

Securitate Informațională
[Proiect]

Sistem local de management al cheilor de
criptare pentru mai mulți algoritmi
Documentație

Proiect realizat de:

Georgiță Adrian
Mihălucă Mădălina-Maria
Popa Andrei
Grupa 1409B

Sub coordonarea profesorului:
ș.l. dr.ing. Tudorache Alexandru-Gabriel

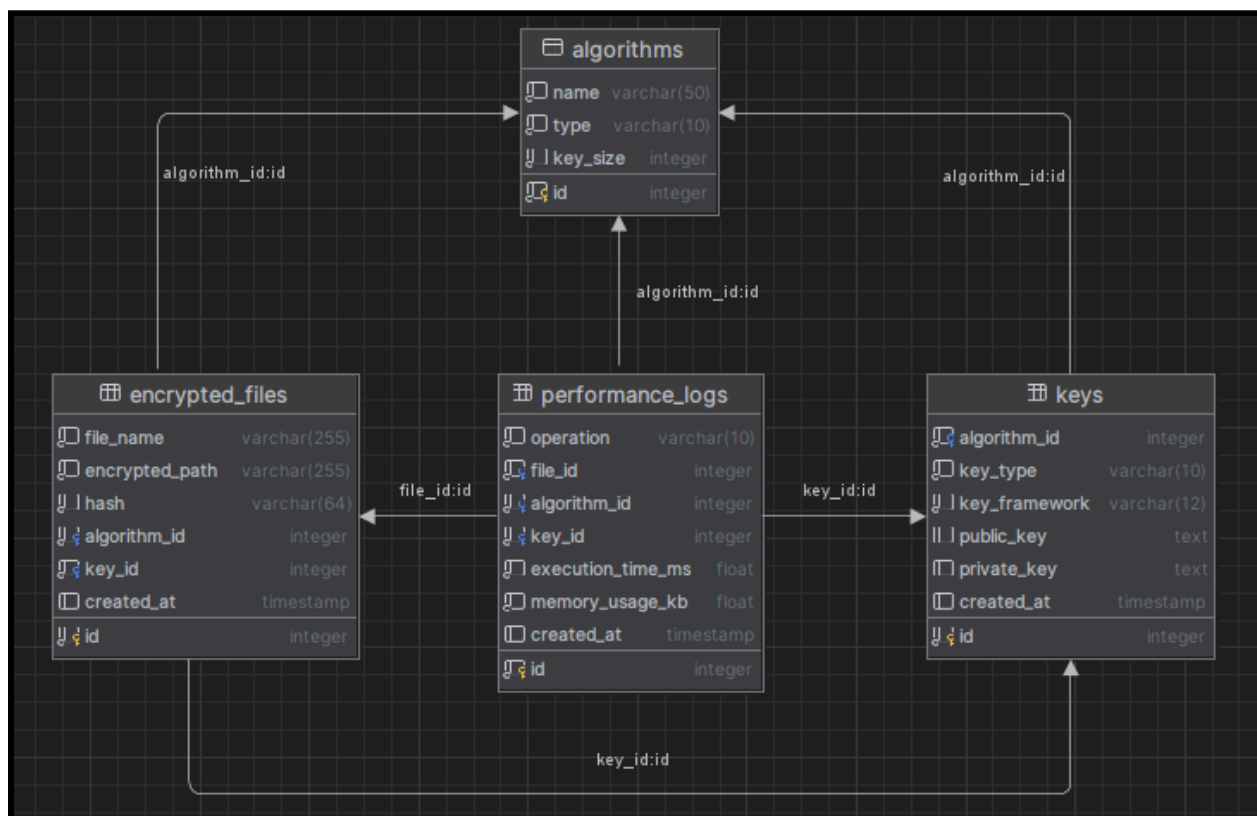
1. Introducere

Proiectul urmărește realizarea unui sistem local de management al cheilor de criptare, cu suport pentru criptarea și decriptarea fișierelor utilizând atât un algoritm simetric (AES), cât și unul asimetric (RSA). Aplicația permite generarea și stocarea cheilor, selectarea algoritmilor, criptarea/decriptarea fișierelor și monitorizarea performanței acestor operații (timp de execuție, memorie utilizată).

Pentru implementare s-au utilizat două framework-uri: OpenSSL și PyCryptodome, permițând compararea eficienței acestora. Toate operațiile sunt gestionate printr-o bază de date relațională, care include entități precum algoritmi, chei, fișiere criptate și loguri de performanță.

2. Designul bazei de date

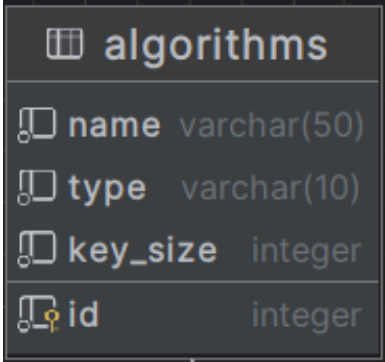
Sistemul proiectat utilizează o bază de date relațională pentru a gestiona informațiile legate de algoritmi criptografici, chei, fișiere criptate și performanțele operațiilor de criptare și decriptare.



Aceasta este alcătuită din cinci tabele principale:

2.1. Tabelul algorithms

Centralizează informațiile despre algoritmi criptografici utilizați în aplicație. Fiecare algoritm este definit printr-un identificator unic, un nume descriptiv, un tip (simetric sau asimetric) și dimensiunea cheii.

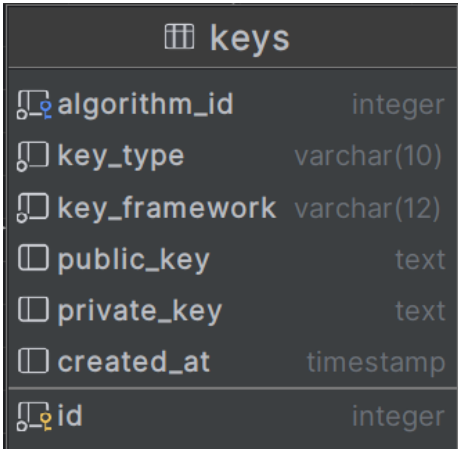
	Numele algoritmului (ex: AES, RSA)
	Tipul algoritmului (sym pentru simetric, asym pentru asimetric)
	Dimensiunea în biți a cheii utilizate (ex: 128, 256, 2048)
	Identificator unic al algoritmului

Relații: Un algoritm poate fi folosit în mai multe fișiere criptate (encrypted_files), în chei (keys) și în logurile de performanță (performance_logs).

Rol: Permite identificarea și clasificarea algoritmilor folosiți, fiind esențial pentru analiza și compararea performanței între diferite metode criptografice.

2.2. Tabelul keys

Sochează toate cheile criptografice generate în aplicație, fie ele simetrice sau asimetrice. Se înregistrează atât cheia publică, cât și cea privată (dacă este cazul), precum și framework-ul utilizat la generare.

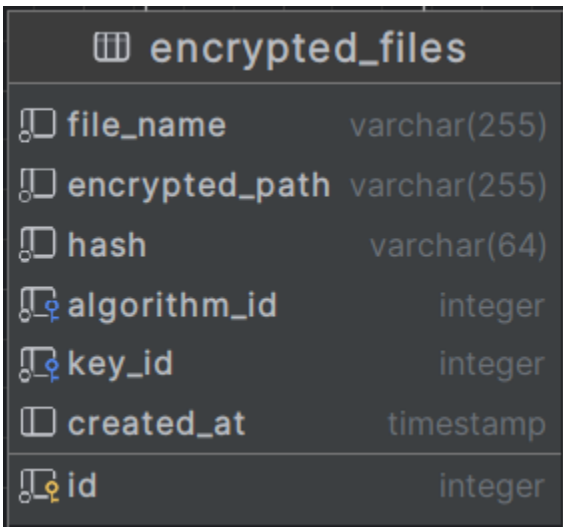
	Algoritmul asociat cheii
	Tipul cheii: symmetric, asymmetric
	Tehnologia folosită (OpenSSL, PyCryptodome)
	Conținutul cheii publice (pentru RSA)
	Conținutul cheii private sau simetrice
	Data și ora generării cheii
	Identificator unic al cheii

Relații: Legătură cu algoritmi (prin `algorithm_id`) și referință în fișierele criptate (`encrypted_files`) și logurile de performanță (`performance_logs`)

Rol: Asigură managementul sigur și trasabil al cheilor criptografice utilizate în procesele de criptare și decriptare.

2.3. Tabelul `encrypted_files`

Păstrează informații despre fiecare fișier criptat, incluzând numele fișierului, locația acestuia, algoritmul și cheia utilizată, precum și hash-ul SHA al fișierului original.

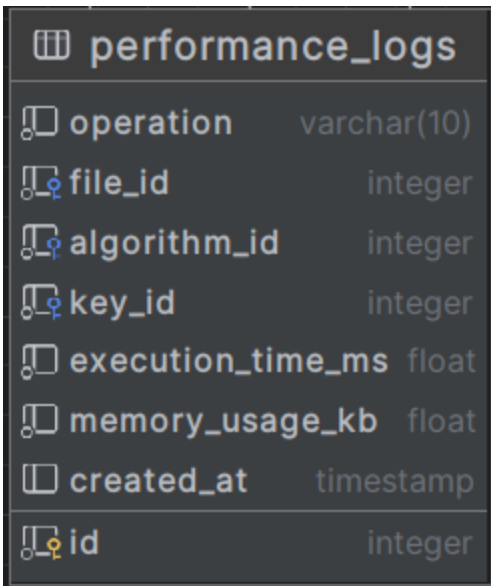
	Identificator unic al fișierului criptat
	Numele fișierului original
	Calea către fișierul criptat pe disc
	Hash SHA256 al fișierului original (pentru verificarea integrității)
	Algoritmul utilizat
	Cheia utilizată la criptare
	Data la care a avut loc criptarea

Relații: Legătură cu `algorithms` și `keys` și referință în `performance_logs`

Rol: Păstrează o evidență completă a fișierelor procesate de sistem și le leagă direct de tehnologiile și cheile folosite.

2.4. Tabelul `performance_logs`

Înregistrează informații despre performanța fiecărei operații de criptare și decriptare, cum ar fi timpul de execuție și memoria folosită. Este esențial pentru analiza eficienței algoritmilor.

	Identificatorul unic al logului
	Tipul operației: encrypt sau decrypt
	Fișierul implicat în operație
	Algoritmul folosit
	Cheia utilizată
	Timpul de execuție în milisecunde
	Cantitatea maximă de memorie folosită (în kilobytes)
	Momentul efectuării operației

Relații: Relații multiple către encrypted_files, algorithms, keys

Rol: Permite monitorizarea și compararea performanței între algoritmi și framework-uri, facilitând optimizarea procesului criptografic.

3. Algoritmi și framework-uri utilizate

3.1. Algoritmi aleși

În cadrul acestui proiect au fost selectați doi algoritmi criptografici, reprezentând ambele paradigme ale criptării:

- **AES (Advanced Encryption Standard)** – algoritm **simetric**, utilizat pe scară largă pentru criptarea rapidă și sigură a datelor. Este cunoscut pentru eficiența sa ridicată și este adoptat ca standard de criptare de către guverne și organizații internaționale. În proiect este folosit în mod implicit cu o cheie de 256 biți și modul de operare **EAX**, care oferă confidențialitate și integritate.
- **RSA (Rivest-Shamir-Adleman)** – algoritm **asimetric**, utilizat pentru criptarea cheilor, autentificare și schimb securizat de informații. RSA este un algoritm matur, cu aplicabilitate extinsă, însă este mai lent decât AES și implică dimensiuni mai mari ale cheilor (ex: 2048 biți). În proiect, RSA este folosit pentru criptarea cheii AES într-un mecanism hibrid.

3.2. Framework-uri folosite

Proiectul integrează două framework-uri diferite pentru criptare și decriptare, oferind astfel posibilitatea de **comparare a performanțelor** între implementări:

- **OpenSSL**

Este una dintre cele mai utilizate biblioteci criptografice open-source. În acest proiect, OpenSSL este utilizat prin comenzi apelate din linia de comandă folosind modulul Python subprocess. OpenSSL oferă performanțe excelente și este utilizat pe scară largă în industrie (ex: HTTPS, VPN-uri, certificate digitale).

- **PyCryptodome**

Este o bibliotecă Python care oferă o alternativă pur Python la OpenSSL, cu o interfață simplificată și integrare nativă. Este folosită pentru criptare directă în Python, fără a apela procese externe. PyCryptodome permite o implementare mai flexibilă și ușor de personalizat, fiind potrivită pentru testare și prototipare.

Aceste două framework-uri, împreună cu cei doi algoritmi principali, permit realizarea unor comparații obiective asupra **timpului de execuție** și a **memoriei consumate**, date care sunt înregistrate și analizate în cadrul aplicației. Astfel, proiectul nu doar aplică criptografia, ci oferă și o platformă de studiu a performanței algoritmilor în contexte reale.

4. Fluxul aplicației

4.1. Selectarea fișierului

Utilizatorul selectează un fișier local pe care dorește să îl creeze sau să îl decripteze. Aplicația extrage numele fișierului și îl verifică pentru a evita suprascrierea sau procesarea dublă.

4.2. Selectarea algoritmului și framework-ului

Utilizatorul alege unul dintre algoritmii disponibili (AES sau RSA) și framework-ul dorit (OpenSSL sau PyCryptodome) pentru efectuarea operației.

4.3. Generarea cheii

În funcție de algoritmul ales, aplicația generează o cheie criptografică (simetrică sau pereche publică/privată) și o stochează în baza de date, împreună cu informații despre framework-ul folosit și algoritm.

4.4. Criptarea fișierului

După selectarea cheii și a parametrilor, utilizatorul inițiază procesul de criptare. Aplicația:

- Criptează fișierul folosind algoritmul și framework-ul selectate
- Generează un hash SHA al fișierului original
- Salvează fișierul criptat, hash-ul și metadatele asociate în baza de date
- Măsoară performanța (timp, memorie) și o înregistrează

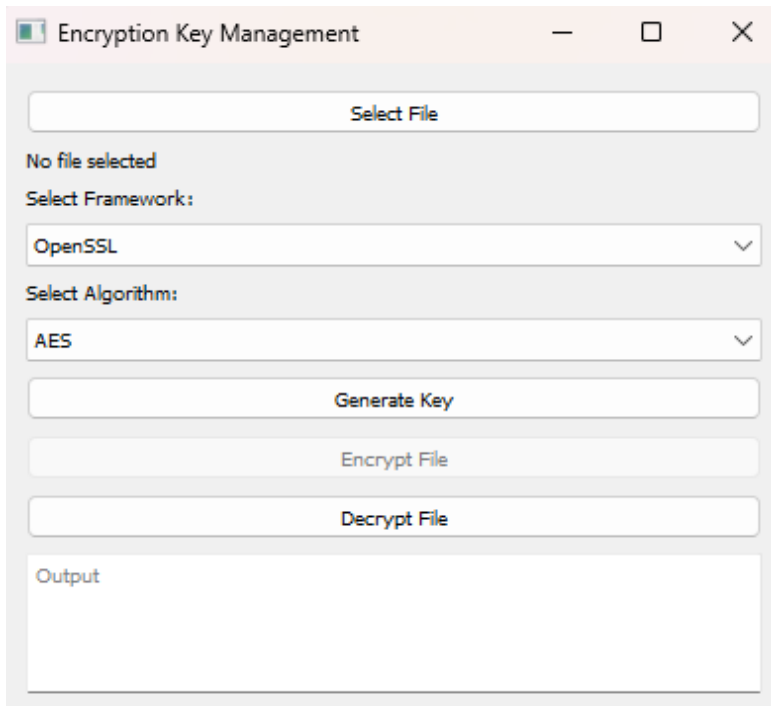
4.5. Decriptarea fișierului

Pentru decriptare, aplicația:

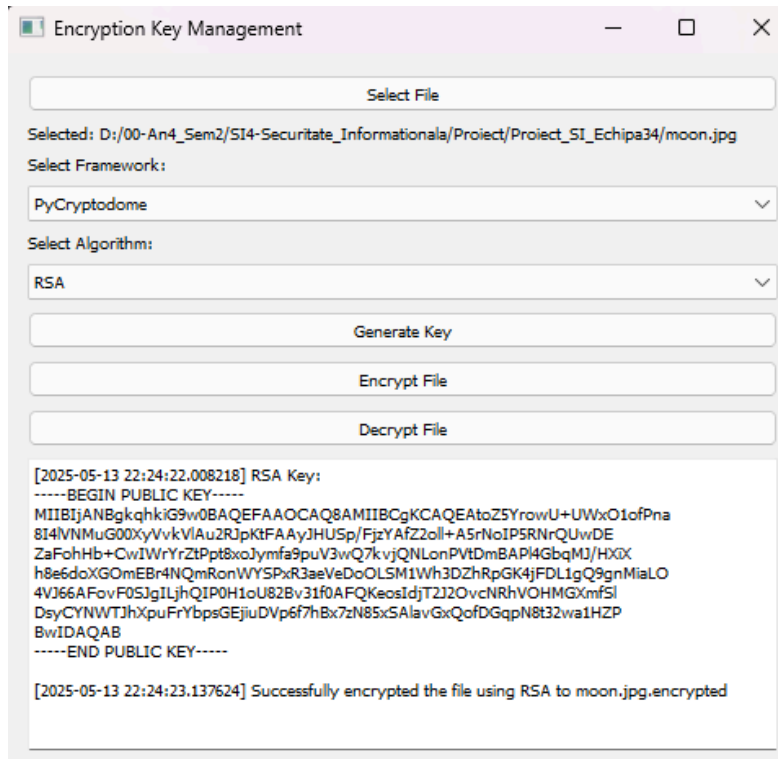
- Identifică fișierul criptat selectat
- Recuperează cheia asociată din baza de date
- Decriptează fișierul și verifică integritatea acestuia prin compararea hash-urilor
- Afișează rezultatul și salvează performanțele înregistrate

5. Rezultate

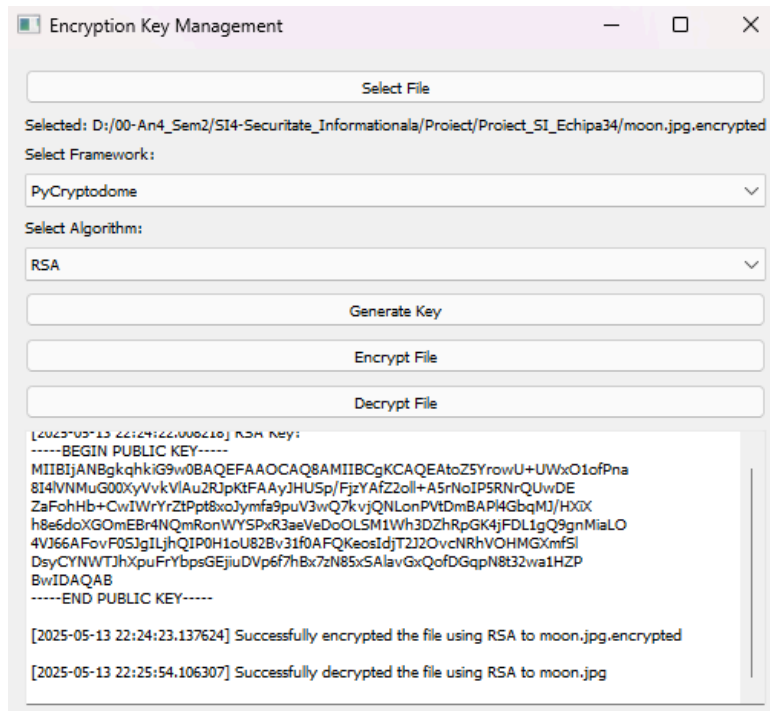
5.1 Interfața cu utilizatorul



5.2 Criptarea unui fișier .jpg folosind PyCryptodome și algoritmul RSA



5.3. Decriptarea fișierului mai devreme criptat



6. Secvențe semnificative de cod

6.1. Algoritmul AES de criptare folosind framework-ul PyCryptodome

Python

```
def encryptAES(file_path, db, currentKey):
    encrypted_file_path = (file_path + ".encrypted")

    input_file = open(file_path, "rb")
    output_file = open(encrypted_file_path, "wb")

    with open("keys/keyfile.txt", "r") as key_file:
        key = bytes.fromhex(key_file.read())

    with open(file_path, "rb") as input_file, open(encrypted_file_path, "wb")
as output_file:
        cipher = AES.new(key, AES.MODE_EAX)
        nonce = cipher.nonce
        output_file.write(nonce)

        while chunk := input_file.read(64 * 1024):
            ciphertext = cipher.encrypt(chunk)
            output_file.write(ciphertext)

        tag = cipher.digest()
        output_file.write(tag)

        hash = hashing.hash_file(file_path)

        file_name = os.path.basename(file_path)
        file_name = file_name.replace(".encrypted", "")

        file_dto = EncryptedFileDTO(file_name=file_name,
encrypted_path=encrypted_file_path, hash=hash, algorithm_id=1,
key_id=currentKey, created_at=datetime.now())
        file = EncryptedFileCRUD.create_file(db, file_dto)

    return True, "Successfully encrypted the file", file
```

6.2. Algoritmul RSA de criptare folosind framework-ul OpenSSL

Python

```
def encryptRSA(file_path, db, currentKey):
    encrypted_file_path = (file_path + ".encrypted")
    key_file_path = encrypted_file_path + ".key"

    # Generated the AES Key
    command = ["openssl", "rand", "-hex", "32"]
    try:
        result = subprocess.run(command, capture_output=True, text=True,
                                check=True)
        with open("keys/tempAESKey.txt", "w") as key_file:
            key_file.write(result.stdout.strip())
    except subprocess.CalledProcessError as e:
        print(f"Error generating OpenSSL AES key for the RSA Encryption: {e}")
        return False, "Error generating OpenSSL AES key for the RSA Encryption"

    try:
        # Encrypt the file using AES
        subprocess.run([
            "openssl", "enc", "-aes-256-cbc",
            "-in", file_path,
            "-out", encrypted_file_path,
            "-kfile", "keys/tempAESKey.txt"
        ], check=True)

        # Encrypt the AES Key using RSA
        subprocess.run([
            "openssl", "pkeyutl", "-encrypt",
            "-pubin", "-inkey", "keys/public.pem",
            "-in", "keys/tempAESKey.txt",
            "-out", key_file_path
        ], check=True)

        os.remove("keys/tempAESKey.txt")

    hash = hashing.hash_file(file_path)

    file_name = os.path.basename(file_path)
    file_name = file_name.replace(".encrypted", "")

    file_dto = EncryptedFileDTO(file_name=file_name,
                                encrypted_path=encrypted_file_path, hash=hash, algorithm_id=0,
```

```

                                key_id=currentKey,
created_at=datetime.now())
    file = EncryptedFileCRUD.create_file(db, file_dto)

    print(f"File encrypted successfully: {encrypted_file_path}")
    return True, "File encrypted successfully", file

except subprocess.CalledProcessError as e:
    print(f"Error during encryption: {e}")
    if os.path.exists("keys/tempAESKey.txt"):
        os.remove("keys/tempAESKey.txt")
    return False, "Error during encryption", None

```

6.3. Generarea cheii de criptare AES

Python

```

def generateKeyAES(db):
    key = get_random_bytes(32)

    with open("keys/keyfile.txt", "w") as key_file:
        key_file.write(key.hex())

    key_string = key.hex()

    key_dto = KeyDTO(
        id=None,
        algorithm_id=1,
        key_type="symmetric",
        key_framework="PyCryptodome",
        public_key=None,
        private_key=key.hex(),
        created_at=datetime.now())
    key = KeyCRUD.create_key(db, key_dto)

    print("AES key successfully generated and saved!")
    return key.id, key_string

```

7. Dificultăți întâmpinate

7.1. Criptarea cu RSA

Algoritmul RSA nu este proiectat pentru criptarea directă a fișierelor de dimensiuni mari. Acest lucru se datorează faptului că RSA poate cripta doar un volum foarte mic de date raportat la dimensiunea cheii.

Pentru a depăși această limitare, folosim o abordare hibridă:

- Fișierul este criptat cu un algoritm simetric AES
- Cheia AES este apoi criptată cu RSA, pentru a asigura confidențialitatea acesteia.

Astfel, combinația AES + RSA oferă atât performanță (prin AES), cât și securitate în schimbul de chei (prin RSA).

7.2. Măsurarea performanței

Capturarea corectă a timpului de execuție și a memoriei utilizate, într-un mod care să nu afecteze comportamentul real al aplicației, a fost o etapă sensibilă. A fost nevoie de testare și ajustare a modului în care se loghează performanțele pentru a evita crash-uri și scurgeri de resurse.

8. Concluzie

Aplicația implementează funcționalitățile esențiale pentru criptarea și decriptarea fișierelor folosind algoritmi clasici precum AES și RSA. Structura modulară și utilizarea unei baze de date relaționale permit gestionarea cheilor, a fișierelor și a performanțelor într-un mod organizat. Prin integrarea a două framework-uri diferite, se pot compara rezultatele în condiții controlate. Proiectul oferă un punct de plecare solid pentru îmbunătățiri viitoare.