Install the necessary front-end dependencies in Visual Studio for a Blazor project,

To install the necessary front-end dependencies in Visual Studio for a Blazor project, you typically need to ensure you have the required packages and tools installed. This process involves setting up your project with the right NuGet packages and potentially some npm packages if you're using additional JavaScript libraries.

Here's a step-by-step guide to set up the front-end dependencies in Visual Studio for a Blazor project:

Step 1: Ensure Visual Studio is Up-to-Date

Make sure you have the latest version of Visual Studio installed with the required workloads:

1. **ASP.NET and web development** workload.
2. **.NET Core cross-platform development** workload.

Step 2: Create or Open Your Blazor Project

If you don't already have a Blazor project, you can create one:

1. Open Visual Studio.
2. Go to **File > New > Project**.
3. Search for **Blazor** and select **Blazor WebAssembly App** or **Blazor Server App**.
4. Follow the prompts to create your project.

Step 3: Install Required NuGet Packages

You may need to install specific NuGet packages for your Blazor project. Here's how to do it:

1. Open the **NuGet Package Manager** by right-clicking on your project in the Solution Explorer and selecting **Manage NuGet Packages**.
2. In the **Browse** tab, search for and install the following packages (if not already installed):
   - `Microsoft.AspNetCore.Components.Web`
   - `Microsoft.AspNetCore.Components.WebAssembly`
   - `Microsoft.AspNetCore.Components.Authorization`
   - `Microsoft.Extensions.Logging`
   - `Microsoft.Extensions.DependencyInjection`

Step 4: Add JavaScript and CSS Libraries (Optional)

If you need additional JavaScript libraries or CSS frameworks (e.g., Bootstrap), you can install them via npm. Here's how to do it:

1. Ensure you have Node.js installed. You can download it from nodejs.org.
2. Open a command prompt or terminal in the root directory of your project.
3. Initialize npm (if not already done):

```
npm init -y
```

4. Install the required libraries. For example, to install Bootstrap:

```
npm install bootstrap
```

Step 5: Reference JavaScript and CSS in Your Blazor Project

After installing the JavaScript and CSS libraries, you need to reference them in your Blazor project. You can do this by updating the `index.html` (for Blazor WebAssembly) or `_Host.cshtml` (for Blazor Server) files.

For Blazor WebAssembly (index.html):

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>BlazorApp</title>
    <base href="/" />
    <link href="css/bootstrap.min.css" rel="stylesheet" />
    <link href="css/app.css" rel="stylesheet" />
</head>
<body>
    <app>Loading...</app>

    <script src="_framework/blazor.webassembly.js"></script>
    <script src="js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

For Blazor Server (_Host.cshtml):

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initia
l-scale=1.0" />
    <title>BlazorApp</title>
    <link href="~/css/bootstrap.min.css" rel="stylesheet" />
    <link href="~/css/app.css" rel="stylesheet" />
</head>
<body>
    <app>
        <component type="typeof(App)" render-mode="ServerPrer
endered" />
    </app>
```

```
    <script src="_framework/blazor.server.js"></script>
    <script src="~/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

Step 6: Restore and Build the Project

1. **Restore NuGet Packages:**
   - Right-click on the solution in the Solution Explorer.
   - Select **Restore NuGet Packages**.
2. **Build the Project:**
   - Right-click on the project in the Solution Explorer.
   - Select **Build**.

Step 7: Run the Project

Run your project to ensure everything is working correctly:

- Press `F5` to start debugging or `Ctrl+F5` to run without debugging.

By following these steps, you should have all the necessary front-end dependencies installed and configured in your Blazor project using Visual Studio. If you encounter any issues or need additional libraries, you can follow similar steps to add and configure them accordingly.

MVP Development Plan for Dwellingly | AI

Overview

To develop the MVP for Dwellingly | AI using the Microsoft technology stack within a 1-month timeframe, we will leverage various Microsoft tools and services to streamline development. Here's a detailed plan:

Technology Stack

- **Backend:** ASP.NET Core
- **Frontend:** Blazor
- **Database:** Azure SQL Database
- **AI/ML:** Azure Machine Learning and Azure OpenAI (GPT-4)
- **Hosting and Infrastructure:** Azure App Service
- **DevOps:** Azure DevOps

Timeline and Key Activities

Week 1: Project Kickoff and Initial Setup

1. **Project Kickoff:**
    - Conduct a project kickoff meeting, assign roles and responsibilities, and set up project management tools and communication channels.
2. **Requirement Gathering:**
    - Conduct stakeholder interviews to gather core requirements.
    - Document essential functional requirements.
    - Create basic user stories and use cases.

3. **System Architecture:**
    - Design high-level system architecture using Microsoft technologies.
    - Define key system components.
    - Create initial wireframes for the user interface.

Week 2: Development Setup and Backend Implementation

1. **Development Environment Setup:**
    - Set up development environments for the backend and frontend using Visual Studio.
    - Configure version control systems with Azure Repos.
    - Install necessary development tools and libraries.
2. **Backend Development:**
    - Implement core backend functionalities using ASP.NET Core.
    - Develop data models and database schema in Azure SQL Database.
    - Create API endpoints for property search and listings using canned data.
    - Implement AI-driven valuation tools using Azure Machine Learning with simulated data.

Week 3: Frontend Implementation and Integration

1. **Frontend Development:**
    - Implement core frontend components using Blazor.
    - Develop user interface for property search and listings.
    - Use mock data to simulate real estate properties.
2. **Integration:**
    - Integrate frontend with backend APIs using canned data.
    - Ensure data flow and functionality using simulated data.
    - Deploy backend and frontend to Azure App Service for hosting.

Week 4: Finalization and Demo Preparation

1. **Testing and Bug Fixing:**
    - Conduct basic functionality tests to ensure the MVP works as intended.

- Fix any critical issues identified during testing.

2. **Demo Preparation:**
   - Prepare demo scripts and presentations.
   - Create marketing materials and documentation for the demo.
   - Conduct internal demo to ensure readiness.
3. **Final Adjustments:**
   - Make any final adjustments based on internal feedback.
   - Ensure the MVP is polished and ready for presentation.
4. **Investor Presentation:**
   - Present the sandboxed MVP to potential investors.
   - Collect feedback and address any questions.
   - Plan next steps based on investor feedback and interest.

Detailed Instructions for Development

Prerequisites

Software

- **Visual Studio 2022**
- **.NET 8 SDK**
- **Azure CLI**
- **Postman**
- **Git**

Accounts

- **Azure Subscription**
- **GitHub or Azure Repos Account**

Development Environment Setup

Step 1: Install Visual Studio 2022

1. Download and install Visual Studio 2022 from the official website.
2. 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 the Azure CLI 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 from 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"
```

Backend Development

Step 1: 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 2: 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 3: Configure Entity Framework Core

1. Add a DbContext class:

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

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

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

```
jsonCopy code
"ConnectionStrings": {
  "DefaultConnection": "Server=tcp:<your_server>.database.window
s.net,1433;Initial Catalog=<your_db>;Persist Security Info=False;
User ID=<your_user>;Password=<your_password>;MultipleActiveResult
```

```
Sets=False;Encrypt=True;TrustServerCertificate=False;Connection T
imeout=30;"
}
```

1. Register the DbContext in `Program.cs`:

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

Step 4: 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 context)
    {
        _context = context;
    }

    [HttpGet]
    public async Task<ActionResult<IEnumerable<Property>>> GetPro
perties()
    {
```

```
        return await _context.Properties.ToListAsync();
    }
}
```

Frontend Development

Step 1: Create Blazor Project

1. Open Visual Studio 2022.
2. Create a new project:
    - Select **Blazor WebAssembly App**.
    - Name the project `Dwellingly.Client`.

Step 2: Install Required NuGet Packages

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

```
shCopy code
Install-Package Blazorise
Install-Package ChartJs.Blazor
Install-Package MudBlazor
```

Step 3: Configure Blazor Components

1. Configure Blazorise in `Program.cs`:

```
csharpCopy code
builder.Services.AddBlazorise(options =>
{
    options.ChangeTextOnKeyPress = true;
})
.AddBootstrapProviders()
```

```
.AddFontAwesomeIcons();
```

Step 4: Create Property Listing Component

1. Create a new Razor component `PropertyList.razor`:

```
razorCopy code
@page "/properties"
@inject HttpClient Http

<h3>Property Listings</h3>
<ul>
    @foreach (var property in properties)
    {
        <li>@property.Name - @property.Price</li>
    }
</ul>

@code {
    private List<Property> properties;

    protected override async Task OnInitializedAsync()
    {
        properties = await Http.GetFromJsonAsync<List<Property>>
("api/properties");
    }
}
```

AI/ML Integration

Step 1: Set Up Azure Machine Learning

1. Create an Azure Machine Learning workspace in the Azure portal.
2. Install the AzureML SDK for Python:

```
shCopy code
pip install azureml-sdk
```

Step 2: Train and Deploy Models

1. Use the AzureML SDK to train and deploy models:

```
pythonCopy code
from azureml.core import Workspace, Experiment
from azureml.train.automl import AutoMLConfig

ws = Workspace.from_config()
experiment = Experiment(ws, "property-valuation")

automl_config = AutoMLConfig(
    task="regression",
    training_data=train_data,
    label_column_name="price",
    primary_metric="r2_score",
    compute_target=compute_target,
    max_trials=5
)

run = experiment.submit(automl_config, show_output=True)
```

Hosting and Deployment

Step 1: Deploy to Azure App Service

1. Right-click the project in Visual Studio and select **Publish**.
2. Choose **Azure** as the target.
3. Follow the prompts to create a new Azure App Service instance and deploy the application.

Step 2: Configure CI/CD with Azure DevOps

1. Create a new project in Azure DevOps.
2. Set up a pipeline using the YAML file:

```
yamlCopy code
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'
```

References

For additional information and resources on developing ASP.NET Core and Blazor applications with Azure integration, refer to:

- Blazor | Build client web apps with C#
- ASP.NET documentation | Microsoft Learn
- ASP.NET Core | Open-source web framework for .NET
- Error after deploying asp.net core app to azure

This detailed developer guide ensures that the engineering team has all the necessary instructions and resources to successfully develop the Dwellingly | AI Sandboxed MVP using the specified Microsoft technologies.

Detailed Instructions for Development

Prerequisites

Software

- **Visual Studio 2022**
- **.NET 8 SDK**
- **Azure CLI**
- **Postman**
- **Git**

Accounts

- **Azure Subscription**
- **GitHub or Azure Repos Account**

Development Environment Setup

Step 1: Install Visual Studio 2022

1. Download and install Visual Studio 2022 from the official website.
2. 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 the Azure CLI 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 from 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"
```

Backend Development

Step 1: 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 2: 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 3: Configure Entity Framework Core

1. Add a DbContext class:

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

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

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

```
jsonCopy code
"ConnectionStrings": {
  "DefaultConnection": "Server=tcp:<your_server>.database.window
s.net,1433;Initial Catalog=<your_db>;Persist Security Info=False;
User ID=<your_user>;Password=<your_password>;MultipleActiveResult
Sets=False;Encrypt=True;TrustServerCertificate=False;Connection T
imeout=30;"
}
```

1. Register the DbContext in `Program.cs`:

```
csharpCopy code
public void ConfigureServices(IServiceCollection services)
{
```

```
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("D
efaultConnection")));
    services.AddControllers();
}
```

Step 4: 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 context)
    {
        _context = context;
    }

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

Frontend Development

Step 1: Create Blazor Project

1. Open Visual Studio 2022.

2. Create a new project:
    - Select **Blazor WebAssembly App**.
    - Name the project `Dwellingly.Client`.

Step 2: Install Required NuGet Packages

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

```
shCopy code
Install-Package Blazorise
Install-Package ChartJs.Blazor
Install-Package MudBlazor
```

Step 3: Configure Blazor Components

1. Configure Blazorise in `Program.cs`:

```
csharpCopy code
builder.Services.AddBlazorise(options =>
{
    options.ChangeTextOnKeyPress = true;
})
.AddBootstrapProviders()
.AddFontAwesomeIcons();
```

Step 4: Create Property Listing Component

1. Create a new Razor component `PropertyList.razor`:

```
razorCopy code
@page "/properties"
@inject HttpClient Http

<h3>Property Listings</h3>
```

```
<ul>
    @foreach (var property in properties)
    {
        <li>@property.Name - @property.Price</li>
    }
</ul>

@code {
    private List<Property> properties;

    protected override async Task OnInitializedAsync()
    {
        properties = await Http.GetFromJsonAsync<List<Property>>
("api/properties");
    }
}
```

AI/ML Integration

Step 1: Set Up Azure Machine Learning

1. Create an Azure Machine Learning workspace in the Azure portal.
2. Install the AzureML SDK for Python:

```
shCopy code
pip install azureml-sdk
```

Step 2: Train and Deploy Models

1. Use the AzureML SDK to train and deploy models:

```
pythonCopy code
from azureml.core import Workspace, Experiment
from azureml.train.automl import AutoMLConfig
```

```
ws = Workspace.from_config()
experiment = Experiment(ws, "property-valuation")

automl_config = AutoMLConfig(
    task="regression",
    training_data=train_data,
    label_column_name="price",
    primary_metric="r2_score",
    compute_target=compute_target,
    max_trials=5
)

run = experiment.submit(automl_config, show_output=True)
```

Hosting and Deployment

Step 1: Deploy to Azure App Service

1. Right-click the project in Visual Studio and select **Publish**.
2. Choose **Azure** as the target.
3. Follow the prompts to create a new Azure App Service instance and deploy the application.

Step 2: Configure CI/CD with Azure DevOps

1. Create a new project in Azure DevOps.
2. Set up a pipeline using the YAML file:

```
yamlCopy code
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'
```

References

For additional information and resources on developing ASP.NET Core and Blazor applications with Azure integration, refer to:

- Blazor | Build client web apps with C#
- ASP.NET documentation | Microsoft Learn
- ASP.NET Core | Open-source web framework for .NET
- Error after deploying asp.net core app to azure

This detailed developer guide ensures that the engineering team has all the necessary instructions and resources to successfully develop the Dwellingly | AI Sandboxed MVP using the specified Microsoft technologies.

import os

Define the content for each file

content_mvp_plan = """

MVP Development Plan for Dwellingly | AI

Overview

To develop the MVP for Dwellingly | AI using the Microsoft technology stack within a 1-month timeframe, we will leverage various Microsoft tools and services to streamline development. Here's a detailed plan:

Technology Stack

- **Backend:** ASP.NET Core
- **Frontend:** Blazor
- **Database:** Azure SQL Database
- **AI/ML:** Azure Machine Learning and Azure OpenAI (GPT-4)
- **Hosting and Infrastructure:** Azure App Service
- **DevOps:** Azure DevOps

Timeline and Key Activities

Week 1: Project Kickoff and Initial Setup

1. **Project Kickoff:**
    - Conduct a project kickoff meeting, assign roles and responsibilities, and set up project management tools and communication channels.
2. **Requirement Gathering:**
    - Conduct stakeholder interviews to gather core requirements.
    - Document essential functional requirements.
    - Create basic user stories and use cases.
3. **System Architecture:**
    - Design high-level system architecture using Microsoft technologies.
    - Define key system components.
    - Create initial wireframes for the user interface.

Week 2: Development Setup and Backend Implementation

1. **Development Environment Setup:**

- Set up development environments for the backend and frontend using Visual Studio.
- Configure version control systems with Azure Repos.
- Install necessary development tools and libraries.

2. **Backend Development:**
    - Implement core backend functionalities using ASP.NET Core.
    - Develop data models and database schema in Azure SQL Database.
    - Create API endpoints for property search and listings using canned data.
    - Implement AI-driven valuation tools using Azure Machine Learning with simulated data.

Week 3: Frontend Implementation and Integration

1. **Frontend Development:**
    - Implement core frontend components using Blazor.
    - Develop user interface for property search and listings.
    - Use mock data to simulate real estate properties.
2. **Integration:**
    - Integrate frontend with backend APIs using canned data.
    - Ensure data flow and functionality using simulated data.
    - Deploy backend and frontend to Azure App Service for hosting.

Week 4: Finalization and Demo Preparation

1. **Testing and Bug Fixing:**
    - Conduct basic functionality tests to ensure the MVP works as intended.
    - Fix any critical issues identified during testing.
2. **Demo Preparation:**
    - Prepare demo scripts and presentations.
    - Create marketing materials and documentation for the demo.
    - Conduct internal demo to ensure readiness.
3. **Final Adjustments:**

- Make any final adjustments based on internal feedback.
- Ensure the MVP is polished and ready for presentation.

4. **Investor Presentation:**
    - Present the sandboxed MVP to potential investors.
    - Collect feedback and address any questions.
    - Plan next steps based on investor feedback and interest.
    """

content_prerequisites = """

Detailed Instructions for Development

Prerequisites

Software

- **Visual Studio 2022**
- **.NET 8 SDK**
- **Azure CLI**
- **Postman**
- **Git**

Accounts

- **Azure Subscription**
- **GitHub or Azure Repos Account**
  """

content_setup = """

Development Environment Setup

Step 1: Install Visual Studio 2022

1. Download and install Visual Studio 2022 from the official website.
2. 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 the Azure CLI from the official Azure documentation.
2. Sign in to your Azure account:

```
az login
```

step 3: Set Up Git

1. Download and install Git from the official Git website.
2. Configure Git with your user information:

```
shAlways show details
Copy codegit config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

"""

content_backend = """

Backend Development

Step 1: 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 2: Install Required NuGet Packages

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

```
shAlways show details
Copy codeInstall-Package Microsoft.EntityFrameworkCore
Install-Package Microsoft.EntityFrameworkCore.SqlServer
Install-Package Dapper
Install-Package AutoMapper
Install-Package Microsoft.Data.SqlClient
```

Step 3: Configure Entity Framework Core

1. Add a DbContext class:

```
csharpAlways show details
Copy codepublic class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbCon
text> options)
        : base(options)
    {
    }

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

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

```
jsonAlways show details
Copy code"ConnectionStrings": {
  "DefaultConnection": "Server=tcp:<your_server>.database.window
s.net,1433;Initial Catalog=<your_db>;Persist Security Info=False;
User ID=<your_user>;Password=<your_password>;MultipleActiveResult
```

```
Sets=False;Encrypt=True;TrustServerCertificate=False;Connection T
imeout=30;"
}
```

1. Register the DbContext in `Program.cs`:

```
csharpAlways show details
Copy codepublic void ConfigureServices(IServiceCollection service
s)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("D
efaultConnection")));
    services.AddControllers();
}
```

Step 4: Create API Endpoints

1. Create a controller for property listings:

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

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

    [HttpGet]
    public async Task<ActionResult<IEnumerable<Property>>> GetPro
perties()
```

```
    {
        return await _context.Properties.ToListAsync();
    }
}
```

"""

content_frontend = """

Frontend Development

Step 1: Create Blazor Project

1. Open Visual Studio 2022.
2. Create a new project:
   - Select **Blazor WebAssembly App**.
   - Name the project `Dwellingly.Client`.

Step 2: Install Required NuGet Packages

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

```
shAlways show details
Copy codeInstall-Package Blazorise
Install-Package ChartJs.Blazor
Install-Package MudBlazor
```

Step 3: Configure Blazor Components

1. Configure Blazorise in `Program.cs`:

```
csharpAlways show details
Copy codebuilder.Services.AddBlazorise(options =>
{
    options.ChangeTextOnKeyPress = true;
```

```
})
.AddBootstrapProviders()
.AddFontAwesomeIcons();
```

Step 4: Create Property Listing Component

1. Create a new Razor

```
Always show details
Copy code
```

Define the content for each code file

content_backend = """

Backend Development

Step 1: 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 2: Install Required NuGet Packages

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

sh

Install-Package Microsoft.EntityFrameworkCore
Install-Package Microsoft.EntityFrameworkCore.SqlServer
Install-Package Dapper

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

Step 3: Configure Entity Framework Core

1. Add a DbContext class:

```
csharpAlways show details
Copy codepublic class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbCon
text> options)
        : base(options)
    {
    }

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

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

```
jsonAlways show details
Copy code"ConnectionStrings": {
  "DefaultConnection": "Server=tcp:<your_server>.database.window
s.net,1433;Initial Catalog=<your_db>;Persist Security Info=False;
User ID=<your_user>;Password=<your_password>;MultipleActiveResult
Sets=False;Encrypt=True;TrustServerCertificate=False;Connection T
imeout=30;"
}
```

1. Register the DbContext in `Program.cs`:

```
csharpAlways show details
Copy codepublic void ConfigureServices(IServiceCollection service
s)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("D
```

```
efaultConnection")));
    services.AddControllers();
}
```

Step 4: Create API Endpoints

1. Create a controller for property listings:

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

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

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

"""

content_frontend = """

Frontend Development

Step 1: Create Blazor Project

1. Open Visual Studio 2022.

2. Create a new project:
    - Select **Blazor WebAssembly App**.
    - Name the project `Dwellingly.Client`.

Step 2: Install Required NuGet Packages

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

```
shAlways show details
Copy codeInstall-Package Blazorise
Install-Package ChartJs.Blazor
Install-Package MudBlazor
```

Step 3: Configure Blazor Components

1. Configure Blazorise in `Program.cs`:

```
csharpAlways show details
Copy codebuilder.Services.AddBlazorise(options =>
{
    options.ChangeTextOnKeyPress = true;
})
.AddBootstrapProviders()
.AddFontAwesomeIcons();
```

Step 4: Create Property Listing Component

1. Create a new Razor

```
Always show details
Copy code
```

just tho code file types

Define the content for each code file

content_backend = """

Backend Development

Step 1: 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 2: Install Required NuGet Packages

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

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

tep 3: Configure Entity Framework Core

1. Add a DbContext class:

```
csharpAlways show details
Copy codepublic class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbCon
text> options)
        : base(options)
    {
    }
```

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

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

```
jsonAlways show details
Copy code"ConnectionStrings": {
  "DefaultConnection": "Server=tcp:<your_server>.database.window
s.net,1433;Initial Catalog=<your_db>;Persist Security Info=False;
User ID=<your_user>;Password=<your_password>;MultipleActiveResult
Sets=False;Encrypt=True;TrustServerCertificate=False;Connection T
imeout=30;"
}
```

1. Register the DbContext in `Program.cs`:

```
csharpAlways show details
Copy codepublic void ConfigureServices(IServiceCollection service
s)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("D
efaultConnection")));
    services.AddControllers();
}
```

Step 4: Create API Endpoints

1. Create a controller for property listings:

```
csharpAlways show details
Copy code[ApiController]
[Route("api/[controller]")]
```

```
public class PropertiesController : ControllerBase
{
    private readonly ApplicationDbContext _context;

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

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

"""

content_frontend = """

Frontend Development

Step 1: Create Blazor Project

1. Open Visual Studio 2022.
2. Create a new project:
    - Select **Blazor WebAssembly App**.
    - Name the project `Dwellingly.Client`.

Step 2: Install Required NuGet Packages

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

```
shAlways show details
Copy codeInstall-Package Blazorise
```

```
Install-Package ChartJs.Blazor
Install-Package MudBlazor
```

Step 3: Configure Blazor Components

1. Configure Blazorise in `Program.cs`:

```
csharpAlways show details
Copy codebuilder.Services.AddBlazorise(options =>
{
    options.ChangeTextOnKeyPress = true;
})
.AddBootstrapProviders()
.AddFontAwesomeIcons();
```

Step 4: Create Property Listing Component

1. Create a new Razor component `PropertyList.razor`:

```
razorAlways show details
Copy code@page "/properties"
@inject HttpClient Http

<h3>Property Listings</h3>
<ul>
    @foreach (var property in properties)
    {
        <li>@property.Name - @property.Price</li>
    }
</ul>

@code {
    private List<Property> properties;

    protected override async Task OnInitializedAsync()
```

```
    {
        properties = await Http.GetFromJsonAsync<List<Property>>
("api/properties");
    }
}
```

"""

content_ai_ml = """

AI/ML Integration

Step 1: Set Up Azure Machine Learning

1. Create an Azure Machine Learning workspace in the Azure portal.
2. Install the AzureML SDK for Python:

```
shAlways show details
Copy codepip install azureml-sdk
```

Step 2: Train and Deploy Models

1. Use the AzureML SDK to train and deploy models:

```
pythonAlways show details
Copy codefrom azureml.core import Workspace, Experiment
from azureml.train.automl import AutoMLConfig

ws = Workspace.from_config()
experiment = Experiment(ws, "property-valuation")

automl_config = AutoMLConfig(
    task="regression",
    training_data=train_data,
    label_column_name="price",
    primary_metric="r2_score",
    compute_target=compute_target,
```

```
    max_trials=5
)

run = experiment.submit(automl_config, show_output=True)
```

"""

content_devops = """

Hosting and Deployment

Step 1: Deploy to Azure App Service

1. Right-click the project in Visual Studio and select **Publish**.
2. Choose **Azure** as the target.
3. Follow the prompts to create a new Azure App Service instance and deploy the application.

Step 2: Configure CI/CD with Azure DevOps

1. Create a new project in Azure DevOps.
2. Set up a pipeline using the YAML file:

```
yamlAlways show details
Copy codetrigger:
- 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'
```

"""

Save each content to a separate file

file_paths = {
"backend_development.md": content_backend,
"frontend_development.md": content_frontend,
"ai_ml_integration.md": content_ai_ml,
"devops.md": content_devops
}

for filename, content in file_paths.items():
with open(f"/mnt/data/{filename}", "w") as file:
file.write(content)

file_paths

```
Always show details
Copy code
```

Result

```
{'backend_development.md': '\n### Backend Development\n\n#### Step 1: Create ASP.NET Core Project\n\n1. Open Visual Studio 2022.\n2. Create a new project:\n   - Select **ASP.NET Core Web API**.\n   - Choose **.NET 8.0** as the target framework.\n   - Name the project `Dwellingly.API`.\n\n#### Step 2: Install Required NuGet Packages\n\n1. Open the NuGet Package Manager Console.\n2. Install the following packages:\n\n```sh\nInstall-Package Microsoft.EntityFrameworkCore\nInstall-Package Microsoft.EntityFrameworkCore.SqlServer\nInstall-Package Dapper\nInstall-Package AutoMapper\nInstall-Package Microsoft.Data.SqlClient\n```\n\n#### Step 3: Configure Entity Framework Core\n\n1. Add a DbContext class:\n\n```csharp\npublic class ApplicationDbContext : DbContext\n{\n    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)\n        : base(options)\n    {\n    }\n\n    public DbSet<Property> Properties { get; set; }\n}\n```\n\n2. Configure the connection string in `appsettings.json`:\n\n```json\n"ConnectionStrings": {\n  "DefaultConnection":
```

```
"Server=tcp:<your_server>.database.windows.net,1433;Initial Catalog=<your_db>;Persist Security Info=False;User ID=<your_user>;Password=<your_password>;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;"\n}\n```\n\n3. Register the DbContext in `Program.cs`:\n\n```csharp\npublic void ConfigureServices(IServiceCollection services)\n{\n    services.AddDbContext<ApplicationDbContext>(options =>\n        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));\n    services.AddControllers();\n}\n```\n\n#### Step 4: Create API Endpoints\n\n1. Create a controller for property listings:\n\n```csharp\n[ApiController]\n[Route("api/[controller]")]\npublic class PropertiesController : ControllerBase\n{\n    private readonly ApplicationDbContext _context;\n\n    public PropertiesController(ApplicationDbContext context)\n    {\n        _context = context;\n    }\n\n    [HttpGet]\n    public async Task<ActionResult<IEnumerable<Property>>> GetProperties()\n    {\n        return await _context.Properties.ToListAsync();\n    }\n}\n```\n\n',
 'frontend_development.md': '\n### Frontend Development\n\n#### Step 1: Create Blazor Project\n\n1. Open Visual Studio 2022.\n2. Create a new project:\n   - Select **Blazor WebAssembly App**.\n   - Name the project `Dwellingly.Client`.\n\n#### Step 2: Install Required NuGet Packages\n\n1. Open the NuGet Package Manager Console.\n2. Install the following packages:\n\n```sh\nInstall-Package Blazorise\nInstall-Package ChartJs.Blazor\nInstall-Package MudBlazor\n```\n\n#### Step 3: Configure Blazor Components\n\n1. Configure Blazorise in `Program.cs`:\n\n```csharp\nbuilder.Services.AddBlazorise(options =>\n{\n    options.ChangeTextOnKeyPress = true;\n})\n.AddBootstrapProviders()\n.AddFontAwesomeIcons();\n```\n\n#### Step 4: Create Property Listing Component\n\n1. Create a new Razor component `PropertyList.razor`:\n\n```razor\n@page "/properties"\n@inject HttpClient Http\n\n<h3>Property Listings</h3>\n<ul>\n    @foreach (var property in properties)\n    {\n        <li>@property.Name - @property.Price</li>\n    }\n</ul>\n\n@code {\n    private List<Property> properties;\n\n    protected override async Task OnInitializedAsync()\n    {\n        properties = await Http.GetFromJsonAsync<List<Property>>("api/properties");\n    }\n}\n```\n\n',
 'ai_ml_integration.md': '\n### AI/ML Integration\n\n#### Step 1: Set Up Azure Machine Learning\n\n1. Create an Azure Machine Learning workspace in the Azure portal.\n2. Install the AzureML SDK for Python:\n\n```sh\npip install azureml-sdk\n```\n\n#### Step 2: Train and Deploy Models\n\n1. Use the AzureML SDK to train and deploy models:\n\n```python\nfrom azureml.core import Workspace, Experiment\nfrom azureml.train.automl import AutoMLConfig\n\nws = Workspace.from_config()\nexperiment = Experiment(ws, "property-valuation")\n\nautoml_config = AutoMLConfig(\n    task="regression",\n    training_data=train_data,\n    label_column_name="price",\n    primary_metric="r2_score",\n    compute_target=compute_target,\n    max_trials=5\n)\n\nrun = experiment.submit(automl_config, show_output=True)\n```\n\n',
 'devops.md': "\n### Hosting and Deployment\n\n#### Step 1: Deploy to
```

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>(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. **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.