Complete Backend Setup for Dwellingly | AI in Visual Studio

Below is the complete backend setup for the Dwellingly | AI application, which includes the project structure, configurations, services, middleware, and controllers.

Step-by-Step Guide

Project Structure

```
luaCopy code
Dwellingly
|-- Controllers/
|   |-- PropertyController.cs
|-- Data/
|   |-- NexHomeAgentContext.cs
|-- Middleware/
|   |-- ExceptionMiddleware.cs
|   |-- RequestLoggingMiddleware.cs
|-- Models/
|   |-- ChatMessage.cs
|   |-- Property.cs
|   |-- User.cs
|   |-- Favorite.cs
|   |-- PropertyImage.cs
|   |-- PropertyHistory.cs
|-- Services/
|   |-- ChatService.cs
|   |-- IChatService.cs
|   |-- PropertyCommandService.cs
|   |-- PropertyQueryService.cs
|-- Program.cs
|-- appsettings.json
```

1. Define the Database Context and Models

Data/NexHomeAgentContext.cs

```
csharpCopy code
using Microsoft.EntityFrameworkCore;
using NexHomeAgent.Models;

namespace NexHomeAgent.Data
{
    public class NexHomeAgentContext : DbContext
    {
        public NexHomeAgentContext(DbContextOptions<NexHomeAg
entContext> options)
            : base(options)
        {
        }

        public DbSet<Property> Properties { get; set; }
        public DbSet<ChatMessage> ChatMessages { get; set; }
        public DbSet<User> Users { get; set; }
        public DbSet<Favorite> Favorites { get; set; }
        public DbSet<PropertyImage> PropertyImages { get; se
t; }
        public DbSet<PropertyHistory> PropertyHistories { ge
t; set; }
    }
}
```

Models/Property.cs

```
csharpCopy code
namespace NexHomeAgent.Models
{
    public class Property
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public decimal Price { get; set; }
        public string Location { get; set; }
        public int Bedrooms { get; set; }
        public int Bathrooms { get; set; }
        public string PhotoUrl { get; set; }
    }
}
```

Models/ChatMessage.cs

```
csharpCopy code
namespace NexHomeAgent.Models
{
    public class ChatMessage
    {
        public int Id { get; set; }
        public string Text { get; set; }
        public bool IsUser { get; set; }
        public DateTime Timestamp { get; set; }
    }
}
```

Models/User.cs

```
csharpCopy code
namespace NexHomeAgent.Models
{
    public class User
    {
        public int Id { get; set; }
        public string Email { get; set; }
        public string Password { get; set; }
    }
}
```

Models/Favorite.cs

```
csharpCopy code
namespace NexHomeAgent.Models
{
    public class Favorite
    {
        public int Id { get; set; }
        public int UserId { get; set; }
        public int PropertyId { get; set; }
    }
}
```

Models/PropertyImage.cs

```
csharpCopy code
namespace NexHomeAgent.Models
{
    public class PropertyImage
    {
        public int Id { get; set; }
        public int PropertyId { get; set; }
        public string ImageUrl { get; set; }
    }
}
```

Models/PropertyHistory.cs

```
csharpCopy code
namespace NexHomeAgent.Models
{
    public class PropertyHistory
    {
        public int Id { get; set; }
        public int PropertyId { get; set; }
        public string Description { get; set; }
        public DateTime Date { get; set; }
    }
}
```

2. Configure Entity Framework Core

1. **Install Entity Framework Core NuGet Packages**:
    - Open the **NuGet Package Manager Console** (Tools > NuGet Package Manager > Package Manager Console).
    - Run the following commands:

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

2. **Add Connection String to `appsettings.json`:**
   - Add the connection string to `appsettings.json`:

```
jsonCopy code
{
  "ConnectionStrings": {
    "AzureSqlDatabase": "Server=your_server_name;Databa
se=DwellinglyDB;Trusted_Connection=True;MultipleActiveR
esultSets=true"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "OpenAI": {
    "ApiKey": "YOUR_API_KEY"
  }
}
```

3. **Configure `Program.cs`:**
   - Update `Program.cs` to register the `DbContext` and configure other services:

```
csharpCopy code
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Serilog;
using Azure.Identity;
using Azure.Security.KeyVault.Secrets;
using NexHomeAgent.Middleware;
using NexHomeAgent.Data;
using NexHomeAgent.Services;

var builder = WebApplication.CreateBuilder(args);

// Configure Serilog
Log.Logger = new LoggerConfiguration()
    .MinimumLevel.Information()
    .WriteTo.Console()
    .WriteTo.File("logs/nexhomeagent.txt", rollingInter
val: RollingInterval.Day)
    .CreateLogger();
builder.Host.UseSerilog();

// Add services to the container
builder.Services.AddControllers();
builder.Services.AddAuthentication(AzureADB2CDefaults.B
earerAuthenticationScheme)
    .AddAzureADB2CBearer(options =>
    {
        options.Instance = builder.Configuration["Azure
AdB2C:Instance"];
        options.ClientId = builder.Configuration["Azure
AdB2C:ClientId"];
        options.Domain = builder.Configuration["AzureAd
B2C:Domain"];
        options.SignUpSignInPolicyId = builder.Configur
ation["AzureAdB2C:SignUpSignInPolicyId"];
    });
```

```
builder.Services.AddDbContext<NexHomeAgentContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("AzureSqlDatabase")));

builder.Services.AddHttpClient<IChatService, ChatService>(client =>
{
    client.BaseAddress = new Uri("https://api.openai.com/v1/");
    client.DefaultRequestHeaders.Add("Authorization", "Bearer " + builder.Configuration["OpenAI:ApiKey"]);
});

builder.Services.AddApplicationInsightsTelemetry(options =>
{
    options.InstrumentationKey = builder.Configuration["ApplicationInsights:InstrumentationKey"];
});

var app = builder.Build();

// Seed the database
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    var context = services.GetRequiredService<NexHomeAgentContext>();
    DbInitializer.Initialize(context);
}

// Configure the HTTP request pipeline
if (app.Environment.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
}

app.UseHttpsRedirection();
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();

app.UseMiddleware<ExceptionMiddleware>();

app.MapControllers();

app.Run();
```

4. **Add Migrations and Update the Database**:

   - Open the **Package Manager Console** and run:

```
shCopy code
Add-Migration InitialCreate
Update-Database
```

3. Seed Initial Data

1. **Create Seed Data Class**:

   - Add a new class `DbInitializer.cs` in the `Data` folder:

```
csharpCopy code
using Microsoft.Extensions.DependencyInjection;
using NexHomeAgent.Models;

namespace NexHomeAgent.Data
{
    public static class DbInitializer
    {
        public static void Initialize(NexHomeAgentConte
xt context)
        {
            context.Database.EnsureCreated();

            // Check if users already exist
            if (context.Users.Any())
            {
                return;   // DB has been seeded
            }

            var users = new User[]
            {
                new User{Email="user1@example.com", Pas
sword="Password1"},
                new User{Email="user2@example.com", Pas
sword="Password2"},
            };

            foreach (var u in users)
            {
                context.Users.Add(u);
            }

            context.SaveChanges();

            var properties = new Property[]
            {
                new Property{Title="Property1", Descrip
tion="Description1", Price=100000, Location="Location
1", Bedrooms=3, Bathrooms=2, PhotoUrl="URL1"},
                new Property{Title="Property2", Descrip
tion="Description2", Price=200000, Location="Location
2", Bedrooms=4, Bathrooms=3, PhotoUrl="URL2"},
            };
```

```
                foreach (var p in properties)
                {
                    context.Properties.Add(p);
                }

                context.SaveChanges();
            }
        }
    }
```

2. **Update `Program.cs` to Seed Data**:

   - Ensure the database seeding is called during application startup:

```
csharpCopy code
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Serilog;
using Azure.Identity;
using Azure.Security.KeyVault.Secrets;
using NexHomeAgent.Middleware;
using NexHomeAgent.Data;
using NexHomeAgent.Services;

var builder = WebApplication.CreateBuilder(args);

// Configure Serilog
Log.Logger = new LoggerConfiguration()
    .MinimumLevel.Information()
    .WriteTo.Console()
    .WriteTo.File("logs/nexhomeagent.txt", rollingInter
val: RollingInterval.Day)
    .CreateLogger();
builder.Host.UseSerilog();
```

```
// Add services to the container
builder.Services.AddControllers();
builder.Services.AddAuthentication(AzureADB2CDefaults.B
earerAuthenticationScheme)
    .AddAzureADB2CBearer(options =>
    {
        options.Instance = builder.Configuration["Azure
AdB2C:Instance"];
        options.ClientId = builder.Configuration["Azure
AdB2C:ClientId"];
        options.Domain = builder.Configuration["AzureAd
B2C:Domain"];
        options.SignUpSignInPolicyId = builder.Configur
ation["AzureAdB2C:SignUpSignInPolicyId"];
    });

builder.Services.AddDbContext<NexHomeAgentContext>(opti
ons =>
    options.UseSqlServer(builder.Configuration.GetConne
ctionString("AzureSqlDatabase")));

builder.Services.AddHttpClient<IChatService, ChatServic
e>(client =>
{
    client.BaseAddress = new Uri("https://api.openai.co
m/v1/");
    client.DefaultRequestHeaders.Add("Authorization",
"Bearer " + builder.Configuration["OpenAI:ApiKey"]);
});

builder.Services.AddApplicationInsightsTelemetry(option
s =>
{
    options.InstrumentationKey = builder.Configuration
["ApplicationInsights:InstrumentationKey"];
});

var app = builder.Build();

// Seed the database
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    var context = services.GetRequiredService<NexHomeAg
entContext>();
    DbInitializer.Initialize(context);
}

// Configure the HTTP request pipeline
if (app.Environment.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
}

app.UseHttpsRedirection();
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();

app.UseMiddleware<ExceptionMiddleware>();

app.MapControllers();

app.Run();
```

4. Implement Middleware

Middleware/ExceptionMiddleware.cs

```
csharpCopy code
using Microsoft.AspNetCore.Http;
using Serilog;
using System;
using System.Threading.Tasks;

namespace NexHomeAgent.Middleware
{
    public class ExceptionMiddleware
    {
        private readonly RequestDelegate _next;

        public ExceptionMiddleware(RequestDelegate next)
        {
            _next = next;
        }

        public async Task InvokeAsync(HttpContext context)
        {
            try
            {
                await _next(context);
            }
            catch (Exception ex)
            {
                Log.Error(ex, "An unhandled exception occurre
d.");
                context.Response.StatusCode = StatusCodes.Sta
tus500InternalServerError;
                await context.Response.WriteAsync("An error o
ccurred. Please try again later.");
            }
        }
    }
}
```

5. Implement Controllers and Services

Controllers/PropertyController.cs

```
csharpCopy code
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using NexHomeAgent.Models;
using NexHomeAgent.Services;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace NexHomeAgent.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class PropertyController : ControllerBase
    {
        private readonly IPropertyCommandService _propertyCom
mandService;
        private readonly IPropertyQueryService _propertyQuery
Service;
        private readonly ILogger<PropertyController> _logger;

        public PropertyController(
            IPropertyCommandService propertyCommandService,
            IPropertyQueryService propertyQueryService,
            ILogger<PropertyController> logger)
        {
            _propertyCommandService = propertyCommandService;
            _propertyQueryService = propertyQueryService;
            _logger = logger;
        }
        [HttpGet("search")]
        public async Task<ActionResult<IEnumerable<Property>>
> Search([FromQuery] PropertySearchQuery query)
        {
            _logger.LogInformation("Searching for properties
with criteria: {criteria}", query);
            var properties = await _propertyQueryService.Sear
chPropertiesAsync(query);
            return Ok(properties);
        }

        [HttpGet("{id}")]
        public async Task<IActionResult> GetProperty(int id)
        {
            _logger.LogInformation("Fetching property with I
D: {id}", id);
            var property = await _propertyQueryService.GetPro
pertyAsync(id);
            if (property == null)
            {
                _logger.LogWarning("Property with ID {id} not
found", id);
                return NotFound();
            }
            return Ok(property);
        }

        [HttpPost]
        public async Task<IActionResult> CreateProperty(Creat
ePropertyCommand command)
        {
            _logger.LogInformation("Creating a new property:
{property}", command);
            await _propertyCommandService.CreatePropertyAsync
(command);
```

```
            return CreatedAtAction(nameof(GetProperty), new {
id = command.Id }, command);
        }
    }
}
```

Services/PropertyCommandService.cs

```
csharpCopy code
using NexHomeAgent.Models;
using NexHomeAgent.Data;
using Microsoft.Extensions.Logging;
using System.Threading.Tasks;

namespace NexHomeAgent.Services
{
    public class PropertyCommandService : IPropertyCommandSer
vice
    {
        private readonly NexHomeAgentContext _context;
        private readonly ILogger<PropertyCommandService> _log
ger;

        public PropertyCommandService(NexHomeAgentContext con
text, ILogger<PropertyCommandService> logger)
        {
            _context = context;
            _logger = logger;
        }

        public async Task CreatePropertyAsync(CreatePropertyC
ommand command)
        {
            _logger.LogInformation("Creating a new property:
{property}", command);
```

```
            var property = new Property
            {
                Title = command.Title,
                Description = command.Description,
                Price = command.Price,
                Location = command.Location,
                Bedrooms = command.Bedrooms,
                Bathrooms = command.Bathrooms,
                PhotoUrl = command.PhotoUrl
            };
            _context.Properties.Add(property);
            await _context.SaveChangesAsync();
        }
    }
}
```

Services/PropertyQueryService.cs

```
csharpCopy code
using NexHomeAgent.Models;
using NexHomeAgent.Data;
using Microsoft.Extensions.Logging;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;

namespace NexHomeAgent.Services
{
    public class PropertyQueryService : IPropertyQueryService
    {
        private readonly NexHomeAgentContext _context;
        private readonly ILogger<PropertyQueryService> _logge
r;

        public PropertyQueryService(NexHomeAgentContext conte
xt, ILogger<PropertyQueryService> logger)
        {
            _context = context;
            _logger = logger;
        }

        public async Task<IEnumerable<Property>> SearchProper
tiesAsync(PropertySearchQuery query)
        {
            _logger.LogInformation("Searching for properties
with criteria: {criteria}", query);
            return await _context.Properties
                .Where(p => p.Location.Contains(query.Locatio
n) &&
                            p.Price >= query.MinPrice &&
                            p.Price <= query.MaxPrice &&
                            p.Bedrooms == query.Bedrooms)
                .ToListAsync();
        }

        public async Task<Property> GetPropertyAsync(int id)
        {
            _logger.LogInformation("Fetching property with I
D: {id}", id);
            return await _context.Properties.FindAsync(id);
        }
    }
}
```

6. Interface Definitions

Services/IPropertyCommandService.cs

```
csharpCopy code
using NexHomeAgent.Models;
using System.Threading.Tasks;

namespace NexHomeAgent.Services
{
    public interface IPropertyCommandService
    {
        Task CreatePropertyAsync(CreatePropertyCommand comman
d);
    }
}
```

Services/IPropertyQueryService.cs

```
csharpCopy code
using NexHomeAgent.Models;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace NexHomeAgent.Services
{
    public interface IPropertyQueryService
    {
        Task<IEnumerable<Property>> SearchPropertiesAsync(Pro
pertySearchQuery query);
        Task<Property> GetPropertyAsync(int id);
    }
}
```

Conclusion

This guide provides a complete setup for the backend of the Dwellingly | AI application using Visual Studio. It includes the configuration for Entity Framework Core, the necessary models, services, middleware, and controllers. This setup ensures a scalable, secure, and maintainable backend for your application.

Please let me know if you need further assistance or additional features!

This guide provides detailed instructions to set up the backend for the Dwellingly | AI application in Visual Studio, including creating the project, configuring Entity Framework Core, and integrating services.