0.1:8000/v1/generate-keys/' \
  -H 'accept: application/json' \
  -d ''

{"apiVersion":"1.0","requestId":"374b2aae-569e-4973-8956-782b38e5373b","context":"Private and
 Public keys generation","data":{"item":{"privateKeyPath":"private_key.pem","publicKeyPath":"
public_key.pem"}}}%
❯ curl -X 'POST' \
  'http://127.0.0.1:8000/v1/sign-message/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{"message": "This is a secret message"}'

{"apiVersion":"1.0","requestId":"662ff60d-31ea-40fc-8892-91fb958b94b8","context":"Signing of
message by private key","data":{"item":{"signature":"51a652912eb9ff4c39997dad3412ce8cc80f2652
d26267e23e7fa6f51e43c6255bdeb1dc7c4138f1702dad5b3b4f2fc8bdd86cf32d37d748082c6b3016b5d3154b3e3
6c140b0b239c25ccd188fc6f1c80d8de6729b72a464e78bf0776a270296232a91d4f6e14be885064d03a21400812a
eff4449375ccbdf3da8412f458e38ce921b873232f6824086dd07e9f7d5a9190f5eada77d46dc9750b7c1f56a54df
50c7d07e64df70eec96461bcfead1177ed6bdcf56dfc4a44cd347b4084b75d5aab3e8152b86e8a741a115a07ffe74
19efab95ff36b30354e5d2f82b3e5427ccb6116f68293a455e8e99a56f244f7e27a2e5221fe86187262666dbba11d
640244c"}}}%
❯ curl -X 'POST' \
  'http://127.0.0.1:8000/v1/verify-signature/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "message": "This is a secret message",
  "signature": "51a652912eb9ff4c39997dad3412ce8cc80f2652d26267e23e7fa6f51e43c6255bdeb1dc7c413
8f1702dad5b3b4f2fc8bdd86cf32d37d748082c6b3016b5d3154b3e36c140b0b239c25ccd188fc6f1c80d8de6729b
72a464e78bf0776a270296232a91d4f6e14be885064d03a21400812aeff4449375ccbdf3da8412f458e38ce921b87
3232f6824086dd07e9f7d5a9190f5eada77d46dc9750b7c1f56a54df50c7d07e64df70eec96461bcfead1177ed6bd
cf56dfc4a44cd347b4084b75d5aab3e8152b86e8a741a115a07ffe7419efab95ff36b30354e5d2f82b3e5427ccb61
16f68293a455e8e99a56f244f7e27a2e5221fe86187262666dbba11d640244c"
}'
{"apiVersion":"1.0","requestId":"bd5f20e8-6289-43da-b5a7-42b64b6a926d","context":"Verifying o
f signature by public key","data":{"item":{"valid":true}}}%
```

Змінив останній символ в зашифрованому хексі підпису:

```
❯ curl -X 'POST' \
  'http://127.0.0.1:8000/v1/verify-signature/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "message": "This is a secret message",
  "signature": "51a652912eb9ff4c39997dad3412ce8cc80f2652d26267e23e7fa6f51e43c6255bdeb1dc7c413
8f1702dad5b3b4f2fc8bdd86cf32d37d748082c6b3016b5d3154b3e36c140b0b239c25ccd188fc6f1c80d8de6729b
72a464e78bf0776a270296232a91d4f6e14be885064d03a21400812aeff4449375ccbdf3da8412f458e38ce921b87
3232f6824086dd07e9f7d5a9190f5eada77d46dc9750b7c1f56a54df50c7d07e64df70eec96461bcfead1177ed6bd
cf56dfc4a44cd347b4084b75d5aab3e8152b86e8a741a115a07ffe7419efab95ff36b30354e5d2f82b3e5427ccb61
16f68293a455e8e99a56f244f7e27a2e5221fe86187262666dbba11d640244b"
}'
{"detail":{"apiVersion":"1.0","requestId":"c1469a50-a121-45c9-b4e6-7cdae787c56f","context":"V
erifying of signature by public key","error":{"code":"400","message":"Wrong signature!"}}}%
```

4. Код

```
import uuid

from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes
from cryptography.exceptions import InvalidSignature

app = FastAPI()

BITS = 2048
PRIVATE_KEY_PATH = "private_key.pem"
PUBLIC_KEY_PATH = "public_key.pem"


class Message(BaseModel):
    message: str


class Signature(BaseModel):
    message: str
    signature: str


def load_private_key(private_key_path):
    with open(private_key_path, "rb") as key_file:
        private_key = serialization.load_pem_private_key(
            key_file.read(),
            password=None
        )
    return private_key


def load_public_key(public_key_path):
    with open(public_key_path, "rb") as key_file:
        public_key = serialization.load_pem_public_key(
            key_file.read()
        )
    return public_key
```

```
# Generate RSA keys and save to files
@app.post("/v1/generate-keys/")
def generate_keys():
    request_id = str(uuid.uuid4())
    context = "Private and Public keys generation"
    try:
        private_key = rsa.generate_private_key(public_exponent=65537,
key_size=BITS)
        public_key = private_key.public_key()

        with open(PRIVATE_KEY_PATH, "wb") as f:
            f.write(private_key.private_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PrivateFormat.PKCS8,
                encryption_algorithm=serialization.NoEncryption()
            ))

        with open(PUBLIC_KEY_PATH, "wb") as f:
            f.write(public_key.public_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PublicFormat.SubjectPublicKeyInfo
            ))
        return {
            "apiVersion": "1.0",
            "requestId": request_id,
            "context": context,
            "data": {
                "item": {
                    "privateKeyPath": PRIVATE_KEY_PATH,
                    "publicKeyPath": PUBLIC_KEY_PATH
                }
            }
        }
    except Exception as e:
        content = {
            "apiVersion": "1.0",
            "requestId": request_id,
            "context": context,
            "error": {
                "code": str(500),
                "message": str(e)
            }
```

```
        }
        raise HTTPException(status_code=500, detail=content)


# Sign a message
@app.post("/v1/sign-message/")
def sign_message(message: Message):
    request_id = str(uuid.uuid4())
    context = "Signing of message by private key"
    try:
        private_key = load_private_key(PRIVATE_KEY_PATH)
        message_bytes = message.message.encode()
        signature = private_key.sign(
            message_bytes,
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA256()
        )
        return {
            "apiVersion": "1.0",
            "requestId": request_id,
            "context": context,
            "data": {
                "item": {
                    "signature": signature.hex()
                }
            }
        }
    except Exception as e:
        content = {
            "apiVersion": "1.0",
            "requestId": request_id,
            "context": context,
            "error": {
                "code": str(500),
                "message": str(e)
            }
        }
        raise HTTPException(status_code=500, detail=content)
```

```
# Verify a signature
@app.post("/v1/verify-signature/")
def verify_signature(signature: Signature):
    request_id = str(uuid.uuid4())
    context = "Verifying of signature by public key"
    try:
        public_key = load_public_key(PUBLIC_KEY_PATH)
        message_bytes = signature.message.encode()
        signature_bytes = bytes.fromhex(signature.signature)

        public_key.verify(
            signature_bytes,
            message_bytes,
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA256()
        )
        return {
            "apiVersion": "1.0",
            "requestId": request_id,
            "context": context,
            "data": {
                "item": {
                    "valid": True
                }
            }
        }
    except InvalidSignature:
        content = {
            "apiVersion": "1.0",
            "requestId": request_id,
            "context": context,
            "error": {
                "code": str(400),
                "message": "Wrong signature!"
            }
        }
        raise HTTPException(status_code=400, detail=content)
    except Exception as e:
        content = {
            "apiVersion": "1.0",
```

```
            "requestId": request_id,
            "context": context,
            "error": {
                "code": str(500),
                "message": str(e)
            }
        }
        raise HTTPException(status_code=500, detail=content)


if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```