SECURE 3-TIER PYTHON APPLICATION ON AZURE USING APP SERVICES AND PRIVATE ENDPOINT–ENABLED AZURE SQL

Implementation of a 3-tier architecture featuring an Azure App Service for the UI, a secondary App Service for backend microservices, and an Azure SQL Database configured with Private Endpoints for enhanced security and data integrity.

First: Decide what kind of application stack you are going to deploy either **python, .net, node Js.** I am going with **Python**. Since I am not a developer I will take help of AI tools develop the application based on my requirements.

PYTHON APPLICATION:

1. Create a folder in local named project3 and open it in visual studio code
2. Inside above folder create folder backend-api then below files as guided

backend-api/requirements.txt

```
fastapi==0.110.0
uvicorn==0.27.1
pyodbc==5.1.0
pydantic==2.6.4
email-validator==2.1.1
```

backend-api/main.py

```
import os
import pyodbc
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, EmailStr

app = FastAPI(title="Backend API", version="1.0")

class Submission(BaseModel):
    full_name: str
    email: EmailStr
    message: str | None = None

def get_conn():
    conn_str = os.getenv("SQL_CONN_STR")
    if not conn_str:
        raise RuntimeError("SQL_CONN_STR not set")
    return pyodbc.connect(conn_str)

@app.get("/health")
def health():
    return {"status": "ok"}

@app.post("/submit")
def submit(data: Submission):
    try:
        conn = get_conn()
        cur = conn.cursor()
        cur.execute(
            "INSERT INTO dbo.UserSubmissions (FullName, Email, Message) VALUES (?, ?, ?)",
            data.full_name, data.email, data.message
        )
        conn.commit()
```

```
            return {"status": "saved"}
        except Exception as e:
            raise HTTPException(status_code=500, detail=str(e))
        finally:
            try:
                conn.close()
            except Exception:
                pass
```

3. Now create another folder for UI named ui-flask and do below as guided

ui-flask/requirements.txt

```
Flask==3.0.2
requests==2.31.0
gunicorn==21.2.0
```

ui-flask/app.py

```
import os
import requests
from flask import Flask, render_template, request

app = Flask(__name__)
API_BASE = os.getenv("API_BASE_URL")

@app.get("/")
def index():
    return render_template("form.html", status=None)

@app.post("/submit")
def submit():
    if not API_BASE:
        return render_template("form.html", status="API_BASE_URL not set")

    payload = {
        "full_name": request.form.get("full_name", ""),
        "email": request.form.get("email", ""),
        "message": request.form.get("message", "")
    }

    try:
        r = requests.post(f"{API_BASE}/submit", json=payload, timeout=15)
        if r.status_code == 200:
            return render_template("form.html", status="Saved successfully ✅")
        return render_template("form.html", status=f"Backend error ❌: {r.text}")
    except Exception as e:
        return render_template("form.html", status=f"Request failed ❌: {str(e)}")
```

ui-flask/templates/form.html

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>3-Tier Python App</title>
```

```
</head>
<body style="font-family: Arial; max-width: 520px; margin: 40px auto;">
  <h2>UI Tier - Flask Form</h2>

  <form method="POST" action="/submit">
    <input name="full_name" placeholder="Full Name" required
style="width:100%;padding:10px;margin:8px 0;" />
    <input name="email" placeholder="Email" required style="width:100%;padding:10px;margin:8px 0;" />
    <textarea name="message" placeholder="Message" style="width:100%;padding:10px;margin:8px
0;"></textarea>
    <button type="submit" style="padding:10px 14px;">Submit</button>
  </form>

  {% if status %}
    <div style="margin-top:12px;padding:10px;background:#f3f3f3;">{{ status }}</div>
  {% endif %}
</body>
</html>
```

- Your Python application code is ready with backend frontend and backend and if we submit form it will store in SQL database.

INFRASTRUCTURE:

---

1) Create Resource Group

1. Azure Portal → Search **Resource groups**
2. **Create**
3. Enter:
    - Resource group: rg-3tier-python
    - Region: (your region, ex: Central India)
4. **Create**

2) Create Virtual Network (VNet) + 2 Subnets

1. Portal → Search **Virtual networks** → **Create**
2. Basics:
    - RG: rg-3tier-python
    - Name: vnet-3tier
    - Region: same region
3. IP Addresses:
    - Address space: 10.10.0.0/16
4. Subnets:
    - **Subnet for App Service VNet integration**
        - Name: snet-app
        - Range: 10.10.1.0/24
    - **Subnet for Private Endpoint**
        - Name: snet-pe
        - Range: 10.10.2.0/24
5. **Review + Create → Create**

So what is **APP SERVICE VNET INTEGRATION** here : enables your web app to access resources within or through a virtual network, such as virtual machines, databases, or services secured with **private endpoints.**

3) Create Azure SQL Server

1. Portal → Search **SQL servers** → **Create**
2. Fill:
    - RG: rg-3tier-python
    - Server name: sqlsrv3tier01 (unique)
    - Region: same
    - Authentication: **SQL authentication**
    - Admin: sqladminuser
    - Password: Strong password
3. **Create**

4) Create Azure SQL Database with Private Access (Private Endpoint)

1. In the Azure Portal, search for **SQL databases** and select **Create**.
2. In the **Basics** tab, configure the following:
    - **Resource group**: rg-3tier-python
    - **Database name**: sqldb3tier01
    - **Server**: sqlsrv3tier01
    - **Compute + storage**: Select **Basic / S0** (suitable for demo or POC)
3. Move to the **Networking** tab and configure network access:
    - **Connectivity method**: Select **Private endpoint**
    - **Public network access**: Set to **Disabled**
    - **Allow Azure services and resources to access this server**: Set to **No**
4. Under the **Private endpoints** section, click **+ Add private endpoint** and provide the following details:
    - **Private endpoint name**: pe-sql-3tier
    - **Virtual network**: vnet-3tier
    - **Subnet**: snet-pe (dedicated subnet for private endpoints)
5. Enable **Private DNS integration** and select:
    - **Private DNS zone**: privatelink.database.windows.net
6. Review the configuration and select **Create** to provision the database along with the private endpoint.

Outcome

- Azure SQL Database is assigned a private IP address within the virtual network.
- The DNS name sqlsrv3tier01.database.windows.net resolves to the private IP through the private DNS zone.
- Public access to the database is fully disabled.
- The database is accessible only from resources within the virtual network, such as the backend App Service integrated with the VNet.

SQL Server Networking Configuration (Controlled Access Setup)

Configuration Performed

The following network settings were configured on the Azure SQL Server (sqlsrv3tier01) under **Networking → Public access**:

1. **Public network access** was set to **Selected networks**
2. A **temporary firewall rule** was added to allow the administrator's current client IPv4 address
3. **Allow Azure services and resources to access this server** was enabled
4. No virtual network service endpoint rules were added
5. Private Endpoint configuration was already in place for production access

Why this was done

The Azure SQL Database is configured to use **Private Endpoint only**, so it cannot be accessed from the internet. However, to perform **initial setup tasks** (such as creating tables), temporary access from the administrator's machine was required. A firewall rule was added to allow only a **single trusted IP address**.

5) Create SQL Table (for storing submitted data)

1. Open **SQL Database → Query editor (preview)**

2. Login with SQL admin user
3. Run:

```
CREATE TABLE dbo.UserSubmissions (
  Id INT IDENTITY(1,1) PRIMARY KEY,
  FullName NVARCHAR(100) NOT NULL,
  Email NVARCHAR(150) NOT NULL,
  Message NVARCHAR(500) NULL,
  CreatedAt DATETIME2 NOT NULL DEFAULT SYSDATETIME()
);
```

6) Create App Service Plan (Linux)

1. Portal → Search **App Service plans** → **Create**
2. Enter:
    - RG: rg-3tier-python
    - Name: asp-3tier-python
    - OS: **Linux**
    - Region: same
    - Pricing: B1 (for testing)
3. Create

- If above plan didn't work you need to upgrade the plan

7) Create Backend App Service (FastAPI)

1. Portal → Search **App Services** → **Create**
2. Basics:
    - RG: rg-3tier-python
    - Name: app-api-python-01 (unique)
    - Publish: Code
    - Runtime stack: **Python 3.10/3.11**
    - OS: Linux
    - Plan: asp-3tier-python
3. Create

8) Enable VNet Integration for Backend (required for SQL Private Endpoint)

1. Open **Backend App Service** (app-api-python-01)
2. Go to **Networking→ Outbound traffic configuration**
3. Under **Outbound traffic configuration**→ Click **Add VNet**
4. Select:
    - VNet: vnet-3tier
    - Subnet: snet-app
5. Save/Apply

Now backend can reach SQL privately.

9) Create UI App Service (Flask)

1. Portal → **App Services → Create**
2. Basics:
    - RG: rg-3tier-python
    - Name: app-ui-python-01 (unique)
    - Runtime: Python 3.10/3.11
    - OS: Linux
    - Plan: asp-3tier-python
3. Create

UI does not need VNet integration because it will not access SQL directly.

Add Azure SQL Connection String to Backend App Service

Step 1: Go to Environment variables

1. In the left menu, undesr **Settings**
2. Click **Environment variables**

This is where **Application settings/ Connection strings** live in the new UI.

Step 2: Open App settings tab

1. At the top, click + then
Name: SQL_CONN_STR

Value: DRIVER={ODBC Driver 18 for SQL Server};SERVER=tcp:sqlsrv3tier01.database.windows.net,1433;DATABASE=sqldb3tier01;UID=LakshmiNarayana;PWD=Lakshmi@871545456789$$@;Encrypt=yes;TrustServerCertificate=no;Connection Timeout=30;

Step 3: Open Connection strings tab

1. At the top, click the **Connection strings** tab
2. Click **+ Add**

Name SQL_CONN_STR

Value

Paste the connection string **exactly like this** (no spaces):

DRIVER={ODBC Driver 18 for SQL Server};SERVER=tcp:sqlsrv3tier01.database.windows.net,1433;DATABASE=sqldb3tier01;UID=LakshmiNarayana;PWD=Lakshmi@871545456789$$@;Encrypt=yes;TrustServerCertificate=no;Connection Timeout=30;

CHANGE THE YELLOW COLOR TO YOUR VALUES

Type: SQLAzure

(This tells Azure this is an Azure SQL connection.)

Apply and Save

1. Click **Apply** (bottom of the panel)
2. Click **Save** (top bar, if shown)

Restart the app

How the backend uses this

In your Python code, the backend reads it using:
os.getenv("SQL_CONN_STR")

Force Azure to run build and install requirements

Step 1: Add build setting

Backend App Service → **Environment variables** → **App settings** tab → **+ Add**

Add:

- **Name**: SCM_DO_BUILD_DURING_DEPLOYMENT
- **Value**: 1

Click **Apply** → **Save**

Step 2: Restart

Backend App Service → **Restart**

Backend App Deployment – Sequential Steps (Publish files – New)

Step 1: Prepare files locally

1. Open your local folder **backend-api**

2. Confirm it contains:

main.py
requirements.txt

Step 2: Create ZIP file

1. Select **main.py** and **requirements.txt**
2. Right-click → **Send to → Compressed (zipped) folder**
3. Name the ZIP file:
4. backend-api.zip

Important:

- The ZIP must **not** contain a parent folder
- Files must be at the ZIP root

Step 3: Open Backend App Service

1. Azure Portal → **App Services**
2. Select **app-api-python-01**

Go to Deployment Center

1. In the left menu, click **Deployment Center**
2. Click the **Settings** tab (top)

Select deployment source

1. Under **Source**
2. Choose **Publish files (new)**

Upload ZIP file

1. After saving, click **Browse / Upload**
2. Select **backend-api.zip**
3. Click **Open / Publish**
4. Wait for deployment to complete

Step 7: Verify files in App Service

1. Go to **Advanced Tools (Kudu)** → **Go**
2. Click **site/wwwroot**
3. Confirm files exist:
   main.py
   requirements.txt

Step 8: Set Startup Command

1. Backend App Service → **Configuration**
2. Open **Stack settings**
3. Set **Startup Command**:
   python -m uvicorn main:app --host 0.0.0.0 --port 8000
4. Click **Save**
5. Restart the App Service

Quick verification

After restart, open app URL:
https://app-api-python-01-enheh0haezbsfqb2.centralindia-01.azurewebsites.net/health
If it opens, the backend is running and has loaded the configuration.

BACKEND IS SUCCESFULLY DONE

FRONTEND

Step 1: Set Startup Command

1. Frontnd App Service → **Configuration**
2. Open **Stack settings**

3. Set **Startup Command**:
   gunicorn --bind=0.0.0.0 --timeout 600 app:app
4. Click **Save**

Add API Base URL as Environment Variable (CRITICAL)

UI App Service → **Environment variables**

1. Go to **App settings** tab
2. Click **+ Add**
3. Add:

- **Name:** API_BASE_URL
- **Value:** https://app-api-python-01-enheh0haezbsfqb2.centralindia-01.azurewebsites.net

4. Click **Apply**
5. Click restart from Overview

Add build setting

Front App Service → **Environment variables** → **App settings** tab → **+ Add**

Add:

- **Name**: SCM_DO_BUILD_DURING_DEPLOYMENT
- **Value**: true

Click **Apply** → **Save**

Frontend App Deployment – Sequential Steps (Publish files – New)

Step 1: Prepare files locally

3. Open your local folder **ui-flask**
4. Confirm it contains the required files

Step 2: Create ZIP file

5. Select **templates folder and app.py, requirements.txt**
6. Right-click → **Send to → Compressed (zipped) folder**
7. Name the ZIP file:
8. ui.zip

Step 3: Open Frontend App Service

3. Azure Portal → **App Services**
4. Select **app-ui-python-02**

Go to Deployment Center

3. In the left menu, click **Deployment Center**
4. Click the **Settings** tab (top)

Select deployment source

3. Under **Source**
4. Choose **Publish files (new)**

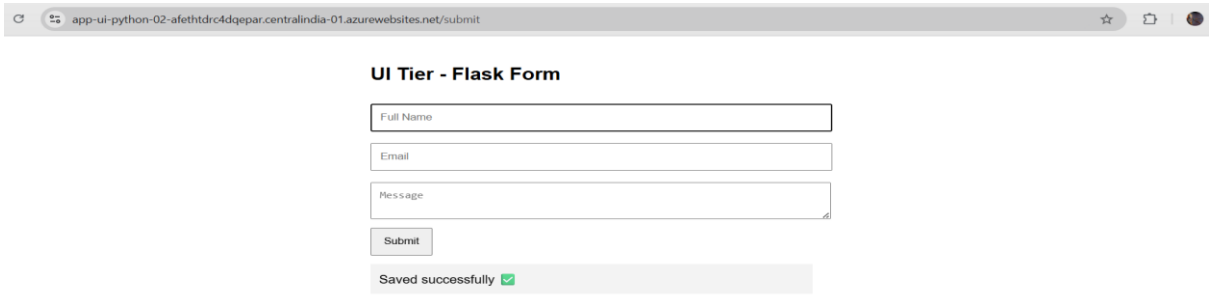
Upload ZIP file

5. After saving, click **Browse / Upload**
6. Select **ui.zip**
7. Click **Open / Publish**
8. Wait for deployment to complete

Step 7: Verify files in App Service

4. Go to **Advanced Tools (Kudu)** → **Go**
5. Click **site/wwwroot**
6. Confirm files exist:
   main.py
   requirements.txt

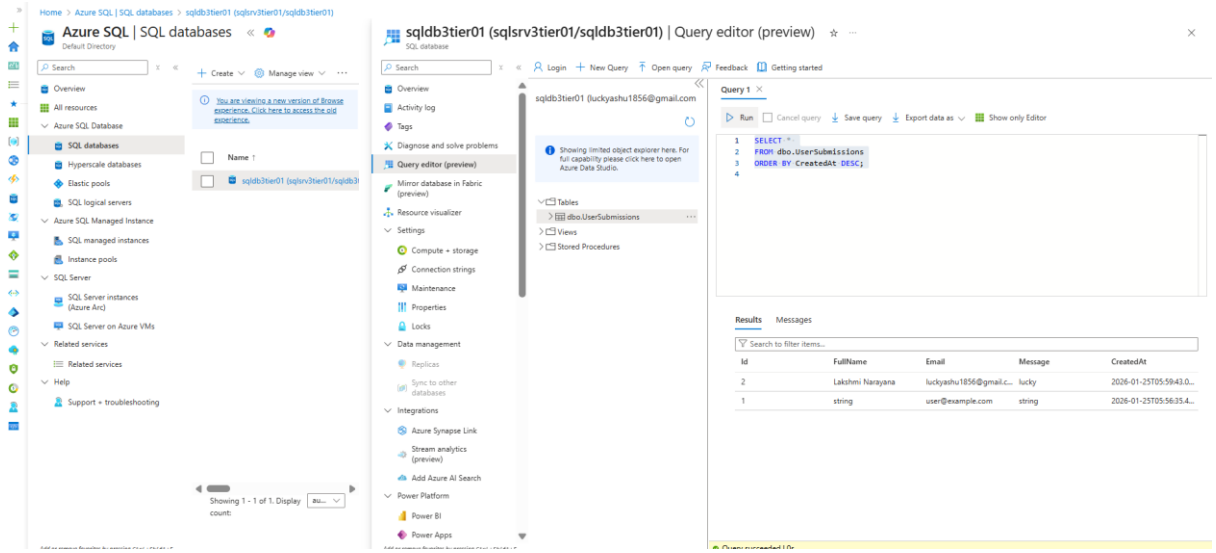
Finally open the frontend app service and submit the form it should store your data In SQL database like below screenshot.



1) Open Query editor

Azure Portal → **SQL database (sqldb3tier01)** → **Query editor (preview)** Login with your SQL user and then run

SELECT *
FROM dbo.UserSubmissions
ORDER BY CreatedAt DESC;



PROJECT END