Laboratorium nr 6

Temat: Programowanie aplikacji w Azure

1 Zadanie 1 – Utworzenie API w Azure Functions (HTTP Trigger)

1.1 Cel

Celem zadania było utworzenie prostego API w Azure Functions, które obsługuje żądania HTTP, przyjmuje parametr `name` i zwraca dynamiczną odpowiedź w formacie JSON. Funkcję przetestowano lokalnie w Postmanie.

1.2 Tworzenie projektu

1. Utworzono folder projektu w VS Code i wirtualne środowisko Pythona.
2. Zainstalowano pakiety: `azure-functions`.
3. Zainicjalizowano projekt Azure Functions:

```
func init MyFunctionProj --worker-runtime python
cd MyFunctionProj
```

1.3 Dodanie HTTP Trigger

```
func new --name HelloFunction --template "HTTP trigger" --authlevel "anonymous"
```

1.4 Implementacja funkcji

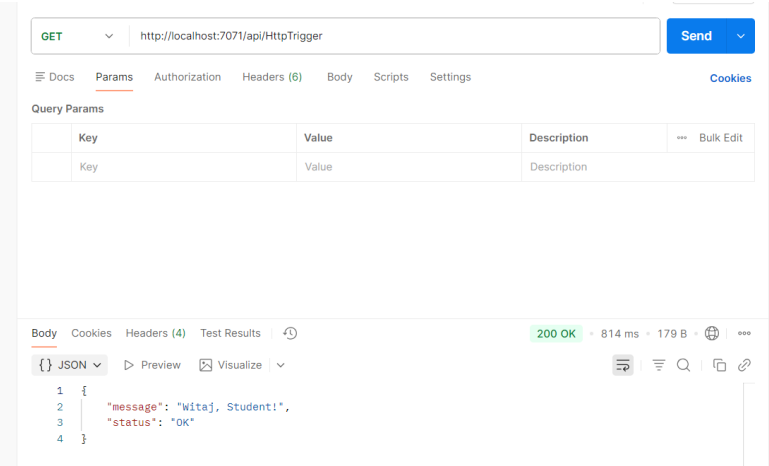
```
import azure.functions as func
import json

def main(req: func.HttpRequest) -> func.HttpResponse
    :
name = req.params.get('name')
if not name:
try:
req_body = req.get_json()
except ValueError:
req_body = {}
name = req_body.get('name')

response = {"message": f"Hello, {name or 'World'}!"}
return func.HttpResponse(
json.dumps(response),
mimetype="application/json",
status_code=200
)
```

1.5 Uruchomienie i testowanie

Funkcję uruchomiono lokalnie:

```
func start
```

Endpoint dostępny był pod adresem: `http://localhost:7071/api/HelloFunction`
Testy wykonano w Postmanie – wysłano żądanie GET z parametrem `name`.



Rysunek 1: Test API w Postmanie – odpowiedź JSON z dynamicznym imieniem

2 Zadanie 2 – System przesyłania plików do Azure Storage

2.1 Cel

Celem zadania było utworzenie systemu przesyłania plików do Azure Blob Storage. System obsługuje żądania HTTP POST z plikami zakodowanymi w base64, generuje unikalne nazwy blobów z timestamp, zapisuje plik do Azure Blob Storage i zwraca szczegółowe informacje o przesłanym pliku w formacie JSON wraz z SAS URL.

2.2 Implementacja funkcji `upload_file` z Azure Blob Storage

Listing 1: Funkcja do przesyłania plików do Azure Blob Storage

```
import azure.functions as func
import json
import base64
from datetime import datetime, timedelta
```

```
import logging
from azure.storage.blob import BlobServiceClient,
    generate_blob_sas, BlobSasPermissions
import uuid

app = func.FunctionApp()

STORAGE_CONNECTION_STRING = "
    DefaultEndpointsProtocol=https;AccountName=
    z6storage;AccountKey=..."
CONTAINER_NAME = "uploads"

@app.route(route="UploadFile", methods=["POST"])
def upload_file(req: func.HttpRequest) -> func.
    HttpResponse:
try:
logging.info('File␣upload␣function␣started.')
req_body = req.get_json()

if 'filename' not in req_body or 'content' not in
    req_body:
return func.HttpResponse(
json.dumps({"error": "Missing␣filename␣or␣content"})
    ,
mimetype="application/json",
status_code=400
)

filename = req_body['filename']
file_content = base64.b64decode(req_body['content'])
logging.info(f'Received␣file:␣{filename},␣size:␣{len
    (file_content)}␣bytes')

# Generowanie unikalnej nazwy blob
blob_name = f"{datetime.utcnow().strftime('%Y%m%d_%H
    %M%S')}_{uuid.uuid4().hex[:8]}_{filename}"

# Upload do Azure Blob Storage
blob_service_client = BlobServiceClient.
    from_connection_string(STORAGE_CONNECTION_STRING)
blob_client = blob_service_client.get_blob_client(
container=CONTAINER_NAME,
blob=blob_name
)

blob_client.upload_blob(file_content, overwrite=True
    )
logging.info(f'File␣uploaded␣to␣Blob␣Storage:␣{
    blob_name}')
```

```
        # Generowanie SAS URL (wa ny 24h)
        sas_token = generate_blob_sas(
        account_name=blob_service_client.account_name,
        container_name=CONTAINER_NAME,
        blob_name=blob_name,
        account_key=blob_service_client.credential.
            account_key,
        permission=BlobSasPermissions(read=True),
        expiry=datetime.utcnow() + timedelta(hours=24)
        )

        blob_url = f"https://{blob_service_client.
            account_name}.blob.core.windows.net/{
            CONTAINER_NAME}/{blob_name}"
        sas_url = f"{blob_url}?{sas_token}"

        response = {
                "success": True,
                "message": "File uploaded to Azure Blob
                    Storage",
                "filename": filename,
                "blob_name": blob_name,
                "file_size": len(file_content),
                "blob_url": blob_url,
                "sas_url": sas_url,
                "timestamp": datetime.utcnow().isoformat(),
        }

        return func.HttpResponse(
        json.dumps(response),
        mimetype="application/json",
        status_code=200
        )

        except Exception as e:
        logging.error(f'Error: {str(e)}')
        return func.HttpResponse(
        json.dumps({"error": str(e)}),
        mimetype="application/json",
        status_code=500
        )
```

2.3 Blob Trigger – automatyczne logowanie uploadów

Listing 2: Blob Trigger do logowania uploadów do Azure Table Storage

```
        import azure.functions as func
        import logging
        from datetime import datetime
```

```
        from azure.data.tables import TableServiceClient

        @app.blob_trigger(
        arg_name="myblob",
        path="uploads/{name}",
        connection="AzureWebJobsStorage"
        )
        def blob_trigger(myblob: func.InputStream):
        logging.info(f"Blob␣trigger␣activated␣for:␣{myblob.
            name}")

        # Zapis metadanych do Azure Table Storage
        try:
        table_service = TableServiceClient.
            from_connection_string("DefaultEndpointsProtocol=
            https;AccountName=z6storage;...")
        table_client = table_service.get_table_client("
            filelogs")

        entity = {
                "PartitionKey": "upload",
                "RowKey": datetime.utcnow().strftime('%Y%m%
                    d_%H%M%S_%f'),
                "filename": myblob.name,
                "size": myblob.length,
                "uploaded_at": datetime.utcnow().isoformat()
                    ,
                "container": "uploads"
        }

        table_client.create_entity(entity=entity)
        logging.info(f"Logged␣upload␣to␣Table␣Storage:␣{
            myblob.name}")

        except Exception as e:
        logging.error(f"Error␣logging␣to␣Table␣Storage:␣{str
            (e)}")
```

2.4 Konfiguracja – `local.settings.json`

Listing 3: Plik konfiguracyjny z connection string

```
        {
                "IsEncrypted": false,
                "Values": {
                        "AzureWebJobsStorage": "
                            DefaultEndpointsProtocol=https;
                            AccountName=z6storage;AccountKey
                            =...",
```

```
                        "FUNCTIONS_WORKER_RUNTIME": "python
                            ",
                        "STORAGE_CONNECTION_STRING": "
                            DefaultEndpointsProtocol=https;
                            AccountName=z6storage;AccountKey
                            =..."
                }
        }
```

2.5 Testowanie w Postmanie

2.5.1 Request POST – Upload pliku

Metoda: POST
URL: `https://z6functions.azurewebsites.net/api/UploadFile`
Headers: Content-Type: application/json
Body (raw JSON):

```
        {
                "filename": "test.txt",
                "content": "SGVsbG8gZnJvbSBQb3N0bWFuIQ=="
        }
```

Odpowiedź (200 OK):

```
        {
                "success": true,
                "message": "File uploaded to Azure Blob
                    Storage",
                "filename": "test.txt",
                "blob_name": "20251204_182815_a1b2c3d4_test.
                    txt",
                "file_size": 20,
                "blob_url": "https://z6storage.blob.core.
                    windows.net/uploads/20251204
                    _182815_a1b2c3d4_test.txt",
                "sas_url": "https://z6storage.blob.core.
                    windows.net/uploads/20251204
                    _182815_a1b2c3d4_test.txt?sv=...",
                "timestamp": "2025-12-04T18:28:15.888956"
        }
```

2.5.2 Weryfikacja w Azure Portal

- Plik dostępny w kontenerze `uploads` w Storage Account `z6storage`
- Logi uploadów w tabeli `filelogs` w Azure Table Storage
- SAS URL umożliwia pobranie pliku przez 24 godziny

3 Zadanie 3 – Harmonogramowane zadania (Timer Trigger)

3.1 Cel

Celem zadania było utworzenie funkcji uruchamianej periodycznie co minutę. Funkcja zapisuje timestamp do **Azure Table Storage**, a logi są dostępne przez dedykowany endpoint HTTP.

3.2 Implementacja Timer Trigger z Azure Table Storage

Listing 4: Implementacja Timer Trigger z zapisem do Azure Tables

```
import azure.functions as func
from datetime import datetime
import logging
from azure.data.tables import TableServiceClient
import uuid

@app.timer_trigger(schedule="0 */1 * * * *",
    arg_name="myTimer", run_on_startup=True)
def timer_trigger(myTimer: func.TimerRequest):
logging.info('Timer trigger function executed.')

timestamp = datetime.utcnow().isoformat()

# Zapis do Azure Table Storage
try:
table_service = TableServiceClient.
    from_connection_string("DefaultEndpointsProtocol=
    https;AccountName=z6storage;...")
table_client = table_service.get_table_client("
    timerlogs")

entity = {
        "PartitionKey": "timer",
        "RowKey": datetime.utcnow().strftime('%Y%m%
            d_%H%M%S_%f'),
        "executed_at": timestamp,
        "task_name": "scheduled_task",
        "status": "success"
}

table_client.create_entity(entity=entity)
logging.info(f'[TIMER TRIGGER] Log saved to Azure
    Table: {timestamp}')

except Exception as e:
```

```
        logging.error(f'[TIMER TRIGGER] Error saving to
            Table: {str(e)}')
```

3.3 Endpoint do przeglądu logów z Azure Table Storage

Listing 5: Endpoint GetTimerLogs pobierający dane z Azure Tables

```
        @app.route(route="TimerLogs", methods=["GET"])
        def get_timer_logs(req: func.HttpRequest) -> func.
            HttpResponse:
        """
        Endpoint do przegl du log w z Timer Trigger z
            Azure Table Storage.
        """
        try:
        # Pobierz logi z Azure Table Storage
        table_service = TableServiceClient.
            from_connection_string("DefaultEndpointsProtocol=
            https;AccountName=z6storage;...")
        table_client = table_service.get_table_client("
            timerlogs")

        entities = table_client.query_entities("PartitionKey
             eq 'timer'")
        logs = []

        for entity in entities:
        logs.append({
                "executed_at": entity["executed_at"],
                "task_name": entity["task_name"],
                "status": entity["status"]
        })

        logs.sort(key=lambda x: x['executed_at'], reverse=
            True)

        response = {
                "total_executions": len(logs),
                "logs": logs[:20],  # Ostatnie 20 wykonan
                "message": "Timer Trigger uruchamia si Ź co
                    minut Ź",
                "source": "Azure Table Storage"
        }

        return func.HttpResponse(
        json.dumps(response),
        mimetype="application/json",
        status_code=200
        )
```

```
except Exception as e:
return func.HttpResponse(
json.dumps({"error": str(e)}),
mimetype="application/json",
status_code=500
)
```

3.4 Testowanie

Endpoint: `GET https://z6functions.azurewebsites.net/api/TimerLogs`
Odpowiedź po kilku wykonaniach:


```
{
        "total_executions": 5,
        "logs": [
        {
                "executed_at": "2025-12-07T14
                    :05:00.123456",
                "task_name": "scheduled_task",
                "status": "success"
        },
        {
                "executed_at": "2025-12-07T14
                    :04:00.123456",
                "task_name": "scheduled_task",
                "status": "success"
        },
        {
                "executed_at": "2025-12-07T14
                    :03:00.123456",
                "task_name": "scheduled_task",
                "status": "success"
        }
        ],
        "message": "Timer Trigger uruchamia si Ź co
            minut Ź",
        "source": "Azure Table Storage"
}
```


3.5 Potwierdzenie działania w Azure

- Dane trwale przechowywane w tabeli `timerlogs` w Azure Table Storage
- Logi dostępne nawet po restarcie funkcji
- Monitorowanie w Azure Portal: Application Insights i Table Storage
- Automatyczne skalowanie i wysoka dostępność

4 Zadanie 4 – Praca z danymi (Zapis i odczyt w bazie danych)

4.1 Cel

Celem zadania było stworzenie systemu CRUD (Create, Read, Update, Delete) dla produktów przy użyciu **Azure Tables** – zarządzanej usługi NoSQL w chmurze Microsoft Azure. System udostępnia RESTful API do zarządzania produktami z trwałym przechowywaniem danych w usłudze Azure.

- **Azure Storage Account:** `z6storage` (West Europe)
- **Azure Tables:** Zarządzana NoSQL na bazie Cosmos DB API
- **Persistent Data:** Dane zachowywane między wywołaniami funkcji
- **Automatic Replication:** Replikacja danych dla wysokiej dostępności

4.2 Konfiguracja Azure Tables

4.2.1 Inicjalizacja bazy danych

Tabela `products` jest tworzona automatycznie przy starcie aplikacji:

Listing 6: Inicjalizacja Azure Tables

```
        import os
        import uuid
        import json
        import logging
        from datetime import datetime
        from azure.data.tables import TableServiceClient
        import azure.functions as func

        STORAGE_CONNECTION_STRING = os.getenv("
            AzureWebJobsStorage", "")
        PRODUCTS_TABLE_NAME = "products"

        def get_table_client():
        """Pobierz␣klienta␣Azure␣Tables␣dla␣tabeli␣
            produkt w."""
        try:
        table_service_client = TableServiceClient.
            from_connection_string(
        STORAGE_CONNECTION_STRING
        )
        table_client = table_service_client.get_table_client
            (
        table_name=PRODUCTS_TABLE_NAME
        )
```

```
        logging.info(f'Connected␣to␣Azure␣Table:␣{
            PRODUCTS_TABLE_NAME}')
        return table_client
        except Exception as e:
        logging.error(f'Error␣connecting␣to␣Azure␣Table:␣{
            str(e)}')
        raise

        def init_database():
        """Utw rz␣tabel Ź␣je Żli␣nie␣istnieje."""
        try:
        table_service_client = TableServiceClient.
            from_connection_string(
        STORAGE_CONNECTION_STRING
        )
        table_service_client.create_table_if_not_exists(
        table_name=PRODUCTS_TABLE_NAME
        )
        logging.info('[DATABASE]␣Initialized␣Azure␣Table:␣
            products')
        except Exception as e:
        logging.error(f'[DATABASE]␣Error␣initializing:␣{str(
            e)}')

        # Uruchom przy starcie
        init_database()
```

4.2.2 Schemat tabeli

Każdy rekord w Azure Tables posiada następujące pola:

Pole	Typ	Rola	Opis
PartitionKey	STRING	Klucz partycji	Zawsze: "product"
RowKey	STRING	Klucz wiersza	UUID produktu
name	STRING	Dane	Nazwa produktu
description	STRING	Dane	Opis produktu
price	DOUBLE	Dane	Cena w USD
quantity	INT32	Dane	Liczba dostępnych sztuk
created_at	STRING	Metadane	ISO timestamp
updated_at	STRING	Metadane	ISO timestamp

4.3 Endpoint: SaveProduct – Zapis produktu

Metoda: POST
URL: `/api/SaveProduct`
Status: 201 Created

4.3.1 Request i Response

Listing 7: Przykładowy request SaveProduct

```
POST /api/SaveProduct
Content-Type: application/json

{
        "name": "Laptop Dell",
        "description": "High-performance laptop",
        "price": 1200,
        "quantity": 5
}
```

Listing 8: Przykładowy response SaveProduct

```
{
        "success": true,
        "message": "Product saved successfully",
        "product_id": "8f9e1b2c-3a4d-5e6f-7890-
            a1b2c3d4e5f6",
        "name": "Laptop Dell",
        "price": 1200,
        "quantity": 5,
        "created_at": "2025-12-07T15:30:22.450123"
}
```

4.3.2 Implementacja funkcji

Listing 9: Funkcja SaveProduct

```
@app.route(route="SaveProduct", methods=["POST"],
    auth_level=func.AuthLevel.ANONYMOUS)
def save_product(req: func.HttpRequest) -> func.
    HttpResponse:
try:
req_body = req.get_json()
if 'name' not in req_body or 'price' not in req_body
    :
return func.HttpResponse(json.dumps({"error": "
    Missing␣name␣or␣price"}), status_code=400)

product_id = str(uuid.uuid4())
now = datetime.utcnow().isoformat()
product_entity = {
        "PartitionKey": "product",
        "RowKey": product_id,
        "name": req_body['name'],
```

```
            "description": req_body.get('description', '
                ''),
            "price": float(req_body['price']),
            "quantity": int(req_body.get('quantity', 0))
                ,
            "created_at": now,
            "updated_at": now
        }

        table_client = get_table_client()
        table_client.upsert_entity(entity=product_entity)
        logging.info(f'[DATABASE] Saved product: {product_id
            }')

        response = {"success": True, "message": "Product
            saved successfully",
            "product_id": product_id, "name": req_body['
                name'],
            "price": req_body['price'], "quantity":
                req_body.get('quantity', 0),
            "created_at": now}
        return func.HttpResponse(json.dumps(response),
            status_code=201)
        except Exception as e:
        logging.error(f'[DATABASE] Error saving product: {
            str(e)}')
        return func.HttpResponse(json.dumps({"error": str(e)
            }), status_code=500)
```

4.4 Endpoint: GetProducts – Pobranie wszystkich produktów

Metoda: GET
URL: `/api/GetProducts`
Status: 200 OK

4.4.1 Implementacja

Listing 10: Funkcja GetProducts

```
        @app.route(route="GetProducts", methods=["GET"],
            auth_level=func.AuthLevel.ANONYMOUS)
        def get_products(req: func.HttpRequest) -> func.
            HttpResponse:
        try:
        table_client = get_table_client()
        entities = table_client.query_entities("PartitionKey
            eq 'product'")
```

```
        products = []
        for entity in entities:
        products.append({
                "id": entity["RowKey"],
                "name": entity["name"],
                "description": entity.get("description",""),
                "price": entity["price"],
                "quantity": entity["quantity"],
                "created_at": entity["created_at"],
                "updated_at": entity["updated_at"]
        })
        products.sort(key=lambda x: x['created_at'], reverse
            =True)
        response = {"success": True, "total": len(products),
             "products": products}
        return func.HttpResponse(json.dumps(response),
            status_code=200)
        except Exception as e:
        logging.error(f'[DATABASE] Error retrieving products
            : {str(e)}')
        return func.HttpResponse(json.dumps({"error": str(e)
            }), status_code=500)
```

4.5 Endpoint: GetProduct – Pobranie produktu po ID

Metoda: GET
URL: `/api/GetProduct?id=<product`$_id$`>`
`Status:`$200 OK / 404 Not Found$

Listing 11: Funkcja GetProduct

```
        @app.route(route="GetProduct", methods=["GET"],
            auth_level=func.AuthLevel.ANONYMOUS)
        def get_product(req: func.HttpRequest) -> func.
            HttpResponse:
        try:
        product_id = req.params.get('id')
        if not product_id:
        return func.HttpResponse(json.dumps({"error": "
            Missing id parameter"}), status_code=400)
        table_client = get_table_client()
        try:
        entity = table_client.get_entity("product",
            product_id)
        product = {"id": entity["RowKey"], "name": entity["
            name"],
                "description": entity.get("description",""),
                "price": entity["price"], "quantity": entity
                    ["quantity"],
```

```
            "created_at": entity["created_at"], "
                updated_at": entity["updated_at"]}
        return func.HttpResponse(json.dumps({"success": True
            , "product": product}), status_code=200)
        except Exception as e:
        if "ResourceNotFound" in str(e):
        return func.HttpResponse(json.dumps({"success":
            False, "error": "Product␣not␣found"}),
            status_code=404)
        raise
        except Exception as e:
        logging.error(f'[DATABASE]␣Error␣retrieving␣product:
            ␣{str(e)}')
        return func.HttpResponse(json.dumps({"error": str(e)
            }), status_code=500)
```

4.6 Endpoint: UpdateProduct – Aktualizacja produktu

Listing 12: Funkcja UpdateProduct

```
        @app.route(route="UpdateProduct", methods=["PUT"],
            auth_level=func.AuthLevel.ANONYMOUS)
        def update_product(req: func.HttpRequest) -> func.
            HttpResponse:
        try:
        req_body = req.get_json()
        if 'id' not in req_body:
        return func.HttpResponse(json.dumps({"error": "
            Missing␣product␣id"}), status_code=400)
        product_id = req_body['id']
        table_client = get_table_client()
        try:
        entity = table_client.get_entity("product",
            product_id)
        except Exception as e:
        if "ResourceNotFound" in str(e):
        return func.HttpResponse(json.dumps({"success":
            False, "error": "Product␣not␣found"}),
            status_code=404)
        raise
        entity_dict = dict(entity)
        for field in ['name','description','price','quantity
            ']:
        if field in req_body:
        entity_dict[field] = req_body[field]
        entity_dict['updated_at'] = datetime.utcnow().
            isoformat()
        table_client.upsert_entity(entity_dict)
```

```
        return func.HttpResponse(json.dumps({"success": True
            ,"message":"Product␣updated","product_id":
            product_id,"updated_at":entity_dict['updated_at'
            ]}), status_code=200)
        except Exception as e:
        logging.error(f'[DATABASE]␣Error␣updating␣product:␣{
            str(e)}')
        return func.HttpResponse(json.dumps({"error": str(e)
            }), status_code=500)
```

4.7 Endpoint: DeleteProduct – Usuwanie produktu

Listing 13: Funkcja DeleteProduct

```
        @app.route(route="DeleteProduct", methods=["DELETE"
            ], auth_level=func.AuthLevel.ANONYMOUS)
        def delete_product(req: func.HttpRequest) -> func.
            HttpResponse:
        try:
        product_id = req.params.get('id')
        if not product_id:
        return func.HttpResponse(json.dumps({"error":"
            Missing␣id␣parameter"}), status_code=400)
        table_client = get_table_client()
        try:
        table_client.delete_entity("product", product_id)
        return func.HttpResponse(json.dumps({"success":True,
            "message":"Product␣deleted","product_id":
            product_id}), status_code=200)
        except Exception as e:
        if "ResourceNotFound" in str(e):
        return func.HttpResponse(json.dumps({"success":False
            ,"error":"Product␣not␣found"}), status_code=404)
        raise
        except Exception as e:
        logging.error(f'[DATABASE]␣Error␣deleting␣product:␣{
            str(e)}')
        return func.HttpResponse(json.dumps({"error": str(e)
            }), status_code=500)
```

4.8 Testy i walidacja

- Test 1: Zapis trzech produktów { status 201, ID generowane unikatowo.
- Test 2: Pobranie wszystkich produktów { dane posortowane po created_at, persystentne.

- Test 3: Aktualizacja produktu { pola zmienione, timestamp updated_at zaktualizowany.
- Test 4: Usunięcie produktu { GET po ID zwraca 404.

5 Zadanie 5 – Key Vault – bezpieczne sekrety

5.1 Cel

Celem zadania było skonfigurowanie usługi Azure Key Vault do przechowywania poufnych danych (sekretów) oraz przygotowanie Azure Functions do pobierania sekretów z wykorzystaniem Managed Identity. Elementem zadania było również ręczne dodanie sekretu w panelu Azure Portal.

5.2 Problem z dodaniem sekretu – błąd RBAC

Podczas próby dodania nowego sekretu do Key Vault pojawił się następujący komunikat:

> *Operacja nie jest dozwolona przez kontrolę RBAC. Jeśli niedawno zmieniono przypisania ról, poczekaj kilka minut, aby te przypisania zaczęły obowiązywać.*

Błąd ten oznacza, że użytkownik nie posiada wymaganych uprawnień RBAC do operacji na sekretach w Key Vault. Mimo to użytkownik miał przypisaną rolę Owner dla całej subskrypcji | czyli najwyższy poziom uprawnień administracyjnych.

5.3 Podjęte próby rozwiązania

Podjęto kilka działań w celu usunięcia błędu:

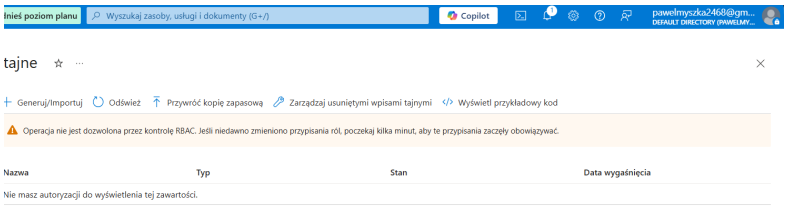
- odczekano ponad kilkanaście minut na propagację uprawnień RBAC,
- zweryfikowano, że Key Vault działa w trybie RBAC (nie Access Policies),
- ponownie zalogowano się do portalu Azure,
- sprawdzono uprawnienia na poziomie zasobu, grupy zasobów oraz subskrypcji.

5.4 Rezultat

Pomimo wykonania wszystkich powyższych kroków próba dodania nowego wpisu tajnego kończyła się komunikatem błedu RBAC.

5.5 Zrzut ekranu błędu

Na rysunku 2 przedstawiono zrzut ekranu potwierdzający wystąpienie błędu.



Rysunek 2: Błąd RBAC podczas próby dodania sekretu do Azure Key Vault mimo pełnych uprawnień

6 Zadanie 6 – Aplikacja WWW + API w chmurze

6.1 Cel

Celem zadania było stworzenie prostej aplikacji webowej, w której frontend (HTML/JavaScript) komunikuje się z API zrealizowanym w Azure Functions. Aplikacja umożliwia wysyłanie plików, zapisywanie ich oraz wyświetlanie listy przesłanych plików.

6.2 Architektura rozwiązania

System został podzielony na dwa podstawowe komponenty:

- Frontend { statyczna aplikacja HTML/JavaScript hostowana lokalnie (lub w Azure Static Web Apps).
- API { Azure Functions (Python), udostępniające dwa endpointy: przesyłanie pliku oraz pobieranie listy plików.

Komunikacja odbywała się w formacie JSON z zawartością pliku kodowaną w base64.

6.3 Tworzenie i konfiguracja projektu Azure Functions

Utworzono nowy projekt:

```
func init ApiApp --worker-runtime python
cd ApiApp
```

Dodano dwie funkcje HTTP Trigger:

```
func new --name UploadFile --template "HTTP trigger" --authlevel "anonymous"
func new --name GetFiles --template "HTTP trigger" --authlevel "anonymous"
```

Zainstalowano wymagane biblioteki:

```
pip install azure-functions azure-storage-blob
```

6.4 Implementacja API

6.4.1 Endpoint UploadFile

Funkcja odbiera nazwę pliku oraz zawartość zakodowaną jako base64. Po dekodowaniu dane są zapisywane w pamięci jako obiekt metadanych. Wprowadzono obsługę CORS (w tym preflight requests).

Fragment implementacji:

```
        @app.route(route="UploadFile", methods=["POST", "
            OPTIONS"],
        auth_level=func.AuthLevel.ANONYMOUS)
        def UploadFile(req: func.HttpRequest) -> func.
            HttpResponse:
        if req.method == "OPTIONS":
        response = func.HttpResponse("", status_code=200)
        response.headers['Access-Control-Allow-Origin'] =
            '*'
        response.headers['Access-Control-Allow-Methods'] = '
            POST, OPTIONS'
        response.headers['Access-Control-Allow-Headers'] = '
            Content-Type'
        return response

        body = req.get_json()
        filename = body.get("filename", "unknown")
        content_b64 = body.get("content", "")

        if not content_b64:
        return func.HttpResponse(
        json.dumps({"error": "Brak zawartoŻci pliku"}),
        mimetype="application/json",
        status_code=400
        )

        file_content = base64.b64decode(content_b64)
        uploaded_files.append({
                "filename": filename,
                "size": len(file_content),
                "uploaded_at": datetime.utcnow().isoformat()
        })

        response = func.HttpResponse(
```

```
        json.dumps({"success": True, "filename": filename}),
        mimetype="application/json"
        )
        response.headers["Access-Control-Allow-Origin"] =
            "*"
        return response
```

6.4.2 Endpoint GetFiles

Zwraca listę wszystkich przesłanych plików wraz z metadanymi:

```
        @app.route(route="GetFiles", methods=["GET", "
            OPTIONS"],
        auth_level=func.AuthLevel.ANONYMOUS)
        def GetFiles(req: func.HttpRequest) -> func.
            HttpResponse:
        if req.method == "OPTIONS":
        response = func.HttpResponse("", status_code=200)
        response.headers['Access-Control-Allow-Origin'] =
            '*'
        return response

        data = {
                "success": True,
                "total": len(uploaded_files),
                "files": uploaded_files
        }

        response = func.HttpResponse(json.dumps(data),
        mimetype="application/json")
        response.headers["Access-Control-Allow-Origin"] =
            "*"
        return response
```

6.5 Frontend HTML/JavaScript

Interfejs użytkownika został zrealizowany jako prosta aplikacja HTML/JS. Umożliwia wybór pliku, wysyłanie go do API oraz wyświetlanie listy przesłanych elementów.

6.5.1 Logika wysyłania pliku

```
        async function uploadFile() {
                const file = fileInput.files[0];
                const reader = new FileReader();

                reader.onload = async function(e) {
```

```
                        const base64 = e.target.result.split
                            (",")[1];

                        await fetch(`${API}/UploadFile`, {
                                method: "POST",
                                headers: {"Content-Type": "
                                    application/json"},
                                body: JSON.stringify({
                                        filename: file.name,
                                        content: base64
                                })
                        });

                        loadFiles();
                };

                reader.readAsDataURL(file);
        }
```

6.5.2 Pobieranie listy plików

```
        async function loadFiles() {
                const response = await fetch(`${API}/
                    GetFiles`);
                const data = await response.json();

                fileList.innerHTML = "";
                data.files.forEach(f => {
                        const li = document.createElement("
                            li");
                        li.textContent = `${f.filename} (${f
                            .size} B)`;
                        fileList.appendChild(li);
                });
        }
```

6.6 Uruchomienie lokalne

6.6.1 API

```
func host start --port 7072
```

6.6.2 Frontend

```
python -m http.server 8000
```

Frontend dostępny był pod adresem:

http://localhost:8000

6.7 Wyniki testów

Wykonano dwa testy:

- przesyłanie pliku (plik poprawnie zakodowany, wysłany i zapisany),
- pobieranie listy plików (zwrócona lista z poprawną liczbą pozycji i metadanymi).

Komunikacja między frontendem a API wymagała konfiguracji CORS, w tym obsługi zapytań typu OPTIONS.

7 Zadanie 7 – Kolejki – przetwarzanie asynchroniczne

7.1 Cel

Stworzenie systemu asynchronicznego przetwarzania wiadomości przy użyciu Azure Storage Queue. Aplikacja wysyła wiadomości do kolejki, QueueTrigger automatycznie je przetwarza w tle.

7.2 Architektura

1. POST /api/SendToQueue -- wysyła wiadomość do Azure Queue (z7-tasks),
2. QueueTrigger automatycznie odbiera i przetwarza wiadomość,
3. dane zapisywane do SQLite z timestampem,
4. GET /api/GetQueueTasks -- lista przetworzonych wiadomości.

7.3 Konfiguracja

- Storage Account: z6storage (West Europe)
- Kolejka: z7-tasks
- Connection: QueueStorageConnection (app settings)

Listing 14: Inicjalizacja

```
from azure.storage.queue import QueueServiceClient

QUEUE_CONNECTION_STRING = os.getenv("
    QueueStorageConnection", "")
QUEUE_NAME = "z7-tasks"
```

7.4 Endpoint SendToQueue – Wysłanie wiadomości do kolejki

```
Metoda: POST
URL: https://z6functions.azurewebsites.net/api/SendToQueue
Status: 200 OK
Usługa: Azure Storage Queue (z7-tasks)
```

7.4.1 Request

Listing 15: SendToQueue request

```
POST /api/SendToQueue HTTP/1.1
Host: z6functions.azurewebsites.net
Content-Type: application/json

{
        "message": "Final Azure Test",
        "task_type": "report_generation",
        "priority": "high"
}
```

7.4.2 Response

Listing 16: SendToQueue response (200 OK)

```
{
        "success": true,
        "message": "Task queued successfully",
        "task_id": "a917168f-0921-4233-b6e8-53
           f78fd48e11",
        "task_type": "report_generation",
        "priority": "high",
        "queued_at": "2025-12-07T15:16:10.123456"
}
```

7.4.3 Implementacja

Listing 17: Funkcja SendToQueue

```
@app.route(route="SendToQueue", methods=["POST"],
auth_level=func.AuthLevel.ANONYMOUS)
def send_to_queue(req: func.HttpRequest) -> func.
    HttpResponse:
"""
␣␣␣␣␣␣␣␣Endpoint␣do␣wysyĆania␣wiadomoŻci␣do␣Azure␣Storage␣
    Queue.
```

```
␣␣␣␣␣␣␣␣Body:␣{"message":␣"content",␣"task_type":␣"process",
    ␣"priority":␣"high"}
␣␣␣␣␣␣␣␣"""
        try:
        req_body = req.get_json()

        if 'message' not in req_body:
        return func.HttpResponse(
        json.dumps({"error": "Missing␣'message'␣field"}),
        mimetype="application/json",
        status_code=400
        )

        # Przygotuj strukturĘ wiadomoŻci
        queue_message = {
                "id": str(uuid.uuid4()),
                "message": req_body['message'],
                "task_type": req_body.get('task_type', '
                    generic_task'),
                "priority": req_body.get('priority', 'normal
                    '),
                "created_at": datetime.utcnow().isoformat(),
                "status": "queued"
        }

        # WyŻlij do Azure Storage Queue
        logging.info(f'[QUEUE]␣Sending␣message:␣{
            queue_message["id"]}')
        queue_service_client = QueueServiceClient.
            from_connection_string(
        QUEUE_CONNECTION_STRING
        )
        queue_client = queue_service_client.get_queue_client
            (QUEUE_NAME)
        queue_client.send_message(json.dumps(queue_message))

        logging.info(f'[QUEUE]␣Message␣queued:␣{
            queue_message["id"]}')

        response = {
                "success": True,
                "message": "Task␣queued␣successfully",
                "task_id": queue_message["id"],
                "task_type": queue_message["task_type"],
                "priority": queue_message["priority"],
                "queued_at": queue_message["created_at"]
        }

        return func.HttpResponse(
        json.dumps(response),
```

```
        mimetype="application/json",
        status_code=200
        )

        except Exception as e:
        logging.error(f'[QUEUE] Error: {str(e)}')
        return func.HttpResponse(
        json.dumps({"error": str(e)}),
        mimetype="application/json",
        status_code=500
        )
```

7.5 QueueTrigger – Automatyczne przetwarzanie

Trigger type: Azure Queue Storage Trigger
Nazwa funkcji: process_queue_message
Wyzwalacz: Automatycznie gdy pojawi się nowa wiadomość w z7-tasks
QueueTrigger wykonuje się asynchronicznie w tle, bez żadnego ręcznego interwencji. Każda wiadomość jest przetwarzana oddzielnie.

7.5.1 Implementacja

Listing 18: QueueTrigger process_queue_message

```
        @app.queue_trigger(
        arg_name="msg",
        queue_name="z7-tasks",
        connection="QueueStorageConnection"
        )
        def process_queue_message(msg: func.QueueMessage):
        """
        Trigger odbiera wiadomo Ż z Azure Queue Storage i
    przetwarza j .
        Uruchamia si Ź automatycznie ka dorazowo gdy pojawi
    si Ź wiadomo Ż .
        """
        try:
        # Odczytaj zawarto Ż wiadomo Żci
        message_json = msg.get_body()
        logging.info(f'[TRIGGER] Processing: {msg.id}')

        # Parsuj JSON
        message_data = json.loads(message_json)

        task_id = message_data.get('id')
        task_type = message_data.get('task_type', 'unknown')
        message_content = message_data.get('message')
        priority = message_data.get('priority')
```

```
        # Zapisz do bazy danych
        conn = sqlite3.connect(DB_PATH)
        cursor = conn.cursor()

        # Utw rz tabel Ź je Żli nie istnieje
        cursor.execute('''
␣␣␣␣␣␣␣␣CREATE␣TABLE␣IF␣NOT␣EXISTS␣queue_tasks␣(
␣␣␣␣␣␣␣␣task_id␣TEXT␣PRIMARY␣KEY,
␣␣␣␣␣␣␣␣message␣TEXT␣NOT␣NULL,
␣␣␣␣␣␣␣␣task_type␣TEXT,
␣␣␣␣␣␣␣␣priority␣TEXT,
␣␣␣␣␣␣␣␣status␣TEXT,
␣␣␣␣␣␣␣␣created_at␣TEXT,
␣␣␣␣␣␣␣␣processed_at␣TEXT
␣␣␣␣␣␣␣␣)
␣␣␣␣␣␣␣␣''')

        # Zapisz zadanie
        processed_at = datetime.utcnow().isoformat()
        cursor.execute('''
␣␣␣␣␣␣␣␣INSERT␣OR␣REPLACE␣INTO␣queue_tasks
␣␣␣␣␣␣␣␣(task_id,␣message,␣task_type,␣priority,␣status,␣
    created_at,␣processed_at)
␣␣␣␣␣␣␣␣VALUES␣(?,␣?,␣?,␣?,␣?,␣?,␣?)
␣␣␣␣␣␣␣␣''', (task_id, message_content, task_type, priority,
     'processed',
        message_data.get('created_at'), processed_at))

        conn.commit()
        conn.close()

        logging.info(f'[TRIGGER]␣Processed:␣{task_id}␣({
            task_type})')

        except json.JSONDecodeError as e:
        logging.error(f'[TRIGGER]␣JSON␣error:␣{str(e)}')
        except Exception as e:
        logging.error(f'[TRIGGER]␣Error:␣{str(e)}')
```

7.6 Endpoint GetQueueTasks – Pobieranie przetworzonych wiadomości

Metoda: GET
URL: https://z6functions.azurewebsites.net/api/GetQueueTasks
Status: 200 OK

7.6.1 Request

Listing 19: GetQueueTasks request

```
GET /api/GetQueueTasks HTTP/1.1
Host: z6functions.azurewebsites.net
```

7.6.2 Response

Listing 20: GetQueueTasks response


```
{
        "success": true,
        "total": 3,
        "tasks": [
        {
                "task_id": "a917168f-0921-4233-b6e8
                    -53f78fd48e11",
                "message": "Final Azure Test",
                "task_type": "report_generation",
                "priority": "high",
                "status": "processed",
                "created_at": "2025-12-07T15
                    :16:10.123456",
                "processed_at": "2025-12-07T15
                    :16:22.654321"
        },
        {
                "task_id": "b217168f-0921-4233-b6e8
                    -53f78fd48e12",
                "message": "Background data sync",
                "task_type": "sync",
                "priority": "normal",
                "status": "processed",
                "created_at": "2025-12-07T15
                    :10:05.111111",
                "processed_at": "2025-12-07T15
                    :10:12.222222"
        }
        ]
}
```


7.7 Endpoint DebugQueue – Podgląd kolejki

Metoda: GET
URL: https://z6functions.azurewebsites.net/api/DebugQueue
Status: 200 OK

Endpoint umożliwia podgląd stanu Azure Storage Queue bez usuwania wiadomości (peek operation):

Listing 21: DebugQueue response

```
{
        "success": true,
        "queue_name": "z7-tasks",
        "message_count": 0,
        "messages_peeked": 0,
        "peeked_messages": [],
        "status": "Queue is empty"
}
```

Jeśli są wiadomości w kolejce:

Listing 22: DebugQueue response with messages

```
{
        "success": true,
        "queue_name": "z7-tasks",
        "message_count": 2,
        "messages_peeked": 2,
        "peeked_messages": [
        {
                "message_id": "uuid-1234",
                "content": {"id": "...", "message":
                    "Test message", "status": "queued
                    "}
        },
        {
                "message_id": "uuid-5678",
                "content": {"id": "...", "message":
                    "Another task", "status": "queued
                    "}
        }
        ]
}
```

7.8 Endpoint ClearPoisonQueue – Czyszczenie poison queue

Metoda: POST
URL: https://z6functions.azurewebsites.net/api/ClearPoisonQueue
Status: 200 OK

Poison queue jest tworzona automatycznie przez Azure Functions dla wiadomości, które nie mogły być przetworzone. Ten endpoint ją czysty:

Listing 23: ClearPoisonQueue response

```
        {
                "success": true,
                "message": "Poison queue cleared",
                "messages_deleted": 0,
                "poison_queue_name": "z7-tasks-poison"
        }
```

7.9 Testy i walidacja

7.9.1 Test 1: Wysłanie wiadomości

```
POST https://z6functions.azurewebsites.net/api/SendToQueue
Content-Type: application/json

{
 "message": "Final Azure Test",
 "task_type": "report_generation",
 "priority": "high"
}

Status: 200 OK
Response:
{
 "success": true,
 "task_id": "a917168f-0921-4233-b6e8-53f78fd48e11",
 "queued_at": "2025-12-07T15:16:10.123456"
}
```

7.9.2 Test 2: Oczekiwanie na przetworzenie

```
(Czekamy 3-5 sekund aby QueueTrigger przetworzył wiadomość)

Azure Portal Logs → GetQueueTasks
Status: 200 OK

[QUEUE] Sending message: a917168f-0921-4233-b6e8-53f78fd48e11
[TRIGGER] Processing: ...
[TRIGGER] Processed: a917168f-0921-4233-b6e8-53f78fd48e11 (report_generation)
```

7.9.3 Test 3: Pobranie przetworzonych wiadomości

```
GET https://z6functions.azurewebsites.net/api/GetQueueTasks
Status: 200 OK

Response:
{
```

```
 "success": true,
 "total": 1,
 "tasks": [
 {
  "task_id": "a917168f-0921-4233-b6e8-53f78fd48e11",
  "message": "Final Azure Test",
  "status": "processed",
  "processed_at": "2025-12-07T15:16:22.654321"
 }
 ]
}
```


```
Wiadomość przetworzona pomyślnie a dane zostały zaspiane do db.
```

7.9.4 Test 4: Sprawdzenie stanu kolejki

```
GET https://z6functions.azurewebsites.net/api/DebugQueue
Status: 200 OK

Response:
{
 "queue_name": "z7-tasks",
 "message_count": 0,
 "status": "Queue is empty"
}

Wszystkie wiadomości przetworzone
```