Phase 3: Develop Conversational AI Chatbot and Virtual Assistant

Table of Contents

- Objectives
- Step-by-Step Implementation Plan
  - Design Chatbot Architecture and Knowledge Base
  - Natural Language Processing (NLP) Model Development
  - Chatbot Development and Integration
  - Testing and Refinement
- Summary of Phase 3

Objectives

- Design and develop a conversational AI chatbot to assist users with property searches, valuations, and recommendations.
- Integrate the chatbot with the NexHomeAgent application.

- Ensure the chatbot can handle common user queries effectively and provide accurate information.

Step-by-Step Implementation Plan

1. Design Chatbot Architecture and Knowledge Base

1. **Define Capabilities:**
    - Outline the main functionalities for the chatbot, including property searches, property valuations, recommendations, and general inquiries.
2. **Create Knowledge Base:**
    - Develop a structured knowledge base containing frequently asked questions (FAQs), property data, valuation information, and other relevant details.
3. **Design Conversation Flow:**
    - Use tools like Microsoft Visio or PowerPoint to design the conversation flow, including user intents, responses, and fallback mechanisms.

2. Natural Language Processing (NLP) Model Development

1. **Select NLP Platform:**
    - Choose an NLP platform such as Microsoft Azure Bot Service, Dialogflow, or Rasa.
2. **Develop NLP Models:**
    - Create and train NLP models for intent recognition, entity extraction, and context management.
3. **Train Models:**
    - Use existing property data, user interactions, and FAQs to train the models for accuracy.

```
# nlp_model.py

from transformers import pipeline
```

```
# Load pre-trained NLP model for intent recognition
nlp = pipeline('sentiment-analysis')

def get_intent(text):
    result = nlp(text)
    return result[0]['label']

if __name__ == "__main__":
    text = "Can you show me properties in New York?"
    intent = get_intent(text)
    print(f"Intent: {intent}")
```

3. Chatbot Development and Integration

1. **Implement Chatbot:**
    - Develop the chatbot using the selected NLP platform and integrate it with the NexHomeAgent backend.
2. **Develop APIs for Chatbot Integration:**
    - Create API endpoints for the chatbot to interact with the backend services for property searches, valuations, and recommendations.

```
# chatbot.py

from flask import Flask, request, jsonify
from nlp_model import get_intent

app = Flask(__name__)

@app.route('/chatbot', methods=['POST'])
def chatbot():
    data = request.get_json()
    user_message = data.get('message')
    intent = get_intent(user_message)
```

```
    if intent == 'property_search':
        # Implement property search logic
        response = search_properties(user_message)
    elif intent == 'property_valuation':
        # Implement property valuation logic
        response = get_property_valuation(user_message)
    else:
        response = {"message": "I'm sorry, I didn't understan
d that."}

    return jsonify(response)

def search_properties(message):
    # Implement property search based on message
    return {"properties": []}

def get_property_valuation(message):
    # Implement property valuation based on message
    return {"valuation": 0}

if __name__ == '__main__':
    app.run(debug=True)
```

4. Testing and Refinement

1. **Conduct User Testing:**
    - Perform user testing to evaluate the chatbot's performance, usability, and accuracy.
2. **Refine the Chatbot:**
    - Based on user feedback, refine the chatbot's responses, improve NLP models, and enhance the conversation flow.
3. **Continuous Improvement:**

- Implement a feedback loop to continuously improve the chatbot based on real user interactions.

```
# test_chatbot.py

import unittest
from chatbot import app

class TestChatbot(unittest.TestCase):
    def setUp(self):
        self.app = app.test_client()
        self.app.testing = True

    def test_property_search(self):
        response = self.app.post('/chatbot', json={'message':
'Show me properties in New York'})
        self.assertEqual(response.status_code, 200)
        self.assertIn('properties', response.get_json())

    def test_property_valuation(self):
        response = self.app.post('/chatbot', json={'message':
'What is the valuation of property ID 123?'})
        self.assertEqual(response.status_code, 200)
        self.assertIn('valuation', response.get_json())

if __name__ == '__main__':
    unittest.main()
```

Summary of Phase 3

1. **Design Chatbot Architecture and Knowledge Base:**
    - Define the chatbot's capabilities and create a structured knowledge base.
2. **Natural Language Processing (NLP) Model Development:**

- Select an NLP platform and develop models for intent recognition and entity extraction.

3. **Chatbot Development and Integration:**
    - Implement the chatbot, develop necessary APIs, and integrate it with the backend services.
4. **Testing and Refinement:**
    - Conduct user testing, refine the chatbot based on feedback, and ensure continuous improvement.

By following these steps, you will develop a robust and effective conversational AI chatbot for the NexHomeAgent application, enhancing user experience and providing valuable assistance to users. If you encounter any issues or need further modifications, please let me know!

SPEC-1: NexHomeAgent | AI Sandboxed MVP Development

NexHomeAgent | AI Sandboxed MVP Development

SPEC-1: NexHomeAgent | AI Sandboxed MVP Development

= SPEC-1: NexHomeAgent | AI Sandboxed MVP Development :sectnums: :toc:

== Background

The real estate industry has increasingly leaned towards leveraging technology to streamline operations and improve user experiences. NexHomeAgent aims to harness AI capabilities to provide a seamless, efficient, and user-friendly platform for property buyers. By integrating advanced features like property search, Comparative Market Analysis (CMA), and automated property valuations, NexHomeAgent seeks to stand out in a competitive market. This Minimum Viable Product (MVP) serves as a prototype to attract investors and demonstrate the potential of the full-scale application. The decision to use Microsoft technologies is driven by their robustness, scalability, and comprehensive suite of tools, which align well with the project requirements.

== Requirements

The following are the functional and non-functional requirements for the NexHomeAgent | AI Sandboxed MVP, categorized using the MoSCoW prioritization method.

=== Must Have

- Ability to search for properties based on user-defined criteria such as location, price range, and number of bedrooms.
- Display detailed property information, including photos, descriptions, features, and price history.
- Allow users to save properties to a favorites list.
- Schedule viewing appointments for properties.
- Enable users to contact real estate agents through the platform.
- Request and generate Comparative Market Analysis (CMA) reports for properties.
- Submit offers on properties through the platform.
- Integration of AI-driven property recommendations and automated property valuation.
- Basic user authentication and profile management.
- Deployment and hosting on Azure App Service.

=== Should Have

- User dashboard to view saved properties, scheduled viewings, and recent activities.
- AI-driven price prediction and trend analysis.
- Buyer assistance chatbot to handle common queries and assist with property searches and scheduling.

=== Could Have

- Enhanced user behavior analytics for personalized content and recommendations.
- Detailed user profile management, including preferences and account settings.

=== Won't Have

- Advanced AI features not critical for the MVP.
- Full-scale deployment and integration with third-party services for property listings.

Method

== Method

To address the requirements of the NexHomeAgent | AI Sandboxed MVP, we will implement a robust architecture using Microsoft technologies. The key components and their interactions are detailed below.

=== System Architecture

The system architecture consists of the following components:

[plantuml] @startuml !define RECTANGLE class RECTANGLE AzureSQL { Database } RECTANGLE ASPNetCore { BackendAPI AIValuationEngine } RECTANGLE Blazor { Frontend } RECTANGLE AzureMachineLearning { RecommendationEngine CMAGenerator } RECTANGLE AzureAppService { Hosting }

AzureSQL -[hidden]-> ASPNetCore AzureSQL -[hidden]-> Blazor AzureSQL -[hidden]-> AzureMachineLearning ASPNetCore -down-> AzureSQL: "Database Queries" ASPNetCore -right-> AzureMachineLearning: "AI/ML Models" Blazor -up-> ASPNetCore: "API Calls" Blazor -right-> AzureAppService: "Deployment" @enduml

=== Database Schema

The Azure SQL Database will store essential data for properties, users, favorites, CMAs, and offers. The following tables are proposed:

Properties: [cols="1,1,1,1,1", options="header"] |=== | Field | Type | Description | id | INT | Primary Key | address | VARCHAR(255) | Property Address | price | DECIMAL | Property Price | bedrooms | INT | Number of Bedrooms | bathrooms | INT | Number of Bathrooms | description | TEXT | Property Description | features | TEXT | Property Features | photos | JSON | Property Photos |=== *Users*: [cols="1,1,1,1,1", options="header"] |=== | Field | Type | Description | id | INT | Primary Key | name | VARCHAR(255) | User Name | email | VARCHAR(255) | User Email | password | VARCHAR(255) | User Password (hashed) |=== *Favorites*: [cols="1,1,1,1,1", options="header"] |=== | Field | Type | Description | user_id | INT | Foreign Key to Users | property_id | INT | Foreign Key to Properties |=== *Viewings*: [cols="1,1,1,1,1", options="header"] |=== | Field | Type | Description | id | INT | Primary Key | user_id | INT | Foreign Key to Users | property_id | INT | Foreign Key to Properties | date_time | DATETIME | Scheduled Viewing Date and Time |=== *CMAs*: [cols="1,1,1,1,1", options="header"] |=== | Field | Type | Description | id | INT | Primary Key | property_id | INT | Foreign Key to Properties | report | JSON | CMA Report Data |=== *Offers*: [cols="1,1,1,1,1", options="header"] |=== | Field | Type | Description | id | INT | Primary Key | user_id | INT | Foreign Key to Users | property_id | INT | Foreign Key to Properties | offer_details| JSON | Offer Details |===

=== Key Algorithms

==== Property Search

The property search functionality will utilize SQL queries to filter properties based on user-defined criteria. The query will dynamically build WHERE clauses based on provided search parameters.

==== AI-Driven Valuation

The AI-driven valuation tool will use models hosted on Azure Machine Learning to estimate property values. The valuation model will consider factors like recent sales, property features, and market trends.

==== Comparative Market Analysis (CMA)

The CMA generation will leverage Azure Machine Learning to analyze comparable sales data and produce a detailed report. The process will include fetching relevant data, running the analysis model, and generating a structured report.

Implementation

== Implementation

The following steps outline the implementation process for the NexHomeAgent | AI Sandboxed MVP. This plan ensures that the development aligns with the proposed architecture and meets the specified requirements within the one-month timeframe.

=== Week 1: Project Kickoff and Initial Setup

Day 1-2: Project Kickoff

- Conduct the kickoff meeting to align on objectives and deliverables.
- Assign roles and responsibilities.
- Set up project management tools (e.g., Azure DevOps Boards).

Day 3-5: Requirements Gathering

- Conduct stakeholder interviews to gather core requirements.
- Document functional requirements.
- Create user stories and use cases.

Day 6-7: System Design

- Design high-level system architecture.

- Define key system components.
- Create wireframes using Microsoft PowerPoint or Visio.

=== Week 2: Development Setup and Backend Implementation

Day 8-9: Setup Development Environment

- Set up Visual Studio for development.
- Configure Azure Repos for version control.
- Install necessary libraries and tools.

Day 10-14: Backend Development

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

=== Week 3: Frontend Implementation and Integration

Day 15-17: Frontend Development

- Implement frontend components using Blazor.
- Develop UI for property search and listings.
- Use mock data for simulation.

Day 18-20: Integration

- Integrate frontend with backend APIs.
- Ensure smooth data flow using simulated data.

Day 21: Hosting Setup

- Deploy the application to Azure App Service.

=== Week 4: Finalization and Demo Preparation

Day 22-24: Basic Testing

- Conduct functionality tests.
- Fix critical issues.

Day 25-26: Demo Preparation

- Prepare demo scripts and presentations.
- Create marketing materials.

Day 27: Internal Demo

- Conduct internal demo to ensure readiness.

Day 28-29: Final Adjustments

- Make final adjustments based on feedback.

Day 30: Demo Launch

- Present the MVP to investors.
- Collect feedback and plan next steps.

Milestones

== Milestones

The following milestones will help track the progress of the NexHomeAgent | AI Sandboxed MVP development. Each milestone corresponds to a major phase or deliverable in the project timeline.

=== Milestone 1: Project Kickoff and Initial Setup (End of Week 1)

- Completion of project kickoff meeting.
- Assignment of roles and responsibilities.
- Setup of project management tools.
- Documentation of functional requirements.
- Creation of user stories and use cases.
- High-level system architecture design.
- Initial wireframes completed.

=== Milestone 2: Development Setup and Backend Implementation (End of Week 2)

- Development environments set up.
- Version control configured with Azure Repos.

- Core backend functionalities implemented.
- Database schema developed in Azure SQL Database.
- API endpoints for property search and listings created.
- AI-driven valuation tools implemented using Azure Machine Learning.

=== Milestone 3: Frontend Implementation and Integration (End of Week 3)

- Frontend components implemented using Blazor.
- User interface for property search and listings developed.
- Integration of frontend with backend APIs.
- Successful data flow and functionality using simulated data.
- Application deployed to Azure App Service.

=== Milestone 4: Finalization and Demo Preparation (End of Week 4)

- Basic functionality tests conducted and critical issues fixed.
- Demo scripts and presentations prepared.
- Marketing materials and documentation created.
- Internal demo conducted and feedback incorporated.
- Final adjustments made based on internal feedback.
- MVP ready for investor presentation.

=== Milestone 5: Demo Launch (End of Week 4)

- Successful presentation of the MVP to potential investors.
- Collection of feedback from investors.
- Planning of next steps based on investor feedback and interest.

Gathering Results

== Gathering Results

To evaluate the success of the NexHomeAgent | AI Sandboxed MVP and ensure it meets the initial requirements and goals, the following steps will be undertaken:

=== Evaluation Metrics

The following metrics will be used to assess the performance and effectiveness of the MVP:

User Engagement Metrics

- Number of users registered.
- Number of property searches conducted.
- Number of properties viewed.
- Number of properties saved to favorites.
- Number of viewings scheduled.
- Number of CMA requests submitted.
- Number of offers submitted.

System Performance Metrics

- Response time for property search.
- Load time for property details pages.
- System uptime and availability.
- Performance of AI-driven valuation and recommendation tools.

User Feedback

- User satisfaction ratings collected through surveys.
- Feedback from internal demos.
- Feedback from potential investors during the demo launch.

=== Post-Launch Review

A comprehensive review will be conducted post-launch to analyze the following:

Requirement Fulfillment

- Verification that all "Must Have" requirements are met.
- Assessment of "Should Have" and "Could Have" requirements based on development progress and feedback.

Performance Analysis

- Detailed analysis of system performance metrics.
- Identification and resolution of any critical performance issues.

User Feedback Analysis

- Compilation and analysis of user feedback.
- Identification of common user issues or requests for future development.

Investor Feedback

- Collection and analysis of feedback from potential investors.
- Evaluation of interest and potential funding opportunities.

=== Next Steps

Based on the evaluation, the next steps will be planned to improve and expand the NexHomeAgent platform:

Development Roadmap

- Prioritization of additional features and improvements.
- Planning for scaling the platform to handle increased user load.

Funding and Investment

- Follow-up with interested investors.
- Preparation of detailed proposals and business plans to secure funding for full-scale development.

Continuous Improvement

- Implementation of feedback-driven changes and enhancements.
- Regular updates and maintenance to ensure optimal performance and user satisfaction.

Phase 4 testing n docs

Development Plan for Phase 4: Testing, Documentation, and Deployment

Objectives

- Conduct thorough testing of the AI models and their integration.
- Provide comprehensive documentation for the AI features.
- Deploy the AI features to a staging environment for final testing.
- Ensure proper monitoring and logging are in place.
- Develop user training materials and provide support.

Detailed Development Plan

Week 1: Testing and Evaluation

Tasks:

1. **Develop Test Cases:**
    - Write detailed test cases to evaluate the AI models and their integration with the application.
    - Include unit tests, integration tests, and end-to-end tests.
2. **Run Tests:**

- Execute the test cases to ensure the AI models and integrations work as expected.
- Identify and fix any issues found during testing.

Milestones:

- Completion of all test cases.
- Successful execution of all test cases without errors.

Week 2: Documentation

Tasks:

1. **Document AI Integration:**
    - Provide comprehensive documentation for the AI features, including the purpose, architecture, and usage.
    - Document the steps for setting up the environment, running tests, and deploying the application.
2. **Prepare Technical Documentation:**
    - Create detailed technical documentation for developers, including API specifications, data flow diagrams, and configuration settings.

Milestones:

- Completion of user and technical documentation.
- Review and approval of documentation by stakeholders.

Week 3: Deployment and Monitoring

Tasks:

1. **Deploy to Staging:**
    - Push the changes to a staging environment for final testing before production deployment.
    - Ensure the staging environment mirrors the production environment as closely as possible.
2. **Implement Monitoring:**

- Set up logging and monitoring to track the performance and usage of the AI features.
- Ensure real-time alerts for any issues detected in the staging environment.

3. **Final Testing in Staging:**
    - Conduct final testing in the staging environment to ensure everything works as expected.
    - Fix any issues identified during final testing.

Milestones:

- Successful deployment to the staging environment.
- Monitoring and logging set up and verified.
- Final testing completed without critical issues.

Week 4: User Training and Support

Tasks:

1. **Develop Training Materials:**
    - Create user guides and training materials to help users understand and utilize the new AI features effectively.
    - Prepare training sessions or webinars if necessary.
2. **Provide Support:**
    - Set up support channels for users to report issues and provide feedback on the AI features.
    - Ensure a process for tracking and resolving user-reported issues.
3. **Collect Feedback:**
    - Gather feedback from users during training and initial usage.
    - Use feedback to make any final adjustments before full production deployment.

Milestones:

- Completion of user training materials.
- Successful user training sessions conducted.

- Support channels set up and operational.
- Feedback collected and reviewed.

Summary of Development Plan

1. **Week 1: Testing and Evaluation**
    - Develop and run detailed test cases.
    - Ensure all AI models and integrations work as expected.
2. **Week 2: Documentation**
    - Provide comprehensive user and technical documentation.
    - Ensure all documentation is reviewed and approved.
3. **Week 3: Deployment and Monitoring**
    - Deploy to staging and conduct final testing.
    - Set up and verify monitoring and logging.
4. **Week 4: User Training and Support**
    - Develop training materials and conduct training sessions.
    - Set up support channels and collect user feedback.

By following this development plan, you will ensure a smooth and successful Phase 4, leading to a robust, well-documented, and user-friendly integration of AI features into the NexHomeAgent application.

Phase 4: Testing, Documentation, and Deployment

Objectives

- Conduct thorough testing of the AI models and their integration.
- Provide comprehensive documentation for the AI features.
- Deploy the AI features to a staging environment for final testing.
- Ensure proper monitoring and logging are in place.
- Develop user training materials and provide support.

Step-by-Step Implementation Plan

1. Testing and Evaluation

1. **Develop Test Cases:**
    - Write detailed test cases to evaluate the AI models and their integration with the application.
    - Include unit tests, integration tests, and end-to-end tests.

```
# test_cases.py

import unittest
from flask import Flask
from flask.testing import FlaskClient
from model_development import train_model
from data_preparation import load_data, feature_engineering
import pandas as pd

class TestAIFeatures(unittest.TestCase):
    def setUp(self):
        self.app = Flask(__name__).test_client()
        self.app.testing = True
        self.data = pd.read_csv('prepared_data.csv')
        self.X = self.data.drop(columns=['price'])
        self.y = self.data['price']
        self.model = train_model(self.X, self.y)

    def test_model_accuracy(self):
        metrics = evaluate_model(self.model, self.X, self.y)
        self.assertGreater(metrics['accuracy'], 0.8)

    def test_data_loading(self):
        data_sources = load_data()
        self.assertEqual(len(data_sources), 8)
```

```
    def test_feature_engineering(self):
        df = pd.DataFrame({'price': [100, 200], 'sqft': [100
0, 1500], 'year_built': [2000, 2010]})
        engineered_df = feature_engineering(df)
        self.assertIn('price_per_sqft', engineered_df.column
s)
        self.assertIn('age_of_property', engineered_df.column
s)

    def test_chatbot_response(self):
        response = self.app.post('/chatbot', json={'message':
'Show me properties in New York'})
        self.assertEqual(response.status_code, 200)
        self.assertIn('properties', response.get_json())

if __name__ == '__main__':
    unittest.main()
```

1. **Run Tests:**
   - Execute the test cases to ensure the AI models and integrations work as expected.

```
python -m unittest test_cases.py
```

2. Documentation

1. **Document AI Integration:**
   - Provide comprehensive documentation for the AI features, including the purpose, architecture, and usage.

```
# AI Integration Documentation

## Property Valuation Model
- **Purpose:** To provide accurate property valuations using
advanced machine learning algorithms.
- **Model:** XGBoost
```

```
- **Data Sources:** Neighborhood demographics, school district info, market trends, property records, public records, economic indicators, social data, geospatial data.
- **Features:** Price per sqft, age of property, etc.

## Recommendation System
- **Purpose:** To provide personalized property recommendations.
- **Algorithm:** Nearest Neighbors and LightGBM
- **Data:** User behavior, property features

## Conversational AI Chatbot
- **Purpose:** To assist users with property searches, valuations, and recommendations.
- **Platform:** Flask, NLP models using transformers
- **Capabilities:** Property searches, valuations, general inquiries
```

3. Deployment and Monitoring

1. **Deploy to Staging:**
    - Push the changes to a staging environment for final testing before production deployment.

```
# Deploying to staging environment
az webapp up --name NexHomeAgentStaging --resource-group NexHomeAgentGroup --plan NexHomeAgentPlan
```

1. **Implement Monitoring:**
    - Set up logging and monitoring to track the performance and usage of the AI features.

```
# monitoring.py

import logging
```

```
logging.basicConfig(level=logging.INFO)

def setup_logging():
    logger = logging.getLogger(__name__)
    return logger

if __name__ == "__main__":
    logger = setup_logging()
    logger.info("Monitoring setup complete")
```

4. User Training and Support

1. **Develop Training Materials:**
    - Create user guides and training materials to help users understand and utilize the new AI features effectively.

```
# User Guide for NexHomeAgent AI Features

## Property Valuation
- **How to use:** Enter the property details and get an accur
ate valuation based on multiple factors.

## Property Recommendations
- **How to use:** Get personalized property recommendations b
ased on your preferences and browsing history.

## Conversational AI Chatbot
- **How to use:** Ask the chatbot for property searches, valu
ations, and general inquiries. Examples:
  - "Show me properties in New York."
  - "What is the valuation of property ID 123?"
```

1. **Provide Support:**
    - Set up support channels for users to report issues and provide feedback on the AI features.

Summary of Phase 4

1. **Testing and Evaluation:**
    - Develop and run detailed test cases to ensure the AI models and integrations work as expected.
2. **Documentation:**
    - Provide comprehensive documentation for the AI features, including the purpose, architecture, and usage.
3. **Deployment and Monitoring:**
    - Deploy the AI features to a staging environment for final testing, and set up logging and monitoring.
4. **User Training and Support:**
    - Develop training materials and provide support channels for users.

By following these steps, you will ensure the AI features are thoroughly tested, well-documented, properly deployed, and effectively supported, leading to a successful integration into the NexHomeAgent application. If you encounter any issues or need further modifications, please let me know!

Chat Bot interface-final test

Let's break down and ensure the complete setup, including the necessary configuration and integration steps for the AI chat interface within the NexHomeAgent application.

1. Frontend: Blazor Component for Chat Interface

Create a new Blazor component named `AiChat.razor` for the chat interface.

AiChat.razor

```
@page "/chat"
@inject IChatService ChatService

<div class="chat-container">
    <div class="chat-header" @onclick="ToggleChat">
        <img src="avatar.png" alt="Chat Avatar" class="chat-a
vatar" />
        <span>How can I assist you today?</span>
    </div>
    @if (isChatOpen)
    {
        <div class="chat-body">
            <div class="chat-messages">
                @foreach (var message in messages)
                {
                    <div class="message @(message.IsUser ? "u
ser-message" : "ai-message")">
                        @message.Text
                    </div>
                }
            </div>
            <div class="chat-input">
                <InputText @bind-Value="userInput" placeholde
r="Type your message..." class="input-field" />
                <button @onclick="SendMessage" class="send-bu
```

```
tton">Send</button>
            </div>
        </div>
    }
</div>

@code {
    private bool isChatOpen = false;
    private string userInput = string.Empty;
    private List<ChatMessage> messages = new List<ChatMessage
>();

    private void ToggleChat()
    {
        isChatOpen = !isChatOpen;
    }

    private async Task SendMessage()
    {
        if (!string.IsNullOrWhiteSpace(userInput))
        {
            messages.Add(new ChatMessage { Text = userInput,
IsUser = true });
            var response = await ChatService.SendMessageAsync
(userInput);
            messages.Add(new ChatMessage { Text = response, I
sUser = false });
            userInput = string.Empty;
        }
    }

    public class ChatMessage
    {
        public string Text { get; set; }
        public bool IsUser { get; set; }
```

```
    }
}
```

2. Backend: Integration with OpenAI API

Ensure you have configured your OpenAI API key securely. Use Azure Key Vault or environment variables to store sensitive information.

ChatService.cs

```
using System.Net.Http;
using System.Net.Http.Json;
using System.Threading.Tasks;
using System.Linq;

public interface IChatService
{
    Task<string> SendMessageAsync(string message);
}

public class ChatService : IChatService
{
    private readonly HttpClient _httpClient;

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

    public async Task<string> SendMessageAsync(string messag
e)
    {
        var response = await _httpClient.PostAsJsonAsync("<ht
tps://api.openai.com/v1/engines/davinci-codex/completions>",
new
        {
```

```
            prompt = message,
            max_tokens = 150
        });

        var result = await response.Content.ReadFromJsonAsync
<OpenAiResponse>();
        return result.Choices.FirstOrDefault()?.Text.Trim();
    }

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

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

3. Configuration

appsettings.json

```
{
  "OpenAI": {
    "ApiKey": "YOUR_API_KEY"
  }
}
```

Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddHttpClient<IChatService, ChatService>(client
=>
```

```
    {
        client.BaseAddress = new Uri("<https://api.openai.co
m/v1/>");
        client.DefaultRequestHeaders.Add("Authorization", "Be
arer " + Configuration["OpenAI:ApiKey"]);
    });

    services.AddRazorPages();
    services.AddServerSideBlazor();
}
```

4. Styling (CSS)

wwwroot/css/site.css

```css
.chat-container {
    position: fixed;
    bottom: 20px;
    right: 20px;
    width: 300px;
    background: rgba(255, 255, 255, 0.5);
    border-radius: 10px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    overflow: hidden;
}

.chat-header {
    display: flex;
    align-items: center;
    padding: 10px;
    cursor: pointer;
}

.chat-avatar {
    width: 40px;
    height: 40px;
```

```
    border-radius: 50%;
    margin-right: 10px;
}

.chat-body {
    max-height: 400px;
    overflow-y: auto;
}

.chat-messages {
    padding: 10px;
}

.message {
    padding: 8px 10px;
    margin: 5px 0;
    border-radius: 20px;
}

.user-message {
    background: #007bff;
    color: white;
    text-align: right;
}

.ai-message {
    background: #f1f1f1;
    text-align: left;
}

.chat-input {
    display: flex;
    padding: 10px;
    border-top: 1px solid #ddd;
}
```

```css
.input-field {
    flex-grow: 1;
    border: none;
    padding: 10px;
    border-radius: 20px;
    margin-right: 10px;
}

.send-button {
    background: #007bff;
    color: white;
    border: none;
    padding: 10px 20px;
    border-radius: 20px;
    cursor: pointer;
}
```

5. Azure Configuration

Azure Key Vault

1. **Create Azure Key Vault:**

```
az keyvault create --name YourKeyVaultName --resource-grou
p YourResourceGroup --location YourLocation
```

2. **Store OpenAI API Key:**

```
az keyvault secret set --vault-name YourKeyVaultName --nam
e OpenAI-ApiKey --value "YOUR_OPENAI_API_KEY"
```

3. **Configure Azure App Service to use Key Vault:**

```
az webapp identity assign --resource-group YourResourceGro
up --name NexHomeAgentApp
```

```
az keyvault set-policy --name YourKeyVaultName --resource-
group YourResourceGroup --object-id <identityPrincipalId>
--secret-permissions get
```

4. **Configure App Settings in Azure App Service:**

```
az webapp config appsettings set --resource-group YourReso
urceGroup --name NexHomeAgentApp --settings "KeyVaultName=
YourKeyVaultName"
```

6. Conclusion

This setup provides a modern, user-friendly AI chat interface integrated with a lifelike avatar, using OpenAI for intelligent responses and incorporating the latest UI trends. This ensures that the NexHomeAgent application offers a state-of-the-art user experience. If you have any further requirements or need additional features, feel free to ask!

To run the NexHomeAgent application with the AI chat interface, you'll need to install several dependencies. These dependencies include libraries and packages for .NET, Blazor, and Azure integration. Here's a comprehensive list of the dependencies and libraries you'll need to install:

.NET Core SDK

First, ensure you have the .NET Core SDK installed. You can download it from the official .NET website.

NuGet Packages for Blazor

Install the necessary NuGet packages for Blazor and ASP.NET Core.

```
dotnet add package Microsoft.AspNetCore.Components.WebAssembl
y
dotnet add package Microsoft.AspNetCore.Components.WebAssembl
y.Authentication
dotnet add package Microsoft.AspNetCore.Components.WebAssembl
y.DevServer
```

```
dotnet add package Microsoft.Extensions.Http
dotnet add package Microsoft.Extensions.Configuration
dotnet add package Microsoft.Extensions.Configuration.Json
dotnet add package Microsoft.Extensions.Configuration.AzureKe
yVault
dotnet add package Microsoft.Azure.KeyVault
dotnet add package Microsoft.Azure.Services.AppAuthentication
dotnet add package System.Net.Http.Json
dotnet add package BlazorInputFile
dotnet add package Microsoft.AspNetCore.SignalR.Client
dotnet add package Microsoft.Identity.Web
```

Azure CLI

Install the Azure CLI to manage Azure resources and configure your application.

```
# Windows
Invoke-WebRequest -Uri <https://aka.ms/installazurecliwindows
> -OutFile .\\AzureCLI.msi; Start-Process msiexec.exe -Argume
ntList '/I AzureCLI.msi /quiet' -Wait

# macOS
brew update && brew install azure-cli

# Ubuntu
curl -sL <https://aka.ms/InstallAzureCLIDeb> | sudo bash
```

Setting up the Environment

Configure environment variables for your application. Create a `.env` file in the root directory of your project to store sensitive information like your OpenAI API key and Azure Key Vault details.

.env

```
OpenAI_ApiKey=YOUR_OPENAI_API_KEY
Azure_KeyVaultName=YourKeyVaultName
```

```
Azure_ClientId=YourAzureClientId
Azure_ClientSecret=YourAzureClientSecret
Azure_TenantId=YourAzureTenantId
```

Middleware Setup

Make sure your `Startup.cs` file includes the necessary middleware and services configuration:

Startup.cs

```
public class Startup
{
    public IConfiguration Configuration { get; }

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

    public void ConfigureServices(IServiceCollection service
s)
    {
        services.AddHttpClient<IChatService, ChatService>(cli
ent =>
        {
            client.BaseAddress = new Uri("<https://api.opena
i.com/v1/>");
            client.DefaultRequestHeaders.Add("Authorization",
"Bearer " + Configuration["OpenAI:ApiKey"]);
        });

        services.AddAzureKeyVault(
            $"https://{Configuration["Azure_KeyVaultName"]}.v
ault.azure.net/",
            Configuration["Azure_ClientId"],
            Configuration["Azure_ClientSecret"]
```

```
        );

        services.AddRazorPages();
        services.AddServerSideBlazor();
    }

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

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

        app.UseAuthentication();
        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapBlazorHub();
            endpoints.MapFallbackToPage("/_Host");
        });
    }
}
```

Azure Key Vault Configuration

Ensure you have the Azure Key Vault properly configured to securely store and access sensitive data.

1. **Create Azure Key Vault:**

```
az keyvault create --name YourKeyVaultName --resource-grou
p YourResourceGroup --location YourLocation
```

2. **Store OpenAI API Key:**

```
az keyvault secret set --vault-name YourKeyVaultName --nam
e OpenAI-ApiKey --value "YOUR_OPENAI_API_KEY"
```

3. **Configure Azure App Service to use Key Vault:**

```
az webapp identity assign --resource-group YourResourceGro
up --name NexHomeAgentApp
```

```
az keyvault set-policy --name YourKeyVaultName --resource-
group YourResourceGroup --object-id <identityPrincipalId>
--secret-permissions get
```

4. **Configure App Settings in Azure App Service:**

```
az webapp config appsettings set --resource-group YourReso
urceGroup --name NexHomeAgentApp --settings "KeyVaultName=
YourKeyVaultName"
```

Running the Application

Once all dependencies are installed and the environment is configured, you can run the application using the following command:

```
dotnet run
```

README File

Create a `README.md` file in your project root directory to provide instructions for setting up and running the application.

README.md

````
# NexHomeAgent Application

## Overview

This project is a web application built using Blazor and .NET Core, integrated with OpenAI for an AI chat interface. The application leverages Azure services for secure key management and deployment.

## Prerequisites

- .NET Core SDK
- Azure CLI
- NuGet Packages (listed below)

## NuGet Packages

```bash
dotnet add package Microsoft.AspNetCore.Components.WebAssembly
dotnet add package Microsoft.AspNetCore.Components.WebAssembly.Authentication
dotnet add package Microsoft.AspNetCore.Components.WebAssembly.DevServer
dotnet add package Microsoft.Extensions.Http
dotnet add package Microsoft.Extensions.Configuration
dotnet add package Microsoft.Extensions.Configuration.Json
dotnet add package Microsoft.Extensions.Configuration.AzureKeyVault
dotnet add package Microsoft.Azure.KeyVault
````
````

```
dotnet add package Microsoft.Azure.Services.AppAuthentication
dotnet add package System.Net.Http.Json
dotnet add package BlazorInputFile
dotnet add package Microsoft.AspNetCore.SignalR.Client
dotnet add package Microsoft.Identity.Web
```

Environment Configuration

Create a `.env` file in the root directory with the following content:

```
OpenAI_ApiKey=YOUR_OPENAI_API_KEY
Azure_KeyVaultName=YourKeyVaultName
Azure_ClientId=YourAzureClientId
Azure_ClientSecret=YourAzureClientSecret
Azure_TenantId=YourAzureTenantId
```

Azure Key Vault Setup

```
# Create Azure Key Vault
az keyvault create --name YourKeyVaultName --resource-group Y
ourResourceGroup --location YourLocation

# Store OpenAI API Key
az keyvault secret set --vault-name YourKeyVaultName --name O
penAI-ApiKey --value "YOUR_OPENAI_API_KEY"

# Assign Managed Identity to App Service
az webapp identity assign --resource-group YourResourceGroup
--name NexHomeAgentApp

# Set Key Vault Policy
az keyvault set-policy --name YourKeyVaultName --resource-gro
up YourResourceGroup --object-id <identityPrincipalId> --secr
et-permissions get
```

```
# Configure App Settings in Azure App Service
az webapp config appsettings set --resource-group YourResourc
eGroup --name NexHomeAgentApp --settings "KeyVaultName=YourKe
yVaultName"
```

Running the Application

```
dotnet run
```

Contact

For any inquiries, please contact [Your Contact Information].

```
By following these instructions and installing the necessary
dependencies, you can successfully run the NexHomeAgent appli
cation on your local machine and ensure a seamless developmen
t experience.
```