Setting Up the Development Environment in Visual Studio

Development Environment in Visual Studio

Setting Up the Development Environment in Visual Studio

This guide will walk you through setting up the Dwellingly | AI application development environment in Visual Studio, configuring Azure resources, and preparing for local development and testing.

Prerequisites

- Visual Studio 2022
- Azure CLI
- .NET 8 SDK
- Azure Subscription

Step-by-Step Guide

Step 1: Install Prerequisites

1. **Install Visual Studio 2022:**
   - Download and install Visual Studio 2022 from the Visual Studio website.
   - Select the following workloads during installation:
     - ASP.NET and web development
     - Azure development
2. **Install Azure CLI:**
   - Follow the instructions from the official Azure documentation.
   - Sign in to your Azure account:

     ```
     az login
     ```
3. **Install .NET SDK:**
   - Download and install the .NET 8 SDK from the .NET website.

Step 2: Create the Solution and Projects in Visual Studio

1. **Create a New Solution:**
   - Open Visual Studio 2022.
   - Go to `File -> New -> Project`.
   - Select `Blank Solution`.
   - Name it `Dwellingly`.
   - Choose a location for the solution and click `Create`.
2. **Add Projects to the Solution:**
   - Right-click on the `Dwellingly` solution in the Solution Explorer -> `Add -> New Project`.

     Backend Project:
     - Select `ASP.NET Core Web API`.
     - Name it `Dwellingly.API`.

Frontend Project:

- Select `Blazor WebAssembly App`.
- Name it `Dwellingly.Client`.

Middleware/Business Logic Project:

- Select `Class Library (.NET Standard)`.
- Name it `Dwellingly.Business`.

Data Access Project:

- Select `Class Library (.NET Standard)`.
- Name it `Dwellingly.Data`.

Step 3: Configure Each Project

3.1 Backend Project (`Dwellingly.API`)

1. **Install Required NuGet Packages**:

   - Open the `NuGet Package Manager Console`.
   - Select `Dwellingly.API` as the Default Project.
   - Install the following packages:

   ```
   Install-Package Microsoft.EntityFrameworkCore
   Install-Package Microsoft.EntityFrameworkCore.SqlServer
   Install-Package Dapper
   Install-Package AutoMapper
   Install-Package Microsoft.Data.SqlClient
   ```

2. **Set Up Entity Framework Core**:

   - Add a `DbContext` class in the `Dwellingly.API` project:

   ```
   using Microsoft.EntityFrameworkCore;

   namespace Dwellingly.Data
   {
   ```

```
    public class ApplicationDbContext : DbContext
    {
        public ApplicationDbContext(DbContextOptions<Ap
plicationDbContext> options)
            : base(options)
        {
        }

        public DbSet<Property> Properties { get; set; }
    }

    public class Property
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Address { get; set; }
        public decimal Price { get; set; }
    }
}
```

- Configure the connection string in `appsettings.json`:

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\\\mssqlloca
ldb;Database=DwellinglyDB;Trusted_Connection=True;Multi
pleActiveResultSets=true"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

- Update `Program.cs` to register the `DbContext`:

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }

    public static IHostBuilder CreateHostBuilder(string
[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseStartup<Startup>();
            });
}

public class Startup
{
    public IConfiguration Configuration { get; }

    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public void ConfigureServices(IServiceCollection se
rvices)
    {
        services.AddDbContext<ApplicationDbContext>(opt
ions =>
```

```
            options.UseSqlServer(Configuration.GetConne
ctionString("DefaultConnection")));
        services.AddControllers();
    }

    public void Configure(IApplicationBuilder app, IWeb
HostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Home/Error");
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        app.UseStaticFiles();

        app.UseRouting();

        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
        });
    }
}
```

3.2 Frontend Project (`Dwellingly.Client`)

1. **Install Required NuGet Packages**:

- Open the `NuGet Package Manager Console`.
- Select `Dwellingly.Client` as the Default Project.
- Install the following packages:

```
Install-Package Microsoft.AspNetCore.Components.WebAsse
mbly
Install-Package Blazored.LocalStorage
```

2. **Set Up HttpClient**:

- Update `Program.cs` to configure `HttpClient`:

```
public class Program
{
    public static async Task Main(string[] args)
    {
        var builder = WebAssemblyHostBuilder.CreateDefa
ult(args);
        builder.RootComponents.Add<App>("#app");

        builder.Services.AddScoped(sp => new HttpClient
{ BaseAddress = new Uri(builder.HostEnvironment.BaseAdd
ress) });
        builder.Services.AddBlazoredLocalStorage();

        await builder.Build().RunAsync();
    }
}
```

3.3 Middleware/Business Logic Project (`Dwellingly.Business`)

1. **Create Business Logic Classes**:

- Create necessary services and business logic in this project.
- For example, create a service to manage properties:

```
using Dwellingly.Data;

namespace Dwellingly.Business
{
    public class PropertyService
    {
        private readonly ApplicationDbContext _context;

        public PropertyService(ApplicationDbContext context)
        {
            _context = context;
        }

        public async Task<List<Property>> GetPropertiesAsync()
        {
            return await _context.Properties.ToListAsync();
        }
    }
}
```

3.4 Data Access Project (`Dwellingly.Data`)

1. **Add Data Entities**:

- Create entities and repositories in this project.
- For example, create a `Property` entity:

```
namespace Dwellingly.Data
{
    public class Property
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Address { get; set; }
        public decimal Price { get; set; }
    }
}
```

Step 4: Configure Azure Resources for Development

1. **Log in to Azure using the Azure CLI:**

```
az login
```

2. **Create a Resource Group:**

```
az group create --name DwellinglyDevResourceGroup --locati
on eastus
```

3. **Create an Azure SQL Database and Server:**

```
az sql server create --name dwellingly-dev-sql-server --re
source-group DwellinglyDevResourceGroup --location eastus
--admin-user sqladmin --admin-password YourStrong(!)Passwo
rd

az sql db create --resource-group DwellinglyDevResourceGro
up --server dwellingly-dev-sql-server --name DwellinglyDev
DB --service-objective S0
```

4. **Create Azure Container Registry (ACR):**

```
az acr create --resource-group DwellinglyDevResourceGroup
--name DwellinglyDevACR --sku Basic
```

5. **Create an Azure Kubernetes Service (AKS) Cluster with ACR Integration:**

```
az aks create --resource-group DwellinglyDevResourceGroup
--name DwellinglyDevAKSCluster --node-count 1 --enable-add
ons monitoring --generate-ssh-keys --attach-acr Dwellingly
DevACR
```

6. **Get AKS Credentials:**

```
az aks get-credentials --resource-group DwellinglyDevResou
rceGroup --name DwellinglyDevAKSCluster
```

7. **Log in to ACR:**

```
az acr login --name DwellinglyDevACR
```

Step 5: Build and Deploy the Application for Development

1. **Build the Application:**
   - Open the `NuGet Package Manager Console` and build the solution.
   - Verify that the projects build successfully.
2. **Create a Dockerfile in the `Dwellingly.API` Project:**
   - Add a `Dockerfile` to the root of the `Dwellingly.API` project:

   ```
   FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
   WORKDIR /app
   EXPOSE 80

   FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
   WORKDIR /src
   ```

Step 5: Build and Deploy the Application for Development (continued)

1. **Create a Dockerfile in the `Dwellingly.API` Project** (continued):
   - Add a `Dockerfile` to the root of the `Dwellingly.API` project:

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
WORKDIR /app
EXPOSE 80

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY ["Dwellingly.API/Dwellingly.API.csproj", "Dwellingly.API/"]
RUN dotnet restore "Dwellingly.API/Dwellingly.API.csproj"
COPY . .
WORKDIR "/src/Dwellingly.API"
RUN dotnet build "Dwellingly.API.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "Dwellingly.API.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "Dwellingly.API.dll"]
```

2. **Build and Tag the Docker Image**:

- Open a terminal and navigate to the `Dwellingly.API` project directory.
- Build the Docker image:

```
docker build -t dwellingly-dev-api .
```

- Tag the Docker image for ACR:

```
docker tag dwellingly-dev-api dwellinglydevacr.azurecr.
```

```
io/dwellingly-dev-api:latest
```

3. **Push the Docker Image to ACR**:

   - Push the image to ACR:

   ```
   docker push dwellinglydevacr.azurecr.io/dwellingly-dev-
   api:latest
   ```

4. **Create Kubernetes Deployment and Service YAML Files**:

   - Create `deployment.yaml`:

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: dwellingly-dev-api
   spec:
     replicas: 1
     selector:
       matchLabels:
         app: dwellingly-dev-api
     template:
       metadata:
         labels:
           app: dwellingly-dev-api
       spec:
         containers:
         - name: dwellingly-dev-api
           image: dwellinglydevacr.azurecr.io/dwellingly-d
   ev-api:latest
           ports:
           - containerPort: 80
   ```

   - Create `service.yaml`:

```
apiVersion: v1
kind: Service
metadata:
  name: dwellingly-dev-api
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: dwellingly-dev-api
```

5. **Deploy to AKS**:
   - Apply the deployment and service configurations:

     ```
     kubectl apply -f deployment.yaml
     kubectl apply -f service.yaml
     ```

6. **Verify the Deployment**:
   - Check the status of the deployment:

     ```
     kubectl get deployments
     ```

   - Check the status of the service:

     ```
     kubectl get services
     ```

   - Get the external IP of the service:

     ```
     kubectl get service dwellingly-dev-api
     ```

Step 6: Setting Up DevOps CI/CD in Azure DevOps

1. **Set Up Azure DevOps Project**:

- Create a new project in Azure DevOps.

2. **Create CI Pipeline**:

   - Go to Pipelines -> Create Pipeline.
   - Select the repository where your code is hosted.
   - Configure the pipeline with the following YAML:

```
trigger:
- main

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: UseDotNet@2
  inputs:
    packageType: 'sdk'
    version: '8.x'
    installationPath: $(Agent.ToolsDirectory)/dotnet

- script: dotnet build --configuration Release
  displayName: 'Build project'

- task: PublishBuildArtifacts@1
  inputs:
    PathtoPublish: '$(Build.ArtifactStagingDirectory)'
    ArtifactName: 'drop'
    publishLocation: 'Container'
```

3. **Create CD Pipeline**:

   - Go to Releases -> New pipeline.
   - Configure the stages for deployment (e.g., Dev, Staging, Prod).
   - Set up deployment tasks to deploy the Docker image to AKS.

Step 7: Integrate User Data and Azure OpenAI for the Chatbot

1. **Configure Azure SQL Database for User Data**:
   - Use Entity Framework Core to interact with Azure SQL Database as shown in the backend configuration.
2. **Integrate Azure OpenAI Service**:
   - Sign up for Azure OpenAI Service and get your API key.
   - Add a service to call Azure OpenAI in your backend project:

```
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text;
using System.Threading.Tasks;
using Newtonsoft.Json;

namespace Dwellingly.API.Services
{
    public class OpenAIService
    {
        private readonly HttpClient _httpClient;
        private readonly string _apiKey = "YOUR_OPENAI_
API_KEY";

        public OpenAIService(HttpClient httpClient)
        {
            _httpClient = httpClient;
        }

        public async Task<string> GetChatbotResponse(st
ring prompt)
        {
            var requestContent = new StringContent(Json
Convert.SerializeObject(new
            {
                prompt = prompt,
                max_tokens = 150
            }), Encoding.UTF8, "application/json");

            _httpClient.DefaultRequestHeaders.Authoriza
tion = new AuthenticationHeaderValue("Bearer", _apiKe
y);

            var response = await _httpClient.PostAsync
("<https://api.openai.com/v1/engines/davinci-codex/comp
letions>", requestContent);
            response.EnsureSuccessStatusCode();

            var responseContent = await response.Conten
t.ReadAsStringAsync();
            var result = JsonConvert.DeserializeObject<
OpenAIResponse>(responseContent);

            return result.Choices.FirstOrDefault()?.Tex
t.Trim();
        }
    }

    public class OpenAIResponse
    {
        public List<Choice> Choices { get; set; }
    }

    public class Choice
    {
        public string Text { get; set; }
    }
}
```

3. **Integrate Chatbot in Frontend**:
    - Add a chat component to your Blazor app that interacts with the OpenAI service.

```
@page "/chat"

<h3>Chat with our AI</h3>

<div>
    <input @bind="userInput" placeholder="Type your message" />
    <button @onclick="SendMessage">Send</button>
</div>

<div>
    <p>@response</p>
</div>

@code {
    private string userInput = string.Empty;
    private string response = string.Empty;

    private async Task SendMessage()
    {
        var openAIService = new OpenAIService(new HttpClient());
        response = await openAIService.GetChatbotResponse(userInput);
    }
}
```

Conclusion

Following these steps will set up the development environment, create and run the Dwellingly | AI application locally, configure Azure resources for development, and integrate Azure OpenAI for the chatbot functionality. This comprehensive guide ensures that the Dwellingly | AI application is well-structured, scalable, and integrated with modern development practices.

ReadMe Instructions

ReadMe for Development Environment Setup and Container Deployment for Dwellingly | AI

Overview

This guide provides detailed instructions to set up the development environment and deploy the Dwellingly | AI application using Azure Kubernetes Service (AKS) and Azure Container Registry (ACR) without using Docker for deployment. Instead, Azure-native services are used to build and deploy the application.

Prerequisites

- Azure CLI installed
- kubectl CLI installed
- .NET 8 SDK
- Azure Subscription

Steps

1. **Log in to Azure:**

   ```
   shCopy code
   az login
   ```

2. **Create a resource group:**

   ```
   shCopy code
   az group create --name DwellinglyResourceGroup --location
   eastus
   ```

3. **Create an Azure Container Registry (ACR):**

   ```
   shCopy code
   az acr create --resource-group DwellinglyResourceGroup --n
   ```

```
ame DwellinglyACR --sku Basic
```

4. **Create an AKS Cluster with ACR integration:**

```
shCopy code
az aks create --resource-group DwellinglyResourceGroup --n
ame DwellinglyAKSCluster --node-count 1 --enable-addons mo
nitoring --generate-ssh-keys --attach-acr DwellinglyACR
```

5. **Get AKS credentials:**

```
shCopy code
az aks get-credentials --resource-group DwellinglyResource
Group --name DwellinglyAKSCluster
```

6. **Log in to ACR:**

```
shCopy code
az acr login --name DwellinglyACR
```

7. **Create a Dockerfile in your project root (this step uses Docker only for building the image):**

```
dockerfileCopy code
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
WORKDIR /app
EXPOSE 80

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY ["Dwellingly.API/Dwellingly.API.csproj", "Dwellingly.
API/"]
RUN dotnet restore "Dwellingly.API/Dwellingly.API.csproj"
COPY . .
WORKDIR "/src/Dwellingly.API"
RUN dotnet build "Dwellingly.API.csproj" -c Release -o /ap
p/build

FROM build AS publish
RUN dotnet publish "Dwellingly.API.csproj" -c Release -o /
app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "Dwellingly.API.dll"]
```

8. **Build the application image:**

```
shCopy code
docker build -t dwellingly-api .
```

9. **Tag the application image for ACR:**

```
shCopy code
docker tag dwellingly-api dwellinglyacr.azurecr.io/dwellin
gly-api:latest
```

10. **Push the application image to ACR:**

```
shCopy code
docker push dwellinglyacr.azurecr.io/dwellingly-api:latest
```

11. **Create a Kubernetes deployment YAML file (`deployment.yaml`):**

```
yamlCopy code
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dwellingly-api
spec:
  replicas: 3
  selector:
    matchLabels:
      app: dwellingly-api
  template:
    metadata:
      labels:
        app: dwellingly-api
    spec:
      containers:
      - name: dwellingly-api
        image: dwellinglyacr.azurecr.io/dwellingly-api:lat
est
        ports:
        - containerPort: 80
```

12. **Create a Kubernetes service YAML file (`service.yaml`):**

```
yamlCopy code
apiVersion: v1
kind: Service
metadata:
  name: dwellingly-api
spec:
  type: LoadBalancer
  ports:
```

```
  - port: 80
    targetPort: 80
  selector:
    app: dwellingly-api
```

13. **Deploy to AKS:**

```
shCopy code
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml
kubectl get deployments
kubectl get services
kubectl get service dwellingly-api
```

Conclusion

This guide provides all necessary steps to set up the development environment, create and run a .NET application image for your ASP.NET Core API, and deploy the containerized application to Azure Kubernetes Service, ensuring a robust and scalable infrastructure for the Dwellingly | AI application using Azure-native services.

create a script to install the pre requirements

Here is a script to install the prerequisites for setting up the Dwellingly | AI application on Azure Kubernetes Service (AKS) and Azure Container Registry (ACR). This script will:

1. Install Azure CLI
2. Install kubectl
3. Install .NET SDK
4. Log in to Azure

Prerequisites Installation Script

```
bashCopy code
#!/bin/bash

# Exit script on any error
set -e

# Update package list and install dependencies
sudo apt update
sudo apt install -y apt-transport-https ca-certificates curl
software-properties-common

# Install Azure CLI
echo "Installing Azure CLI..."
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash

# Verify Azure CLI installation
az --version

# Install kubectl
echo "Installing kubectl..."
sudo az aks install-cli

# Verify kubectl installation
kubectl version --client

# Install .NET SDK
echo "Installing .NET SDK..."
wget https://packages.microsoft.com/config/ubuntu/20.04/packa
ges-microsoft-prod.deb -O packages-microsoft-prod.deb
sudo dpkg -i packages-microsoft-prod.deb
rm packages-microsoft-prod.deb

sudo apt update
sudo apt install -y dotnet-sdk-8.0
```

```
# Verify .NET SDK installation
dotnet --version

# Log in to Azure
echo "Logging in to Azure..."
az login

echo "All prerequisites installed successfully."
```

Usage Instructions

1. **Save the Script:**
   - Save the script as `install_prerequisites.sh`.
2. **Make the Script Executable:**
   - Run the following command to make the script executable:

     ```
     shCopy code
     chmod +x install_prerequisites.sh
     ```
3. **Run the Script:**
   - Execute the script to install all prerequisites:

     ```
     shCopy code
     ./install_prerequisites.sh
     ```

This script will install the necessary tools and log you into Azure. Once the script completes, you will have Azure CLI, kubectl, and .NET SDK installed and ready to use.

Improved Setup Instructions for Dwellingly | AI

Overview

To set up the development environment and deploy the Dwellingly | AI application using Docker and Azure, follow these steps. This guide includes installing necessary tools, setting up the development environment, containerizing the application, and deploying it to Azure.

Prerequisites

- Visual Studio 2022
- Docker
- Azure CLI
- .NET 8 SDK
- Azure Subscription

Detailed Instructions

Step 1: Install Visual Studio 2022

1. **Download and Install Visual Studio 2022:**
    - Visit the Visual Studio website.
    - Download and install the Community, Professional, or Enterprise edition.
    - During installation, select the following workloads:
        - **ASP.NET and web development**
        - **Azure development**
        - **Data storage and processing**

Step 2: Configure Azure CLI

1. **Download and Install Azure CLI:**
    - Follow the instructions from the official Azure documentation.
2. **Sign in to your Azure account:**

```
shCopy code
az login
```

Step 3: Set Up Git

1. **Download and Install Git:**
   - Visit the official Git website.
2. **Configure Git with your user information:**

```
shCopy code
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

Step 4: Set Up Docker

1. **Download and Install Docker Desktop:**
   - Visit the Docker website and follow the installation instructions for your operating system.
2. **Verify Docker Installation:**

```
shCopy code
docker --version
```

Step 5: Create ASP.NET Core Project

1. **Open Visual Studio 2022.**
2. **Create a New Project:**
   - Select **ASP.NET Core Web API**.
   - Choose **.NET 8.0** as the target framework.
   - Name the project `Dwellingly.API`.

Step 6: Install Required NuGet Packages

1. **Open the NuGet Package Manager Console.**
2. **Install the following packages:**

```
shCopy code
Install-Package Microsoft.EntityFrameworkCore
Install-Package Microsoft.EntityFrameworkCore.SqlServer
Install-Package Dapper
Install-Package AutoMapper
Install-Package Microsoft.Data.SqlClient
```

Step 7: Configure Entity Framework Core

1. **Add a DbContext class:**

```
csharpCopy code
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<Applicati
onDbContext> options)
        : base(options)
    {
    }

    public DbSet<Property> Properties { get; set; }
}
```

2. **Configure the connection string in `appsettings.json`:**

```
jsonCopy code
"ConnectionStrings": {
  "DefaultConnection": "Server=tcp:<your_server>.database.
windows.net,1433;Initial Catalog=<your_db>;Persist Securit
y Info=False;User ID=<your_user>;Password=<your_password>;
MultipleActiveResultSets=False;Encrypt=True;TrustServerCer
tificate=False;Connection Timeout=30;"
}
```

3. **Register the DbContext in `Program.cs`:**

```
csharpCopy code
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionSt
ring("DefaultConnection")));
    services.AddControllers();
}
```

Step 8: Create API Endpoints

1. **Create a controller for property listings:**

```
csharpCopy code
[ApiController]
[Route("api/[controller]")]
public class PropertiesController : ControllerBase
{
    private readonly ApplicationDbContext _context;

    public PropertiesController(ApplicationDbContext conte
xt)
    {
        _context = context;
    }

    [HttpGet]
    public async Task<ActionResult<IEnumerable<Property>>>
GetProperties()
    {
        return await _context.Properties.ToListAsync();
    }
}
```

Step 9: Create Dockerfile for the API

1. **Add a `Dockerfile` to the root of the `Dwellingly.API` project:**

```
dockerfileCopy code
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
WORKDIR /app
EXPOSE 80

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY ["Dwellingly.API/Dwellingly.API.csproj", "Dwellingly.
API/"]
RUN dotnet restore "Dwellingly.API/Dwellingly.API.csproj"
COPY . .
WORKDIR "/src/Dwellingly.API"
RUN dotnet build "Dwellingly.API.csproj" -c Release -o /ap
p/build

FROM build AS publish
RUN dotnet publish "Dwellingly.API.csproj" -c Release -o /
app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "Dwellingly.API.dll"]
```

Step 10: Build and Run Docker Image Locally

1. **Open a terminal and navigate to the project directory.**
2. **Build the Docker image:**

   ```
   shCopy code
   docker build -t dwellingly-api .
   ```

3. **Run the Docker container:**

   ```
   shCopy code
   docker run -d -p 8080:80 --name dwellingly-api-container d
   wellingly-api
   ```

4. **Verify the application is running by navigating to `http://localhost:8080/api/properties` in your browser.**

Step 11: Push Docker Image to Azure Container Registry (ACR)

1. **Login to ACR:**

   ```
   shCopy code
   az acr login --name <your_acr_name>
   ```

2. **Tag the Docker image:**

   ```
   shCopy code
   docker tag dwellingly-api <your_acr_name>.azurecr.io/dwell
   ingly-api:latest
   ```

3. **Push the image to ACR:**

```
shCopy code
docker push <your_acr_name>.azurecr.io/dwellingly-api:late
st
```

Step 12: Create Azure App Service and Deploy Container

1. **Create a resource group:**

   ```
   shCopy code
   az group create --name DwellinglyResourceGroup --location
   eastus
   ```

2. **Create an Azure App Service plan:**

   ```
   shCopy code
   az appservice plan create --name DwellinglyAppServicePlan
   --resource-group DwellinglyResourceGroup --sku B1 --is-lin
   ux
   ```

3. **Create a Web App for Containers:**

   ```
   shCopy code
   az webapp create --resource-group DwellinglyResourceGroup
   --plan DwellinglyAppServicePlan --name DwellinglyWebApp --
   deployment-container-image-name <your_acr_name>.azurecr.i
   o/dwellingly-api:latest
   ```

4. **Configure the Web App to pull from ACR:**

   ```
   shCopy code
   az webapp config container set --name DwellinglyWebApp --r
   ```

```
esource-group DwellinglyResourceGroup --docker-custom-imag
e-name <your_acr_name>.azurecr.io/dwellingly-api:latest --
docker-registry-server-url https://<your_acr_name>.azurec
r.io
```

Conclusion

Following these steps will set up your development environment, create and run a Docker container for your ASP.NET Core API, and deploy the containerized application to Azure App Service. This setup ensures a robust and scalable infrastructure for the Dwellingly | AI application.

ReadMe Instructions

ReadMe for Development Environment Setup and Container Deployment for Dwellingly | AI

Overview

This guide provides detailed instructions to set up the development environment and deploy the Dwellingly | AI application using Docker and Azure. It covers installing necessary tools, setting up the development environment, containerizing the application, and deploying it to Azure.

Prerequisites

- Visual Studio 2022
- Docker
- Azure CLI
- .NET 8 SDK
- Azure Subscription

Steps

1. **Install Visual Studio 2022**

- Download and install Visual Studio 2022 from the Visual Studio website.
- Select the workloads: ASP.NET and web development, Azure development, Data storage and processing.

2. **Configure Azure CLI**
   - Download and install the Azure CLI from the official Azure documentation.
   - Sign in to your Azure account: `az login`.
3. **Set Up Git**
   - Download and install Git from the official Git website.
   - Configure Git:

```
shCopy code
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

4. **Set Up Docker**
   - Download and install Docker Desktop from the Docker website.
   - Verify Docker installation: `docker --version`.
5. **Create ASP.NET Core Project**
   - Open Visual Studio 2022.
   - Create a new project: ASP.NET Core Web API, .NET 8.0, named `Dwellingly.API`.
6. **Install Required NuGet Packages**
   - Open the NuGet Package Manager Console.
   - Install packages:

```
shCopy code
Install-Package Microsoft.EntityFrameworkCore
Install-Package Microsoft.EntityFrameworkCore.SqlServer
Install-Package Dapper
```

```
Install-Package AutoMapper
Install-Package Microsoft.Data.SqlClient
```

7. **Configure Entity Framework Core**
   - Add a `DbContext` class.
   - Configure the connection string in `appsettings.json`.
   - Register the `DbContext` in `Program.cs`.
8. **Create API Endpoints**
   - Create a controller for property listings.
9. **Create Dockerfile for the API**
   - Add a `Dockerfile` to the root of the `Dwellingly.API` project.
10. **Build and Run Docker Image Locally**
    - Build the Docker image: `docker build -t dwellingly-api .`
    - Run the Docker container: `docker run -d -p 8080:80 --name dwellingly-api-container dwellingly-api`
    - Verify the application at `http://localhost:8080/api/properties`.
11. **Push Docker Image to Azure Container Registry (ACR)**
    - Login to ACR: `az acr login --name <your_acr_name>`
    - Tag the Docker image: `docker tag dwellingly-api <your_acr_name>.azurecr.io/dwellingly-api:latest`
    - Push the image to ACR: `docker push <your_acr_name>.azurecr.io/dwellingly-api:latest`
12. **Create Azure App Service and Deploy Container**
    - Create a resource group: `az group create --name DwellinglyResourceGroup --location eastus`
    - Create an Azure App Service plan: `az appservice plan create --name DwellinglyAppServicePlan --resource-group DwellinglyResourceGroup --sku B1 --is-linux`
    - Create a Web App for Containers: `az webapp create --resource-group DwellinglyResourceGroup --plan DwellinglyAppServicePlan --name DwellinglyWebApp --`

`deployment-container-image-name <your_acr_name>.azurecr.io/dwellingly-api:latest`

- Configure the Web App to pull from ACR:

  ```
  shCopy code
  az webapp config container set --name DwellinglyWebApp
  --resource-group DwellinglyResourceGroup --docker-custo
  m-image-name <your_acr_name>.azurecr.io/dwellingly-api:
  latest --docker-registry-server-url https://<your_acr_n
  ame>.azurecr.io
  ```

Conclusion

This guide provides all necessary steps to set up the development environment, create and run a Docker container for your ASP.NET Core API, and deploy the containerized application to Azure App Service, ensuring a robust and scalable infrastructure for the Dwellingly | AI application.

Save Files

Let's save these instructions into separate files for download.

1. Backend Development Instructions
2. Frontend Development Instructions
3. AI/ML Integration Instructions
4. DevOps Instructions
5. ReadMe Instructions

Feel free to download each of these files to access the detailed instructions.

Here is a comprehensive script to set up the environment, including creating the necessary Azure resources, configuring them, and deploying the Dwellingly | AI application.

Azure Resource Setup Script

```
#!/bin/bash

# Exit script on any error
set -e

# Variables
RESOURCE_GROUP="DwellinglyResourceGroup"
LOCATION="eastus"
ACR_NAME="DwellinglyACR"
AKS_NAME="DwellinglyAKSCluster"
ACR_URL="$ACR_NAME.azurecr.io"
DOCKER_IMAGE_NAME="dwellingly-api"
DOCKER_IMAGE_TAG="latest"
APP_NAME="Dwellingly.API"
SQL_SERVER_NAME="dwellingly-sql-server"
SQL_DATABASE_NAME="DwellinglyDB"
SQL_ADMIN_USER="sqladmin"
SQL_ADMIN_PASSWORD="YourStrong(!)Password"
KEY_VAULT_NAME="DwellinglyKeyVault"

# Log in to Azure
echo "Logging in to Azure..."
az login

# Create resource group
echo "Creating resource group..."
az group create --name $RESOURCE_GROUP --location $LOCATION

# Create Azure Container Registry (ACR)
echo "Creating Azure Container Registry..."
az acr create --resource-group $RESOURCE_GROUP --name $ACR_NA
ME --sku Basic

# Create AKS Cluster with ACR integration
echo "Creating Azure Kubernetes Service (AKS) cluster..."
```

```
az aks create --resource-group $RESOURCE_GROUP --name $AKS_NA
ME --node-count 1 --enable-addons monitoring --generate-ssh-k
eys --attach-acr $ACR_NAME

# Get AKS credentials
echo "Getting AKS credentials..."
az aks get-credentials --resource-group $RESOURCE_GROUP --nam
e $AKS_NAME

# Create Azure SQL Database and Server
echo "Creating Azure SQL Server..."
az sql server create --name $SQL_SERVER_NAME --resource-group
$RESOURCE_GROUP --location $LOCATION --admin-user $SQL_ADMIN_
USER --admin-password $SQL_ADMIN_PASSWORD

echo "Creating Azure SQL Database..."
az sql db create --resource-group $RESOURCE_GROUP --server $S
QL_SERVER_NAME --name $SQL_DATABASE_NAME --service-objective
S0

# Enable auditing and threat detection on SQL Database
echo "Enabling auditing and threat detection on SQL Databas
e..."
az sql db audit-policy update --name $SQL_DATABASE_NAME --res
ource-group $RESOURCE_GROUP --server $SQL_SERVER_NAME --state
Enabled
az sql db threat-policy update --name $SQL_DATABASE_NAME --re
source-group $RESOURCE_GROUP --server $SQL_SERVER_NAME --stat
e Enabled

# Create Azure Key Vault
echo "Creating Azure Key Vault..."
az keyvault create --name $KEY_VAULT_NAME --resource-group $R
ESOURCE_GROUP --location $LOCATION --sku standard --enable-so
ft-delete true --enable-purge-protection true
az keyvault secret set --vault-name $KEY_VAULT_NAME --name "S
```

```
qlAdminPassword" --value $SQL_ADMIN_PASSWORD

# Log in to ACR
echo "Logging in to ACR..."
az acr login --name $ACR_NAME

# Build and push Docker image (this uses Docker for building)
echo "Building Docker image..."
docker build -t $DOCKER_IMAGE_NAME .

echo "Tagging Docker image..."
docker tag $DOCKER_IMAGE_NAME $ACR_URL/$DOCKER_IMAGE_NAME:$DO
CKER_IMAGE_TAG

echo "Pushing Docker image to ACR..."
docker push $ACR_URL/$DOCKER_IMAGE_NAME:$DOCKER_IMAGE_TAG

# Create Kubernetes deployment YAML file
cat <<EOF > deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dwellingly-api
spec:
  replicas: 3
  selector:
    matchLabels:
      app: dwellingly-api
  template:
    metadata:
      labels:
        app: dwellingly-api
    spec:
      containers:
      - name: dwellingly-api
        image: $ACR_URL/$DOCKER_IMAGE_NAME:$DOCKER_IMAGE_TAG
        ports:
        - containerPort: 80
EOF

# Create Kubernetes service YAML file
cat <<EOF > service.yaml
apiVersion: v1
kind: Service
metadata:
  name: dwellingly-api
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: dwellingly-api
EOF

# Deploy to AKS
echo "Deploying application to AKS..."
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml

echo "Setup and deployment completed successfully."
```

Instructions to Run the Script

1. **Save the Script:**
   - Save the script as `setup_azure_environment.sh`.
2. **Make the Script Executable:**
   - Run the following command to make the script executable:

     ```
     chmod +x setup_azure_environment.sh
     ```

3. **Run the Script:**
    - Execute the script to set up the Azure environment and deploy the application:

      ```
      ./setup_azure_environment.sh
      ```

Explanation

1. **Logging into Azure:**
    - The script starts by logging into Azure using the Azure CLI.
2. **Resource Creation:**
    - It creates a resource group, Azure Container Registry (ACR), Azure Kubernetes Service (AKS) cluster, Azure SQL Server and Database, and an Azure Key Vault.
3. **Build and Push Docker Image:**
    - It builds a Docker image for the Dwellingly API application, tags it, and pushes it to the ACR.
4. **Kubernetes Deployment:**
    - The script creates deployment and service YAML files for Kubernetes and applies them to deploy the application to the AKS cluster.

This script automates the entire setup and deployment process, ensuring a consistent and reproducible environment for the Dwellingly | AI application.