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

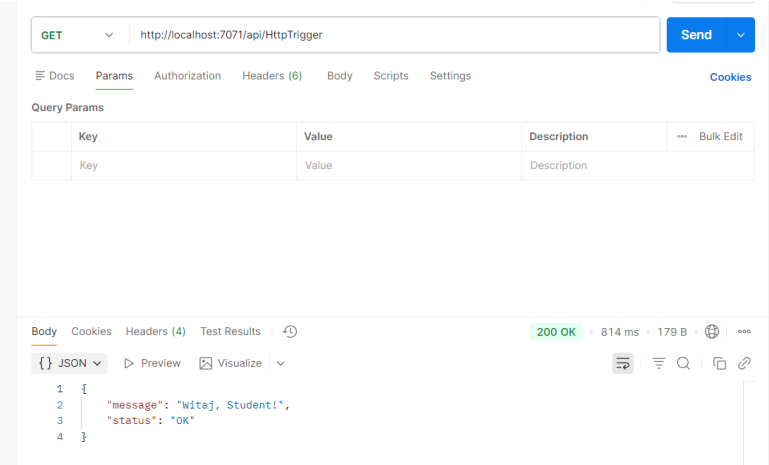
```
import azure.functions as func
import json

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, i zwraca szczegółowe informacje o przesłanym pliku w formacie JSON.

2.2 Tworzenie projektu

1. Rozszerzono istniejący projekt Azure Functions o nowe funkcje.
2. Zainstalowano dodatkowe pakiety:

   ```
   pip install azure-storage-blob azure-data-tables
   ```

3. Zaktualizowano `requirements.txt` z nowymi zależnościami.

2.3 Dodanie HTTP Trigger do przesyłania plików

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

2.4 Implementacja funkcji `upload_file`

Listing 1: Funkcja do przesyłania plików

```
        import azure.functions as func
        import json
        import base64
        from datetime import datetime
        import logging

        app = func.FunctionApp()

        @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')

        blob_name = f"{datetime.utcnow().strftime('%Y%m%d_%H
            %M%S')}_{filename}"

        response = {
                "success": True,
                "message": "File␣received␣and␣processed",
                "filename": filename,
                "blob_name": blob_name,
                "file_size": len(file_content),
                "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
        )
```

2.5 Konfiguracja – `local.settings.json`

Listing 2: Plik konfiguracyjny

```
        {
                "IsEncrypted": false,
                "Values": {
                        "AzureWebJobsStorage": "
                            DefaultEndpointsProtocol=https;
                            AccountName=...",
                        "FUNCTIONS_WORKER_RUNTIME": "python
                            ",
                        "AzureWebJobsSecretStorageType": "
                            Files"
                }
        }
```

2.6 Wymagane pakiety – `requirements.txt`

Listing 3: Zależności projektu

```
        azure-functions==1.24.0
        azure-storage-blob==12.27.1
        azure-data-tables==12.7.0
```

2.7 Uruchomienie i testowanie

Funkcję uruchomiono lokalnie:

```
func host start
```

Endpoint był dostępny pod adresem: `http://localhost:7071/api/UploadFile`

2.8 Testowanie w Postmanie

2.8.1 Request POST – Upload pliku

Metoda: POST
URL: `http://localhost:7071/api/UploadFile`
Headers: Content-Type: application/json
Body (raw JSON):

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

Odpowiedź (200 OK):

```
{
        "success": true,
        "message": "File received and processed",
        "filename": "test.txt",
        "blob_name": "20251204_182815_test.txt",
        "file_size": 20,
        "timestamp": "2025-12-04T18:28:15.888956"
}
```

3 Zadanie 3 – Harmonogramowane zadania (Timer Trigger)

3.1 Cel

Celem zadania było utworzenie funkcji uruchamianej periodycznie co minutę. Funkcja zapisuje timestamp do logów, które są dostępne przez dedykowany endpoint HTTP.

3.2 Implementacja Timer Trigger

Timer Trigger w Azure Functions jest obsługiwany poprzez dekorator `@app.timer_trigger`. Funkcja uruchamia się automatycznie na podstawie harmonogramu w formacie CRON.

Listing 4: Implementacja Timer Trigger

```
from threading import Timer
from datetime import datetime
import logging

# Zmienne globalne do trackowania
scheduled_logs = []
max_logs = 100
```

```
        def log_scheduled_execution():
        """
        Funkcja do logowania wykona  - uruchamiana
            periodycznie co minut Ź.
        """
        global scheduled_logs

        timestamp = datetime.utcnow().isoformat()
        log_entry = {
                "executed_at": timestamp,
                "task_name": "scheduled_task",
                "status": "success"
        }

        logging.info(f'[TIMER TRIGGER] Executed at: {
            timestamp}')

        # Dodaj do listy (ogranicz do max_logs)
        scheduled_logs.append(log_entry)
        if len(scheduled_logs) > max_logs:
        scheduled_logs.pop(0)

        # Zaplanuj nast Źpne wykonanie za 60 sekund
        timer = Timer(60.0, log_scheduled_execution)
        timer.daemon = True
        timer.start()

        # Uruchom Timer na starcie aplikacji
        log_scheduled_execution()
```

3.3 Endpoint do przeglądu logów Timer Trigger

Stworzono dedykowany endpoint do wyświetlania logów wykonań Timer Trigger:

Listing 5: Endpoint GetTimerLogs

```
        @app.route(route="TimerLogs", methods=["GET"])
        def get_timer_logs(req: func.HttpRequest) -> func.
            HttpResponse:
        """
        Endpoint do przegl du log w z Timer Trigger.
        """
        try:
        response = {
                "total_executions": len(scheduled_logs),
                "logs": scheduled_logs[-20:] if
                    scheduled_logs else [],
                "message": "Timer Trigger uruchamia si Ź co
                    minut Ź"
```

```
        }
        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 Uruchomienie i testowanie

Endpoint dostępny jest pod adresem:

```
GET http://localhost:7071/api/TimerLogs
```

3.4.1 Testowanie – zaraz po starcie

Request:

```
GET http://localhost:7071/api/TimerLogs
```

Odpowiedź:

```
        {
                "total_executions": 1,
                "logs": [
                {
                        "executed_at": "2025-12-04T18
                            :40:27.176746",
                        "task_name": "scheduled_task",
                        "status": "success"
                }
                ],
                "message": "Timer Trigger uruchamia si Ź co
                    minut Ź"
        }
```

3.4.2 Testowanie – po 60 sekundach

Request:

```
GET http://localhost:7071/api/TimerLogs
```

Odpowiedź:

```
{
        "total_executions": 2,
        "logs": [
        {
                "executed_at": "2025-12-04T18
                    :40:27.176746",
                "task_name": "scheduled_task",
                "status": "success"
        },
        {
                "executed_at": "2025-12-04T18
                    :41:27.188614",
                "task_name": "scheduled_task",
                "status": "success"
        }
        ],
        "message": "Timer Trigger uruchamia si Ź co
            minut Ź"
}
```

Timer Trigger uruchamia się co minutę, logując każde wykonanie z timestamp. Endpoint `/api/TimerLogs` pozwala monitorować historię wykonań w czasie rzeczywistym.

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 w bazie danych. System wykorzystuje SQLite do przechowywania danych lokalnie oraz udostępnia RESTful API do zarządzania produktami.

4.2 Konfiguracja bazy danych

4.2.1 Inicjalizacja SQLite

Baza danych SQLite jest automatycznie inicjalizowana przy starcie aplikacji:

Listing 6: Inicjalizacja bazy danych

```
import sqlite3
import uuid
import logging
from datetime import datetime

DB_PATH = "data.db"
```

```
def init_database():
"""
Inicjalizacja bazy danych SQLite.
"""
try:
conn = sqlite3.connect(DB_PATH)
cursor = conn.cursor()

cursor.execute('''
CREATE TABLE IF NOT EXISTS products (
id TEXT PRIMARY KEY,
name TEXT NOT NULL,
description TEXT,
price REAL NOT NULL,
quantity INTEGER,
created_at TEXT,
updated_at TEXT
)
''')

conn.commit()
conn.close()
logging.info('[DATABASE] Initialized SQLite database
    ')
except Exception as e:
logging.error(f'[DATABASE] Error initializing
    database: {str(e)}')

# Inicjalizuj baz Ź na starcie
init_database()
```

4.3 Endpoint SaveProduct – Zapis produktu

Metoda: POST
URL: `http://localhost:7071/api/SaveProduct`
Status: 201 Created

Request Body:

```
{
        "name": "Laptop Dell",
        "description": "High-performance laptop",
        "price": 1299.99,
        "quantity": 5
}
```

Response:

```
{
        "success": true,
```

```
            "message": "Product saved successfully",
            "product_id": "62a980b4-78b7-4fee-a967-
                eaffbfecea0b",
            "name": "Laptop Dell",
            "price": 1299.99,
            "quantity": 5,
            "created_at": "2025-12-04T18:48:46.009515"
        }
```

4.3.1 Implementacja

Listing 7: Funkcja SaveProduct

```
        @app.route(route="SaveProduct", methods=["POST"])
        def save_product(req: func.HttpRequest) -> func.
            HttpResponse:
        """
        Endpoint do zapisywania produktu do bazy danych.
        Body: {"name": "...", "description": "...", "price":
             99.99, "quantity": 10}
        """
        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 required fields: name,
             price"}),
        mimetype="application/json",
        status_code=400
        )

        product_id = str(uuid.uuid4())
        name = req_body['name']
        description = req_body.get('description', '')
        price = float(req_body['price'])
        quantity = int(req_body.get('quantity', 0))
        now = datetime.utcnow().isoformat()

        conn = sqlite3.connect(DB_PATH)
        cursor = conn.cursor()

        cursor.execute('''
        INSERT INTO products (id, name, description, price,
            quantity, created_at, updated_at)
        VALUES (?, ?, ?, ?, ?, ?, ?)
        ''', (product_id, name, description, price, quantity
            , now, now))
```

```
        conn.commit()
        conn.close()

        response = {
                "success": True,
                "message": "Product saved successfully",
                "product_id": product_id,
                "name": name,
                "price": price,
                "quantity": quantity,
                "created_at": now
        }

        return func.HttpResponse(
        json.dumps(response),
        mimetype="application/json",
        status_code=201
        )

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

4.4 Endpoint GetProducts – Pobieranie wszystkich produktów

Metoda: GET
URL: `http://localhost:7071/api/GetProducts`
Status: 200 OK
Response:

```
        {
                "success": true,
                "total": 1,
                "products": [
                {
                        "id": "62a980b4-78b7-4fee-a967-
                            eaffbfecea0b",
                        "name": "Laptop Dell",
                        "description": "High-performance
                            laptop",
                        "price": 1299.99,
                        "quantity": 5,
```

```
                        "created_at": "2025-12-04T18
                            :48:46.009515",
                        "updated_at": "2025-12-04T18
                            :48:46.009515"
                }
                ]
        }
```

4.5 Endpoint GetProduct – Pobieranie produktu po ID

Metoda: GET
URL: `http://localhost:7071/api/GetProduct?id=62a980b4-78b7-4fee-a967-eaffbfecea0b`
Status: 200 OK lub 404 Not Found

Response (200):

```
        {
                "success": true,
                "product": {
                        "id": "62a980b4-78b7-4fee-a967-
                            eaffbfecea0b",
                        "name": "Laptop Dell",
                        "description": "High-performance
                            laptop",
                        "price": 1299.99,
                        "quantity": 5,
                        "created_at": "2025-12-04T18
                            :48:46.009515",
                        "updated_at": "2025-12-04T18
                            :48:46.009515"
                }
        }
```

4.6 Endpoint UpdateProduct – Aktualizacja produktu

Metoda: PUT
URL: `http://localhost:7071/api/UpdateProduct`
Status: 200 OK

Request Body:

```
        {
                "id": "62a980b4-78b7-4fee-a967-eaffbfecea0b
                    ",
                "name": "Laptop Dell XPS",
                "price": 1399.99,
                "quantity": 3
        }
```

Response:

```
{
        "success": true,
        "message": "Product updated successfully",
        "product_id": "62a980b4-78b7-4fee-a967-
            eaffbfecea0b",
        "updated_at": "2025-12-04T18:50:00.123456"
}
```

4.7 Endpoint DeleteProduct – Usuwanie produktu

Metoda: DELETE
URL: `http://localhost:7071/api/DeleteProduct?id=62a980b4-78b7-4fee-a967-eaffbfecea0b`
Status: 200 OK lub 404 Not Found

Response:

```
{
        "success": true,
        "message": "Product deleted successfully",
        "product_id": "62a980b4-78b7-4fee-a967-
            eaffbfecea0b"
}
```

4.8 Schemat bazy danych

Tabela `products`:

Kolumna	Typ	Opis
id	TEXT (PRIMARY KEY)	Unikatowy identyfikator produktu (UUID)
name	TEXT	Nazwa produktu
description	TEXT	Opis produktu
price	REAL	Cena produktu
quantity	INTEGER	Ilość sztuk
created_at	TEXT	Timestamp utworzenia (ISO format)
updated_at	TEXT	Timestamp ostatniej aktualizacji (ISO format)

4.9 Testy – Przykładowe operacje

4.9.1 Test 1: Zapis produktu

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

{
 "name": "Laptop Dell",
 "description": "High-performance laptop",
```

```
 "price": 1299.99,
 "quantity": 5
}

Status: 201 Created
Product ID: 62a980b4-78b7-4fee-a967-eaffbfecea0b
```

4.9.2 Test 2: Pobierz wszystkie produkty

```
GET /api/GetProducts

Status: 200 OK
Total products: 1
```

4.9.3 Test 3: Pobierz konkretny produkt

```
GET /api/GetProduct?id=62a980b4-78b7-4fee-a967-eaffbfecea0b

Status: 200 OK
```

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:

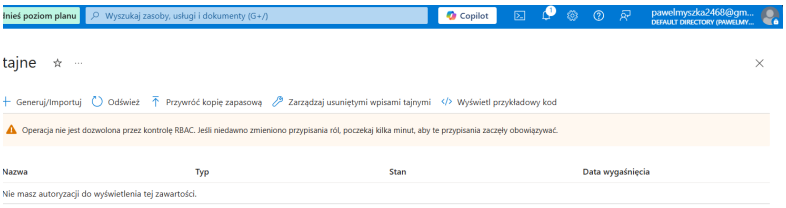
- odczekano ponad kilkanaście minut na propagację uprawnień RBAC,
- zweryfikowano, że Key Vault działa w trybie RBAC (nie Access Policies),
- ponownie zalogowano się do portalu Azure,
- sprawdzono uprawnienia na poziomie zasobu, grupy zasobów oraz subskrypcji.

5.4 Rezultat

Pomimo wykonania wszystkich powyższych kroków próba dodania nowego wpisu tajnego kończyła się komunikatem błędu 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 zawarto Żci 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

Celem zadania było stworzenie systemu asynchronicznego przetwarzania wiadomości przy użyciu Azure Storage Queue. System obejmuje:

- HTTP endpoint do wysyłania wiadomości do kolejki,
- QueueTrigger do automatycznego przetwarzania wiadomości,
- bazę danych SQLite do przechowywania przetworzonych wiadomości,
- logi potwierdzające przetworzenie każdej wiadomości.

7.2 Architektura

Przepływ systemu:

1. Aplikacja wysyła POST request do `/api/SendToQueue`,
2. wiadomość trafia do kolejki `z7-tasks`,

3. QueueTrigger odbiera i przetwarza wiadomość,
4. dane zapisywane są do SQLite (`queue_tasks`),
5. logi rejestrują przetworzenie z timestampem.

7.3 Konfiguracja Azure Storage Queue

- Storage Account: `z6storage`
- Główna kolejka: `z7-tasks`
- Poison queue: `z7-tasks-poison` (dla wiadomości nieprzetworzonych)

Connection string jest przechowywany w `local.settings.json` oraz w Azure Function App settings jako `QueueStorageConnection`.

7.4 Baza danych SQLite

Tabela `queue_tasks` przechowuje przetworzone wiadomości:

```
CREATE TABLE IF NOT EXISTS queue_tasks (
task_id TEXT PRIMARY KEY,
message TEXT,
task_type TEXT,
priority TEXT,
status TEXT,
created_at TEXT,
processed_at TEXT
)
```

7.5 Endpoint SendToQueue

POST /api/SendToQueue – wysyła wiadomości do kolejki. Body JSON:

```
{
        "message": "Tre Ż  zadania",
        "task_type": "demo",
        "priority": "high"
}
```

Odpowiedź:

```
{
        "success": true,
        "task_id": "uuid",
        "task_type": "demo",
        "priority": "high",
        "queued_at": "timestamp"
}
```

7.6 QueueTrigger – process_queue_message

Trigger odbiera wiadomość z kolejki, dekoduje JSON, zapisuje do bazy i loguje przetworzenie. Status wiadomości: `processed` lub `failed`.

7.7 Endpoint GetQueueTasks

GET /api/GetQueueTasks – zwraca listę przetworzonych wiadomości:

```
{
        "success": true,
        "total": 2,
        "tasks": [
        {"task_id": "uuid1", "message": "...", "
            status": "processed"},
        {"task_id": "uuid2", "message": "...", "
            status": "processed"}
        ]
}
```

7.8 Endpoint DebugQueue

GET /api/DebugQueue – podgląd kolejki:

- liczba wiadomości w kolejce,
- pierwsze wiadomości bez usuwania (peek).

7.9 Endpoint ClearPoisonQueue

POST /api/ClearPoisonQueue – usuwa wiadomości z poison queue.

7.10 Testowanie

- Wysłanie wiadomości POST /SendToQueue,
- oczekiwanie na QueueTrigger (kilka sekund),
- sprawdzenie stanu kolejki i bazy danych,
- pobranie listy przetworzonych wiadomości GET /GetQueueTasks.

```
task_id: a917168f-0921-4233-b6e8-53f78fd48e11
message: Final Azure Test
status: processed
processed_at: 2025-12-07T15:16:22
```