
# MVP For Dwellingly.ai

### Database and User Management System for Dwellingly.ai

Let's expand on the previous structure to include a comprehensive user management system. This includes setting up the database schema, creating user roles, and implementing the necessary services for user authentication and management.

### Database Setup

#### SQL Script for Database Creation and User Management Tables

```sqlCREATE DATABASE DwellinglyAI;
USE DwellinglyAI;
-- User TableCREATE TABLE Users (    Id INT PRIMARY KEY IDENTITY(1,1),    FullName NVARCHAR(100) NOT NULL,    Email NVARCHAR(100) UNIQUE NOT NULL,    PasswordHash NVARCHAR(255) NOT NULL,    Role NVARCHAR(50) NOT NULL,    CreatedAt DATETIME DEFAULT GETDATE(),    UpdatedAt DATETIME DEFAULT GETDATE());
-- Property TableCREATE TABLE Properties (    Id INT PRIMARY KEY IDENTITY(1,1),
```

Address NVARCHAR(255), City NVARCHAR(100), State NVARCHAR(50), ZipCode NVARCHAR(20), Price DECIMAL(18, 2), Description NVARCHAR(MAX), CreatedAt DATETIME DEFAULT GETDATE(), UpdatedAt DATETIME DEFAULT GETDATE());
-- Vector Data TableCREATE TABLE VectorData ( Id INT PRIMARY KEY IDENTITY(1,1), Content NVARCHAR(MAX), Vector VARBINARY(MAX), CreatedAt DATETIME DEFAULT GETDATE(), UpdatedAt DATETIME DEFAULT GETDATE());```
### User Management System
#### 1. **Models**
**User.cs**
```csharpnamespace DwellinglyAI.Models{ public class User { public int Id { get; set; } public string FullName { get; set; } public string Email { get; set; } public string PasswordHash { get; set; } public string Role { get; set; } public DateTime CreatedAt { get; set; } public DateTime UpdatedAt { get; set; } }}```
#### 2. **Data Context**
**ApplicationDbContext.cs**
```csharpusing Microsoft.EntityFrameworkCore;using DwellinglyAI.Models;
namespace DwellinglyAI.Data{ public class ApplicationDbContext : DbContext { public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options) { }
public DbSet<User> Users { get; set; } public DbSet<Property> Properties { get; set; } public DbSet<VectorData> VectorData { get; set; }
protected override void OnModelCreating(ModelBuilder modelBuilder) { base.OnModelCreating(modelBuilder); // Additional configuration here } }}```
#### 3. **Repositories**
**IUserRepository.cs**
```csharpusing System.Collections.Generic;using System.Threading.Tasks;using DwellinglyAI.Models;
namespace DwellinglyAI.Repositories{ public interface IUserRepository { Task<List<User>> GetUsersAsync(); Task<User> GetUserByIdAsync(int id); Task<User> GetUserByEmailAsync(string email); Task AddUserAsync(User user); Task UpdateUserAsync(User user); Task DeleteUserAsync(int id); }}```
**UserRepository.cs**
```csharpusing System.Collections.Generic;using System.Linq;using System.Threading.Tasks;using Microsoft.EntityFrameworkCore;using DwellinglyAI.Data;using DwellinglyAI.Models;
namespace DwellinglyAI.Repositories{ public class UserRepository : IUserRepository { private readonly ApplicationDbContext _context;

public UserRepository(ApplicationDbContext context) { _context = context; }
public async Task<List<User>> GetUsersAsync() { return await _context.Users.ToListAsync(); }
public async Task<User> GetUserByIdAsync(int id) { return await _context.Users.FindAsync(id); }
public async Task<User> GetUserByEmailAsync(string email) { return await _context.Users.SingleOrDefaultAsync(u => u.Email == email); }
public async Task AddUserAsync(User user) { _context.Users.Add(user); await _context.SaveChangesAsync(); }
public async Task UpdateUserAsync(User user) { _context.Users.Update(user); await _context.SaveChangesAsync(); }
public async Task DeleteUserAsync(int id) { var user = await _context.Users.FindAsync(id); if (user != null) { _context.Users.Remove(user); await _context.SaveChangesAsync(); } } }}```

#### 4. **Services**

**IUserService.cs**

```csharpusing System.Collections.Generic;using System.Threading.Tasks;using DwellinglyAI.Models;
namespace DwellinglyAI.Services{ public interface IUserService { Task<List<User>> GetUsersAsync(); Task<User> GetUserByIdAsync(int id); Task<User> GetUserByEmailAsync(string email); Task AddUserAsync(User user); Task UpdateUserAsync(User user); Task DeleteUserAsync(int id); }}```

**UserService.cs**

```csharpusing System.Collections.Generic;using System.Threading.Tasks;using DwellinglyAI.Models;using DwellinglyAI.Repositories;
namespace DwellinglyAI.Services{ public class UserService : IUserService { private readonly IUserRepository _userRepository;
public UserService(IUserRepository userRepository) { _userRepository = userRepository; }
public async Task<List<User>> GetUsersAsync() { return await _userRepository.GetUsersAsync(); }
public async Task<User> GetUserByIdAsync(int id) { return await _userRepository.GetUserByIdAsync(id); }
public async Task<User> GetUserByEmailAsync(string email) { return await

_userRepository.GetUserByEmailAsync(email); } public async Task AddUserAsync(User user) { await _userRepository.AddUserAsync(user); } public async Task UpdateUserAsync(User user) { await _userRepository.UpdateUserAsync(user); } public async Task DeleteUserAsync(int id) { await _userRepository.DeleteUserAsync(id); } }}```
#### 5. **Controllers**
**UsersController.cs**
```csharpusing System.Collections.Generic;using System.Threading.Tasks;using Microsoft.AspNetCore.Mvc;using DwellinglyAI.Models;using DwellinglyAI.Services; namespace DwellinglyAI.Controllers{ [ApiController] [Route("api/[controller]")] public class UsersController : ControllerBase { private readonly IUserService _userService; public UsersController(IUserService userService) { _userService = userService; } [HttpGet] public async Task<ActionResult<IEnumerable<User>>> GetUsers() { return await _userService.GetUsersAsync(); } [HttpGet("{id}")] public async Task<ActionResult<User>> GetUser(int id) { var user = await _userService.GetUserByIdAsync(id); if (user == null) { return NotFound(); } return user; } [HttpPost] public async Task<ActionResult<User>> AddUser(User user) { await _userService.AddUserAsync(user); return CreatedAtAction(nameof(GetUser), new { id = user.Id }, user); } [HttpPut("{id}")] public async Task<IActionResult> UpdateUser(int id, User user) { if (id != user.Id) { return BadRequest(); } await _userService.UpdateUserAsync(user); return NoContent(); } [HttpDelete("{id}")] public async Task<IActionResult> DeleteUser(int id) { await _userService.DeleteUserAsync(id); return NoContent(); } }}```
### Final Steps
1. **Configure Authentication**: - Implement authentication using ASP.NET Core Identity or JWT tokens. - Secure endpoints by applying `[Authorize]` attribute where necessary.
2. **Deploy to Azure**: - Set up the Azure environment with SQL Server, App Services, and other required resources. - Deploy the application and ensure all configurations (connection strings, environment variables) are correctly set up.
3. **Testing and Validation**: - Perform thorough testing of all user management functionalities. - Ensure data integrity and security throughout the application.
```

4. **Documentation**: - Document the API endpoints, data models, and usage instructions. - Provide a user manual or guide for both developers and end-users.
By following these steps and implementing the provided code, Dwellingly.ai will have a robust backend setup with user management capabilities adhering to Microsoft best practices and standards.

Let's proceed with completing and integrating the backend, database, user management system, and finalizing the deployment setup for the Dwellingly.ai application.
### Backend Integration
#### 1. **Database and User Management System**
We've already defined the SQL schema and the initial code structure. Let's now proceed to finalize and integrate the database and user management system.
**SQL Script for Database Creation and User Management Tables**
```sqlCREATE DATABASE DwellinglyAI;
USE DwellinglyAI;
-- User TableCREATE TABLE Users ( Id INT PRIMARY KEY IDENTITY(1,1), FullName NVARCHAR(100) NOT NULL, Email NVARCHAR(100) UNIQUE NOT NULL, PasswordHash NVARCHAR(255) NOT NULL, Role NVARCHAR(50) NOT NULL, CreatedAt DATETIME DEFAULT GETDATE(), UpdatedAt DATETIME DEFAULT GETDATE());
-- Property TableCREATE TABLE Properties ( Id INT PRIMARY KEY IDENTITY(1,1), Address NVARCHAR(255), City NVARCHAR(100), State NVARCHAR(50), ZipCode NVARCHAR(20), Price DECIMAL(18, 2), Description NVARCHAR(MAX), CreatedAt DATETIME DEFAULT GETDATE(), UpdatedAt DATETIME DEFAULT GETDATE());
-- Vector Data TableCREATE TABLE VectorData ( Id INT PRIMARY KEY IDENTITY(1,1), Content NVARCHAR(MAX), Vector VARBINARY(MAX), CreatedAt DATETIME DEFAULT GETDATE(), UpdatedAt DATETIME DEFAULT GETDATE());```
### Middleware Setup
Let's integrate the middleware components to connect the front end with the backend and the database.
**Middleware Services**
**UserService.cs**
```csharpusing System.Collections.Generic;using System.Threading.Tasks;using DwellinglyAI.Models;using DwellinglyAI.Repositories;
namespace DwellinglyAI.Services{ public class UserService : IUserService { private readonly IUserRepository _userRepository;

public UserService(IUserRepository userRepository) { _userRepository = userRepository; }
public async Task<List<User>> GetUsersAsync() { return await _userRepository.GetUsersAsync(); }
public async Task<User> GetUserByIdAsync(int id) { return await _userRepository.GetUserByIdAsync(id); }
public async Task<User> GetUserByEmailAsync(string email) { return await _userRepository.GetUserByEmailAsync(email); }
public async Task AddUserAsync(User user) { await _userRepository.AddUserAsync(user); }
public async Task UpdateUserAsync(User user) { await _userRepository.UpdateUserAsync(user); }
public async Task DeleteUserAsync(int id) { await _userRepository.DeleteUserAsync(id); } }}```

**PropertyService.cs**

```csharpusing System.Collections.Generic;using System.Net.Http;using System.Net.Http.Json;using System.Threading.Tasks;using DwellinglyAI.Models;
namespace DwellinglyAI.Services{ public class PropertyService : IPropertyService { private readonly HttpClient _httpClient;
public PropertyService(HttpClient httpClient) { _httpClient = httpClient; }
public async Task<List<Property>> GetPropertiesAsync() { return await _httpClient.GetFromJsonAsync<List<Property>>("api/properties"); }
public async Task<Property> GetPropertyByIdAsync(int id) { return await _httpClient.GetFromJsonAsync<Property>($"api/properties/{id}"); }
public async Task AddPropertyAsync(Property property) { await _httpClient.PostAsJsonAsync("api/properties", property); }
public async Task UpdatePropertyAsync(Property property) { await _httpClient.PutAsJsonAsync($"api/properties/{property.Id}", property); }
public async Task DeletePropertyAsync(int id) { await _httpClient.DeleteAsync($"api/properties/{id}"); }}```

### Backend API Endpoints

**UsersController.cs**

```csharpusing System.Collections.Generic;using System.Threading.Tasks;using Microsoft.AspNetCore.Mvc;using DwellinglyAI.Models;using DwellinglyAI.Services;
namespace DwellinglyAI.Controllers{ [ApiController] [Route("api/[controller]")] public class UsersController : ControllerBase { private readonly IUserService _userService;

public UsersController(IUserService userService) { _userService = userService; }
[HttpGet] public async Task<ActionResult<IEnumerable<User>>> GetUsers() { return await _userService.GetUsersAsync(); }
[HttpGet("{id}")] public async Task<ActionResult<User>> GetUser(int id) { var user = await _userService.GetUserByIdAsync(id); if (user == null) { return NotFound(); } return user; }
[HttpPost] public async Task<ActionResult<User>> AddUser(User user) { await _userService.AddUserAsync(user); return CreatedAtAction(nameof(GetUser), new { id = user.Id }, user); }
[HttpPut("{id}")] public async Task<IActionResult> UpdateUser(int id, User user) { if (id != user.Id) { return BadRequest(); } await _userService.UpdateUserAsync(user); return NoContent(); }
[HttpDelete("{id}")] public async Task<IActionResult> DeleteUser(int id) { await _userService.DeleteUserAsync(id); return NoContent(); } }}```
**PropertiesController.cs**
```csharpusing System.Collections.Generic;using System.Threading.Tasks;using Microsoft.AspNetCore.Mvc;using DwellinglyAI.Models;using DwellinglyAI.Services;
namespace DwellinglyAI.Controllers{ [ApiController] [Route("api/[controller]")] public class PropertiesController : ControllerBase { private readonly IPropertyService _propertyService;
public PropertiesController(IPropertyService propertyService) { _propertyService = propertyService; }
[HttpGet] public async Task<ActionResult<IEnumerable<Property>>> GetProperties() { return await _propertyService.GetPropertiesAsync(); }
[HttpGet("{id}")] public async Task<ActionResult<Property>> GetProperty(int id) { var property = await _propertyService.GetPropertyByIdAsync(id); if (property == null) { return NotFound(); } return property; }
[HttpPost] public async Task<ActionResult<Property>> AddProperty(Property property) { await _propertyService.AddPropertyAsync(property); return CreatedAtAction(nameof(GetProperty), new { id = property.Id }, property); }
[HttpPut("{id}")] public async Task<IActionResult> UpdateProperty(int id, Property property) { if (id != property.Id) { return BadRequest(); } await _propertyService.UpdatePropertyAsync(property); return NoContent(); }
[HttpDelete("{id}")] public async Task<IActionResult> DeleteProperty(int id) {
```

await _propertyService.DeletePropertyAsync(id);  return NoContent();  }  }}```
### Deployment
1. **Configure Authentication**:  - Implement authentication using ASP.NET Core Identity or JWT tokens.  - Secure endpoints by applying `[Authorize]` attribute where necessary.
2. **Deploy to Azure**:  - Set up the Azure environment with SQL Server, App Services, and other required resources.  - Deploy the application and ensure all configurations (connection strings, environment variables) are correctly set up.
3. **Testing and Validation**:  - Perform thorough testing of all user management functionalities.  - Ensure data integrity and security throughout the application.
4. **Documentation**:  - Document the API endpoints, data models, and usage instructions.  - Provide a user manual or guide for both developers and end-users.
5. **Continuous Integration and Deployment (CI/CD)**:  - Set up a CI/CD pipeline using Azure DevOps or GitHub Actions to automate deployment processes.
By following these steps and implementing the provided code, Dwellingly.ai will have a robust backend setup with user management capabilities adhering to Microsoft best practices and standards. This will ensure a scalable, secure, and efficient application.

site map:

```
Home
|-- Login
|-- Register
|-- Dashboard
|   |-- Property Search
|   |-- Property Details
|   |   |-- Make an Offer
|   |   |-- Mortgage Calculator
|   |   |-- Request More Information
|   |-- Transaction Management
|   |-- User Profile
|   |-- Notifications and Alerts
|-- About
|-- Contact
|-- Terms and Conditions
|-- Privacy Policy
```

Frontend blazer design

```
NexHomeAgentFrontend/
|-- Pages/
|   |-- LandingPage.razor
|   |-- UserLogin.razor
|   |-- AccountCreation.razor
|   |-- MainLandingPage.razor
|   |-- HouseSearchPage.razor
|   |-- CMAPage.razor
|   |-- OfferPage.razor
|   |-- SubmitOfferPage.razor
|   |-- PullCMAPage.razor
|   |-- PropertyList.razor
|   |-- PropertyDetails.razor
|-- Services/
|   |-- IUserService.cs
|   |-- UserService.cs
|   |-- IAccountService.cs
|   |-- AccountService.cs
|   |-- IPropertyService.cs
|   |-- PropertyService.cs
|-- Shared/
|   |-- MainLayout.razor
|   |-- NavMenu.razor
|-- wwwroot/
|   |-- css/
|   |   |-- app.css
|-- App.razor
|-- Program.cs
|-- Startup.cs
```

This structure represents the different components of your project, including individual pages, services, shared components, and root files.

# Table of Contents



# Site Map for MVP Application

1. **Landing Page**
    - Provides a concise introduction to the Dwellingly t service, highlighting the key benefits and value proposition.
    - Features a visually engaging hero section with high-quality imagery or video to capture the user's attention immediately.
    - Includes prominent call-to-action (CTA) buttons to guide users toward registration or login.
    - Showcases the application's key features and capabilities through interactive elements, such as a product tour or feature highlight reel.
    - Incorporates client testimonials and success stories to build trust and credibility.
2. **User Authentication**
    - User Registration Page
        - Allows users to create a new account by providing personal information and setting up their credentials.
        - Offers alternative login methods, such as social media or Google account integration, for a smoother user experience.
        - Includes a password strength indicator and guidance to ensure a secure password.
    - User Login Page

    - Enables users to access their accounts securely using their email and password.
    - Provides a 'Forgot Password?' option to facilitate password recovery.
    - Maintains a simple and focused design to keep the user's attention on the authentication process.

3. **User Dashboard**
    - Provides a personalized overview of the user's home-buying progress and activities.
    - Displays interactive data visualizations like charts and graphs to help users track their key metrics.
    - Shows a list of the user's saved properties, with the ability to quickly access detailed information.
    - Includes status updates and timelines for ongoing transactions, allowing users to stay informed.
    - Offers quick links and shortcuts to the application's key features and functionality, enabling seamless navigation.
    - Allows users to customize the layout and information displayed on the dashboard based on their preferences.
4. **Property Search**
    - Provides an intuitive and user-friendly search experience, with clearly labeled filters for location, price range, property type, and other advanced criteria.
    - It incorporates a geolocation-based search feature, enabling users to find properties in their desired area quickly.
    - Displays property listings with prominent images, key details, and the ability to save properties of interest.
    - Leverages the AI assistant to offer personalized property recommendations based on the user's search history and preferences.
    - Allows users to compare multiple properties and access virtual tours or 3D visualizations.
    - It supports pagination or infinite scrolling to handle many property listings.
5. **Property Details**

- Presents a comprehensive view of the selected property, including a carousel or gallery of high-quality images and interactive 3D visualizations or virtual tours.
- Provides detailed information about the property, such as the address, size, features, and location.
- Prominently displays the property's price and other relevant details.
- Includes clear 'Contact Us' and 'Make an Offer' buttons to guide users through the next steps of the home-buying process.
- Seamlessly integrates the AI assistant, allowing users to ask questions and receive detailed, contextual responses about the property.

6. **Offer Submission**
    - Offers a straightforward and user-friendly form for users to submit their offer on a property.
    - Leverages the AI assistant to provide guidance and suggestions on the offer amount, contingencies, and other relevant details.
    - Indicates the required fields and any necessary supporting documentation.
    - Implements a visual progress indicator to keep users informed about the status of their offer submission.
    - Provides timely notifications and updates throughout the offer process, ensuring users are aware of any changes or next steps.
7. **Transaction Management**
    - Utilizes a progress bar or timeline to visually represent the different stages of the user's home-buying transaction.
    - Integrates the AI assistant to guide users through each process step, providing real-time updates and personalized advice.
    - Allows users to securely upload, view, and share relevant documents related to their transactions.
    - Enables direct collaboration between the user and their real estate agent or other stakeholders through communication channels, such as secure messaging or task management tools.

8. **Support**
    - It offers a comprehensive, searchable knowledge base with FAQs, troubleshooting guides, and other helpful resources.
    - It includes a user-friendly 'Submit a Query' form that allows users to request personalized assistance. The AI assistant handles the initial troubleshooting and triage.
    - Provides a live chat feature powered by the AI assistant to offer immediate responses to user inquiries.
    - Continuously gathers user feedback and utilizes data-driven insights to improve the support experience over time.
9. **User Profile**
    - Enables users to view and edit their personal and contact information, such as name, email, and phone number.
    - Allows users to customize their notification preferences, including the frequency, type, and delivery method of alerts.
    - It allows users to manage their saved property searches and ensures they're notified of new listings that match their criteria.
    - Empower users to personalize their experience and stay informed throughout their home-buying journey.
10. **Notifications and Alerts**
    - Implements a system to send push notifications or in-app alerts to users for important events, such as offer acceptance, transaction updates, or the availability of new properties that match their saved search criteria.
    - Ensures these notifications are highly relevant and timely and provides clear calls-to-action to help users stay engaged and informed.
    - It allows users to customize their notification preferences, including snoozing, rescheduling, or managing alerts from a centralized location.
    - Leverages the AI assistant to provide additional context or recommendations within the notifications, enhancing their usefulness and relevance.

This updated site map for the MVP web application of the NexHomeAgent AI service focuses on delivering a user-centric, intuitive, and AI-powered experience that supports users throughout

their home-buying journey. The design and functionality of each section adhere to best practices for protecting/SaaS/AI applications, ensuring a visually appealing, responsive, and accessible user experience.

# User interface

As a SaaS wireframe expert, UI wireframe expert, web application expert, and Microsoft Blazor specialist, I have designed the UI for the NexHomeAgent AI application in the following manner:

**1. Landing Page (Home Page)**

- The landing page features a clean, modern, and visually engaging layout, with a hero section showcasing high-quality imagery or video to introduce the NexHomeAgent service.
- The value proposition is presented clearly and concisely, supported by eye-catching graphics or animations to engage visitors immediately.
- An interactive product tour or feature highlight reel allows visitors to explore the application's key capabilities.
- Prominent call-to-action (CTA) buttons guide users toward registration or login.
- Client testimonials and success stories are incorporated to build trust and credibility.
- The navigation bar is intuitive and consistent and provides easy access to the main sections of the application.

**2. User Authentication Page (Login/Sign Up)**

- The design maintains a simple and focused approach, with minimal distractions, to keep the user's attention on the authentication process.
- Clearly labeled email and password input fields are provided with placeholders to guide the user.
- The 'Register' button for new users is prominently displayed, making it easily accessible.
- A 'Forgot Password?' link enables a seamless password recovery experience.

- Alternative login methods, such as social media or Google account integration, are offered to improve the overall user experience.

**3. User Dashboard**

- The dashboard is personalized, displaying the user's name and a summary of their recent activity.
- The dashboard is organized into distinct sections or widgets, providing a clear overview of the user's home-buying progress, including:
    - Interactive data visualizations (e.g., charts, graphs) to showcase key metrics
    - A list of the user's saved properties with the ability to quickly access details
    - Status updates and timelines for any ongoing transactions
- Quick links or shortcuts to the application's key features and functionality are included, allowing users to navigate to the necessary sections quickly.

**4. Property Search and Listing Page**

- The property search page is designed to be intuitive and user-friendly, with clearly labeled filters for location, price range, property type, and other advanced criteria (e.g., number of bedrooms/bathrooms, lot size, year built).
- A geolocation-based search feature lets users quickly find properties in their desired area.
- Each property listing includes a prominent property image, the listing price, the property's location, and a summary of key features.
- Each listing includes a 'Save Property' button or icon, allowing users to bookmark exciting properties.
- The AI assistant provides personalized property recommendations based on the user's search history and preferences.
- Pagination or infinite scrolling handles many property listings, ensuring a smooth and efficient browsing experience.

**5. Property Details Page**

- The property details page presents a comprehensive view of the selected property, including a carousel or gallery of high-quality property images and interactive 3D property visualizations or virtual tours.
- Detailed information about the property, such as the address, size, features, and a location map, is provided.
- The property's price and other relevant details are prominently displayed.
- Clear 'Contact Us' and 'Make an Offer' buttons guide users through the next steps of the home-buying process.
- The integrated AI assistant allows users to ask questions and receive detailed, contextual responses about the selected property.

**6. Offer Submission Page**

- The offer submission page features a straightforward and user-friendly form for users to submit their offer on a property.
- The AI assistant provides guidance and suggestions on the offer amount, contingencies, and other relevant details, helping users make informed decisions.
- Required fields and any necessary supporting documentation are indicated.
- A visual progress indicator keeps users informed about the status of their offer submission.
- Timely notifications and updates are provided throughout the offer process, ensuring users are aware of any changes or next steps.

**7. Transaction Page**

- A progress bar or timeline visually represents the different stages of the user's home-buying transaction.
- The AI assistant guides users through each process step, providing real-time updates and personalized advice.
- A document management system allows users to securely upload, view, and share relevant documents related to their transactions.

- Communication channels, such as a secure messaging system, enable direct collaboration between the user and their real estate agent or other stakeholders.

**8. Support Page**

- The dedicated support page offers a comprehensive, searchable knowledge base with FAQs, troubleshooting guides, and other helpful resources.
- A 'Submit a Query' form allows users to request personalized assistance, with the AI assistant handling the initial troubleshooting and triage.
- A live chat feature powered by the AI assistant provides immediate responses to user inquiries.

**9. User Profile Page**

- Users can view and edit their personal and contact information, such as name, email, and phone number.
- Customization of notification preferences enables users to tailor the frequency and type of alerts they receive.
- Users can manage their saved property searches, ensuring they're notified of new listings that match their criteria.
- Clear instructions and guidance are provided so users can update their profile information securely.

**10. Notifications and Alerts**

- A system is implemented to send push notifications or in-app alerts to users for important events, such as offer acceptance, transaction updates, or the availability of new properties that match their saved search criteria.
- These notifications are highly relevant and timely, providing clear calls to action to help users stay engaged and informed throughout their home-buying journey.

## Best Practices and Standards

Throughout the development of the NexHomeAgent AI application's UI, I have adhered to Microsoft's best practices and standards, including:

1. **Fluent Design System**: The UI design aligns with Microsoft's Fluent Design System, creating a visually appealing, responsive, and inclusive user experience.
2. **Accessibility**: The UI design adheres to WCAG 2.1 standards, ensuring the application is accessible to users with disabilities.
3. **Responsive Design**: A mobile-first approach and responsive design techniques provide an optimal experience across different platforms.
4. **Performance Optimization**: The UI's performance is optimized by minimizing heavy assets, implementing lazy loading, and leveraging techniques like code splitting and caching.
5. **Reusable Components**: UI components are designed to be easily reused across the application, promoting consistency and maintainability.
6. **Theming and Styling**: A comprehensive design system establishes a consistent visual identity, utilizing Blazor's built-in support for CSS isolation and global styling.
7. **Telemetry and Analytics**: Telemetry and analytics tools are integrated to monitor the UI's usage, identify pain points, and gather insights for continuous improvement.
8. **Documentation and Developer Experience**: Comprehensive documentation for the UI design is provided, and a smooth developer experience is fostered through efficient collaboration and best practices.

## UX design for the NexHomeAgent AI application's buyer-focused features:

1. **Landing Page**
    - **Objective**: Immediately engage and inform potential buyers about Dwellingly’s service, highlighting its key benefits and value proposition.
    - **Design Approach**:

- Striking hero section with high-quality imagery or video showcasing the home-buying experience.
- Clear and concise value proposition statement that resonates with buyers' needs and desires.
- Prominent call-to-action buttons that guide users toward registration or login.
- An interactive product tour or feature highlight reel will showcase the application's capabilities.
- Visually appealing testimonials and success stories from satisfied buyers.
- The intuitive and responsive navigation menu allows easy access to other app sections.

2. **User Authentication**

- **Objective**: Provide buyers with a secure and user-friendly authentication experience to access the NexHomeAgent platform.
- **Design Approach**:
  - Minimalist and distraction-free layout to maintain focus on the authentication process.
  - I clearly labeled input fields for email and password, with placeholders to guide the user.
  - Prominent 'Register' button for new users, making the sign-up flow easily accessible.
  - Conspicuous 'Forgot Password?' link to facilitate a seamless password recovery experience.
  - Integrating social media or Google account login options for a more convenient sign-in method.
  - Subtle visual cues and animations to enhance the intuitiveness of the authentication flow.

3. **User Dashboard**

- **Objective**: Offer buyers a personalized, comprehensive, and interactive overview of their home-buying progress and activities.

- **Design Approach**:
    - Prominent display of the user's name and a summary of their recent activity.
    - Organized layout with distinct sections or widgets to provide a clear view of critical information:
        - Interactive data visualizations (e.g., charts, graphs, timelines) to track progress and metrics.
        - List of saved properties with the ability to quickly access details.
        - Status updates and timelines for ongoing transactions.
    - Intuitive navigation elements and shortcuts to enable easy access to other app features.
    - Customizable layout and information display to allow buyers to personalize their dashboard.

4. **Property Search**

- **Objective**: Provide buyers with an intuitive, AI-powered, and visually engaging property search experience to help them find their ideal home.
- **Design Approach**:
    - User-friendly search interface with clearly labeled filters for location, price range, property type, and advanced criteria.
    - Geolocation-based search functionality enables buyers to find properties in their desired area easily.
    - Visually striking property listings with prominent images, key details, and the ability to save properties of interest.
    - Integrating the AI assistant to provide personalized property recommendations based on the user's search history and preferences.
    - Side-by-side property comparison feature and access to virtual tours or 3D visualizations.
    - Infinite scrolling or pagination to handle a large number of property listings seamlessly.

5. **Property Details**

- **Objective**: To support their decision-making process and provide buyers with a comprehensive and immersive view of a selected property.
- **Design Approach**:
    - Carousel or gallery of high-quality property images to showcase the home in detail.
    - Integration of interactive 3D property visualizations or virtual tours to allow buyers to explore the space.
    - Detailed information about the property, including address, size, features, and location, is presented clearly and organized.
    - Prominent display of the property's price and other relevant details.
    - Strategically placed 'Contact Us' and 'Make an Offer' buttons to guide buyers through the following steps.
    - Seamless integration of the AI assistant enables buyers to ask questions and receive contextual responses about the property.

6. **Offer Submission**
    - **Objective**: Streamline the offer submission process, providing buyers with guidance and transparency.
    - **Design Approach**:
        - The form layout is straightforward and user-friendly for submitting an offer on a property.
        - Incorporation of the AI assistant to offer suggestions on the offer amount, contingencies, and other relevant details.
        - A clear indication of required fields and any necessary supporting documentation.
        - Visual progress indicator to keep buyers informed about the status of their offer submission.
        - Timely in-app notifications and updates throughout the offer process, ensuring buyers are aware of changes or next steps.
7. **Transaction Management**
    - **Objective**: Provide buyers with a centralized and collaborative platform to manage their home-buying transactions from start to finish.

- **Design Approach**:
    - A progress bar or timeline visualization will represent the different stages of the transaction.
    - The seamless integration of the AI assistant to guide buyers through each process step, offering real-time updates and personalized advice.
    - Secure document management system for uploading, viewing, and sharing relevant transaction files.
    - Collaborative communication channels, such as a secure messaging system or task management tools, enable direct collaboration with the real estate agent and other stakeholders.
    - Intuitive and user-friendly interface to ensure buyers can easily access and manage all aspects of their transactions.

8. **Support**
    - **Objective**: Offer buyers a comprehensive and responsive support experience to address their questions and concerns throughout their home-buying journey.
    - **Design Approach**:
        - Curated knowledge base with FAQs, troubleshooting guides, and other helpful resources for easy searching and browsing.
        - User-friendly 'Submit a Query' form that allows buyers to request personalized assistance, with the AI assistant handling the initial triage and response.
        - Integrated live chat feature, powered by the AI assistant, to provide immediate responses to buyer inquiries.
        - Continuous improvement of the support experience based on user feedback and data-driven insights.
9. **User Profile**
    - **Objective**: Empower buyers to manage their personal information and preferences and save searches within the NexHomeAgent application.
    - **Design Approach**:

- Intuitive interface for viewing and editing personal and contact details, such as name, email, and phone number.
- Customizable notification preferences, including frequency, type, and delivery method of alerts.
- Centralized management of saved property searches, enabling buyers to stay informed about new listings that match their criteria.
- Clear instructions and guidance to ensure a seamless and secure user profile management experience.

10. **Notifications and Alerts**

- **Objective**: Deliver timely and relevant notifications to keep buyers informed and engaged throughout their home-buying journey.
- **Design Approach**:
  - We provide visually appealing and informative push notifications or in-app alerts for important events, such as offer acceptance, transaction updates, or new property listings.
  - Highly relevant and personalized alerts based on the buyer's preferences, search history, and current stage in the home-buying process.
  - Clear calls-to-action within the notifications to encourage further engagement and action.
  - Intuitive alert management system, allowing buyers to customize, snooze, or reschedule notifications as needed.
  - Seamless integration of the AI assistant to provide additional context or recommendations within the notifications, enhancing their usefulness and relevance.

By focusing on user-centric design principles, leveraging the power of AI, and ensuring a consistent and responsive user experience, the NexHomeAgent AI application's buyer-focused features will provide a comprehensive and streamlined home-buying journey that meets the evolving needs and expectations of modern buyers.

# Wireframe

As a SaaS wireframe expert, UI wireframe expert, web application expert, and Microsoft Blazor specialist, I have designed the UI for the NexHomeAgent AI application in the following manner:

**1. Landing Page (Home Page)**

- The landing page features a clean, modern, and visually engaging layout, with a hero section showcasing high-quality imagery or video to introduce the NexHomeAgent service.
- The value proposition is presented clearly and concisely, supported by eye-catching graphics, animations, or interactive data visualizations that immediately engage visitors.
- An interactive product tour or feature highlight reel allows visitors to explore the application's key capabilities, utilizing techniques like parallax scrolling or micro-interactions to create a more immersive experience.
- Prominent call-to-action (CTA) buttons guide users toward registration or login, using contrasting colors and clear microcopy to drive conversions.
- Client testimonials and success stories are incorporated in a visually appealing manner, such as embedded video testimonials or carousel-style displays, to build trust and credibility.
- The navigation bar is intuitive and consistent, providing easy access to the application's main sections. It also has clear labeling and a responsive design to accommodate different device sizes.

**2. User Authentication Page (Login/Sign Up)**

- The design maintains a simple and focused approach, with minimal distractions, to keep the user's attention on the authentication process.
- Clearly labeled email and password input fields, with placeholders to guide the user, are provided, along with visual cues or animations to enhance the intuitiveness of the flow.
- The 'Register' button for new users is prominently displayed, making it easily accessible, and may include a password strength indicator or password policy guidance to improve the security and user experience.
- A 'Forgot Password?' link enables a seamless password recovery experience with a clear and user-friendly password reset flow.

- Alternative login methods, such as social media or Google account integration, are offered to improve the overall user experience, with consistent branding and visual styling to maintain a cohesive look and feel.

**3. User Dashboard**

- The dashboard is personalized, displaying the user's name and a summary of their recent activity, with the ability to customize the layout and information displayed.
- The dashboard is organized into distinct sections or widgets, providing a clear overview of the user's home-buying progress, including:
    - Interactive data visualizations (e.g., charts, graphs, timelines) to showcase critical metrics and insights, with the ability to drill down into more detailed information.
    - A list of the user's saved properties with the ability to quickly access details, including features like side-by-side property comparisons.
    - The dashboard provides status updates and timelines for ongoing transactions, and users can view document attachments or collaborate with stakeholders directly.
- Quick links or shortcuts to the application's key features and functionality are included, allowing users to navigate to the needed sections quickly. Users can customize and rearrange these elements based on their preferences.

**4. Property Search and Listing Page**

- The property search page is designed to be intuitive and user-friendly, with clearly labeled filters for location, price range, property type, and other advanced criteria (e.g., number of bedrooms/bathrooms, lot size, year built).
- A geolocation-based search feature enables users to easily find properties in their desired area. Users can save and recall custom search queries or set alerts for new listings matching their criteria.
- Each property listing includes a prominent image, the listing price, the property's location, and a summary of key features. You can view more detailed information or access a virtual tour or 3D visualization of the property.
- Each listing incorporates a 'Save Property' button or icon, allowing users to bookmark properties of interest and compare multiple properties side-by-side.

- The AI assistant provides personalized property recommendations based on the user's search history and preferences. The user can provide feedback or adjust the recommendation algorithm.
- Pagination or infinite scrolling handles many property listings, ensuring a smooth and efficient browsing experience, with the ability to adjust the layout and visual presentation of the listings based on user preferences.

**5. Property Details Page**

- The property details page presents a comprehensive view of the selected property, including a carousel or gallery of high-quality property images and interactive 3D property visualizations or virtual tours to allow users to explore the space in detail.
- Detailed information about the property, such as the address, size, features, and a location map, is provided, with the ability to toggle between different views or data points to suit the user's needs.
- The property's price and other relevant details are prominently displayed, with the option to view the area's historical pricing data or market trends.
- Clear 'Contact Us' and 'Make an Offer' buttons guide users through the next steps of the home-buying process, with the ability to initiate these actions directly from the property details page.
- The AI assistant is seamlessly integrated, allowing users to ask questions and receive detailed, contextual responses about the selected property, with the option to save or review past conversations.

**6. Offer Submission Page**

- The offer submission page features a straightforward and user-friendly form for users to submit their offers on properties, with clear guidance and suggestions from the AI assistant.
- The form is designed with a step-by-step workflow, providing visual progress indicators to help users understand their current status and the next steps in the process.
- Required fields and any necessary supporting documentation are indicated, with the ability to upload files or attachments directly from the page.
- Timely notifications and updates are provided throughout the offer process, ensuring users are aware of any changes or next steps, with the option to receive push notifications or in-app alerts based on their preferences.

- The AI assistant provides guidance and suggestions on the offer amount, contingencies, and other relevant details, helping users make informed decisions. It can also save and recall previous offer templates or strategies.

**7. Transaction Page**

- A progress bar or timeline visually represents the different stages of the user's home-buying transaction, with clear labeling and the ability to drill down into the details of each step.
- The AI assistant guides users through each step of the process, providing real-time updates and personalized advice. Users can also schedule reminders or set custom alerts for important milestones.
- A document management system allows users to securely upload, view, and share relevant documents related to their transactions. They can also collaborate with their real estate agent or other stakeholders through integrated communication channels, such as a secure messaging system or task management tools.
- The transaction page provides a centralized hub for users to track the progress of their home-buying journey. Users can access relevant information, communicate with stakeholders, and manage the necessary documentation seamlessly and efficiently.

**8. Support Page**

- The dedicated support page offers a comprehensive, searchable knowledge base with FAQs, troubleshooting guides, and other helpful resources, with the ability to filter or sort the content based on user needs or shared queries.
- A 'Submit a Query' form allows users to request personalized assistance. The AI assistant handles the initial troubleshooting and triage, and the user can escalate the request to a human support representative if necessary.
- A live chat feature powered by the AI assistant provides immediate responses to user inquiries and can seamlessly transition to a human agent if the issue requires more specialized assistance.
- The support page is designed to be highly intuitive and user-friendly. It features clear navigation, the ability to rate the helpfulness of resources, and the option to provide feedback to continuously improve the support experience.

**9. User Profile Page**

- Users can view and edit their personal and contact information, such as name, email, and phone number, with clear instructions and guidance to ensure secure and prosperous updates.
- Customization of notification preferences enables users to tailor the frequency, type, and delivery method of alerts they receive, as well as set custom thresholds or trigger conditions.
- Users can manage their saved property searches, ensuring they're notified of new listings that match their criteria and the option to share or collaborate on these saved searches with other users or stakeholders.
- The user profile page provides a centralized hub for users to control their account settings, preferences, and interactions with the NexHomeAgent application, empowering them to personalize their experience and stay informed throughout their home-buying journey.

**10. Notifications and Alerts**

- A system is implemented to send push notifications or in-app alerts to users for important events, such as offer acceptance, transaction updates, or the availability of new properties that match their saved search criteria.
- These notifications are highly relevant and timely and provide clear calls to action to help users stay engaged and informed throughout their home-buying journey, with the ability to customize the alerts' frequency, delivery method, and content.
- The notification system is designed to be intuitive and user-friendly. Users can snooze, reschedule, or manage alerts from a centralized location, ensuring they maintain control over their notification experience.
- The AI assistant may provide additional context or recommendations within the notifications, leveraging the user's preferences and past behavior to enhance the relevance and usefulness of the alerts.

**Best Practices for Wireframe Design (Proteck/SaaS/AI)**

Throughout the development of the NexHomeAgent AI application's UI wireframe, I have adhered to the following best practices for project/SaaS/AI applications:

1. **User-Centric Design**: The wireframe places the user at the center of the design process, focusing on understanding their needs, pain points, and goals through thorough user research and continuous validation.

2. **Simplicity and Clarity**: The layout and information architecture maintain a clean and uncluttered design, prioritizing simplicity and ease of use, with clear labeling, intuitive navigation, and minimal cognitive load for the user.
3. **Responsive and Adaptive Design**: The wireframe ensures the UI design is responsive and adapts seamlessly to different device types and screen sizes, optimizing the user experience for mobile, tablet, and desktop environments.
4. **Consistency and Branding**: The wireframe establishes a consistent visual style, typography, color scheme, and branding elements throughout the application, aligning with the NexHomeAgent brand identity and guidelines.
5. **Accessibility and Inclusivity**: The wireframe adheres to WCAG and other accessibility standards, catering to users with diverse abilities and needs. Its features and design choices improve usability for users with various impairments.
6. **Efficient Information Architecture**: The wireframe organizes the application's content and functionality logically and intuitively, employing clear navigation hierarchies, comprehensive labeling, and intuitive information grouping.
7. **Intelligent Interactions and Automation**: The wireframe leverages the AI capabilities of the NexHomeAgent application to enhance the user's experience, seamlessly integrating the AI assistant and automating repetitive or time-consuming tasks.
8. **Data-Driven Insights and Analytics**: The wireframe incorporates data visualization and analytics features to help users track their progress, analyze their activities, and make informed decisions, with the ability to surface relevant insights and actionable data.
9. **Scalability and Extensibility**: The wireframe is designed with future growth and expansion in mind, ensuring the UI can accommodate new features and functionalities. Its modular and extensible architecture allows for easy integration of additional components or services.
10. **Continuous Improvement and Iteration**: The wireframe adopts an iterative design approach, regularly gathering user feedback and incorporating it into the design. This fosters a culture of experimentation, testing, and data-driven decision-making to enhance the UI over time.

Following these best practices for wireframe design in a project/SaaS/AI application, the NexHomeAgent AI application's UI will deliver a visually appealing, user-centric, and highly functional experience, setting it up for long-term success and growth.

Certainly! Here's how I would apply the best practices and standards for the graphic design of the NexHomeAgent AI application:

1. **Adhere to the Fluent Design System**
    - Align the visual design with Microsoft's Fluent Design System guidelines to ensure consistency and adherence to industry-leading standards.
    - Incorporate Fluent Design principles, such as responsive layouts, efficient animations, and tactile interactions, to create a modern and cohesive user experience.
    - Leverage Fluent Design's comprehensive guidelines for typography, color, iconography, and other key visual elements.
2. **Ensure Accessibility and Inclusivity**
    - Prioritize accessibility throughout the design process, adhering to WCAG 2.1 guidelines and other relevant accessibility standards.
    - Utilize high-contrast color palettes, straightforward typography, and appropriate alt-text for images to support users with visual impairments.
    - To accommodate users with motor or cognitive disabilities, provide alternative interaction methods, such as keyboard navigation and screen reader support.
    - Conduct regular accessibility testing and incorporate feedback to improve the application's inclusive design continuously.
3. **Implement Responsive and Adaptive Design**
    - Adopt a mobile-first approach to graphic design, ensuring the user interface is optimized for various screen sizes and device types.
    - Utilize responsive design techniques, such as flexible grids, adaptive layouts, and fluid typography, to provide an optimal experience across different platforms.
    - Test the design across various devices and screen resolutions to identify and address any layout or compatibility issues.
4. **Maintain Consistent Branding and Visual Identity**
    - Establish a solid and cohesive brand identity for NexHomeAgent, aligning the visual design with the brand's mission, values, and target audience.

- Develop a comprehensive design system with a well-defined logo, color palette, typography, iconography, and other visual elements.
- Ensure consistent brand identity application across all touchpoints of the user experience, from the website to the mobile app.

5. **Optimize for Performance and Perceived Performance**
    - Minimize heavy assets, such as high-resolution images or complex animations, to optimize the application's performance.
    - Implement techniques like lazy loading, code splitting, and caching to improve the user interface's perceived performance and responsiveness.
    - Continuously monitor and measure the application's performance, adjusting the graphic design to maintain a smooth and efficient user experience.
6. **Leverage Data-Driven Insights**
    - Integrate telemetry and analytics tools to gather user behavior data and insights.
    - Use these insights to identify areas for improvement, validate design decisions, and continuously enhance the graphic design based on user feedback and performance metrics.
    - Collaborate with product managers and user researchers to ensure the graphic design aligns with the overall user experience goals and objectives.
7. **Foster a Culture of Collaboration and Iteration**
    - Encourage a collaborative design process involving stakeholders, developers, and subject matter experts to gather diverse perspectives and feedback.
    - Adopt an iterative design approach, regularly gathering user feedback and incorporating it into the graphic design.
    - Establish clear design documentation, guidelines, and best practices to ensure consistency and maintainability across the design team.
8. **Provide Comprehensive Design Documentation**
    - Create detailed design documentation, including style guides, component libraries, and design rationale, to ensure a smooth handoff to the development team.

- Establish clear guidelines for using brand assets, visual elements, and interaction patterns to promote consistency and adherence to the NexHomeAgent design system.
- Continuously update the design documentation as the application evolves, ensuring it remains a reliable and up-to-date resource for the entire project team.

9. **Prioritize Modularity and Extensibility**
    - Design the UI components and visual elements modular and reusable, allowing for easy integration and adaptation as the application grows.
    - Ensure the graphic design can accommodate future feature additions and UI enhancements without requiring extensive rework or redesign.
    - Maintain a scalable and extensible design system that can evolve alongside the NexHomeAgent AI application.
10. **Embrace Ongoing Improvement and Iteration**
    - Regularly review user feedback, industry trends, and technological advancements to identify opportunities for improvement in graphic design.
    - Implement a culture of experimentation, testing, and data-driven decision-making to enhance the visual experience of the NexHomeAgent AI application continuously.
    - Foster a collaborative environment where the design team can learn, share best practices, and adapt to the changing needs of the user base.

Based on the key observations from the provided images, here are the main points to consider for the styling guide and updated design of the NexHomeAgent AI application:

1. Consistent Branding and Visual Identity
    - Maintain a clean, modern, and visually striking design aesthetic that aligns with the NexHomeAgent brand.
    - Ensure a cohesive use of typography, color palette, iconography, and other visual elements across the entire application.
2. Intuitive and Responsive User Interface

- Organize the information and functionalities in a clear and logical manner to enhance the user experience.
- Ensure the design is responsive and adapts seamlessly to different device sizes and screen resolutions.

3. Integrated AI Assistant
    - Prominently feature the AI assistant's capabilities and presence throughout the application.
    - Incorporate visual cues and interactions that make the AI assistant's integration feel natural and seamless.
4. Comprehensive Real Estate Toolkit
    - Develop a centralized "Real Estate Toolkit" or similar section that consolidates the key features and tools for homebuyers.
    - Include functionalities such as mortgage calculators, loan comparison tools, market analysis, utility management, and property-related services.
5. Streamlined Property Exploration
    - Provide an intuitive and visually engaging property listing experience, allowing users to easily browse, view details, and interact with the available homes.
    - Integrate features like virtual tours, 3D visualizations, and the ability to submit offers directly within the application.
6. Personalized Dashboard and Insights
    - Design a personalized dashboard that offers a comprehensive overview of the user's home-buying progress and activities.
    - Incorporate data visualizations, timelines, and other interactive elements to help users track their journey and access relevant information.
7. Accessible and Inclusive Design
    - Ensure the application adheres to accessibility standards, catering to users with various abilities and needs.
    - Implement design choices that promote inclusivity and a positive user experience for all.

8. Iterative Improvement and User Feedback
    - Foster a culture of continuous improvement, regularly gathering user feedback and incorporating it into the design updates.
    - Establish a process for monitoring user behavior, analyzing performance metrics, and making data-driven design decisions.

```
<Router AppAssembly="@typeof(Program).Assembly">
    <Found Context="routeData">
        <RouteView RouteData="@routeData" DefaultLayout="@
typeof(MainLayout)" />
    </Found>
    <NotFound>
        <LayoutView Layout="@typeof(MainLayout)">
            <p>Sorry, there's nothing at this address.</p>
        </LayoutView>
    </NotFound>
</Router>
```

By incorporating these key elements into the styling guide and updated design of the NexHomeAgent AI application, the user interface will become more visually appealing, intuitive, and responsive, delivering a seamless and empowering experience for homebuyers throughout their journey.

MVP Property Build Structure.

```
NexHomeAgentFrontend/
|-- Pages/
|   |-- LandingPage.razor
|   |-- UserLogin.razor
|   |-- AccountCreation.razor
|   |-- MainLandingPage.razor
|   |-- HouseSearchPage.razor
|   |-- CMAPage.razor
```

```
|   |-- OfferPage.razor
|   |-- SubmitOfferPage.razor
|   |-- PullCMAPage.razor
|   |-- PropertyList.razor
|   |-- PropertyDetails.razor
|-- Services/
|   |-- IUserService.cs
|   |-- UserService.cs
|   |-- IAccountService.cs
|   |-- AccountService.cs
|   |-- IPropertyService.cs
|   |-- PropertyService.cs
|-- Shared/
|   |-- MainLayout.razor
|   |-- NavMenu.razor
|-- wwwroot/
|   |-- css/
|   |   |-- app.css
|-- App.razor
|-- Program.cs
|-- Startup.cs
```

Sure, here's the full Blazor code for the NexHomeAgent AI application's frontend:

Sure, here's the full Blazor code for the NexHomeAgent AI application's frontend:

1. **App.razor**

```
<Router AppAssembly="@typeof(Program).Assembly">
    <Found Context="routeData">
        <RouteView RouteData="@routeData" DefaultLayout="@typ
eof(MainLayout)" />
    </Found>
    <NotFound>
        <LayoutView Layout="@typeof(MainLayout)">
```

```
            <p>Sorry, there's nothing at this address.</p>
        </LayoutView>
    </NotFound>
</Router>
```

1. **MainLayout.razor**

```
@inherits LayoutComponentBase

<div class="layout">
    <div class="sidebar">
        <NavMenu />
    </div>

    <div class="main">
        <div class="top-bar">
            <a href="/" class="logo">NexHomeAgent</a>
            <div class="nav-links">
                <a href="/search">Search</a>
                <a href="/dashboard">Dashboard</a>
                <a href="/profile">Profile</a>
                <a href="/support">Support</a>
            </div>
            <div class="user-actions">
                <button class="btn btn-primary">Ask AI Agent
</button>
                <button class="btn btn-outline-primary">Logou
t</button>
            </div>
        </div>

        <div class="content">
            @Body
        </div>
    </div>
```

```
</div>

<style>
    /* Add your custom styles here */
</style>
```

1. **NavMenu.razor**

```
<div class="nav-menu">
    <ul>
        <li><a href="/">Home</a></li>
        <li><a href="/search">Search</a></li>
        <li><a href="/dashboard">Dashboard</a></li>
        <li><a href="/profile">Profile</a></li>
        <li><a href="/support">Support</a></li>
    </ul>
</div>

<style>
    /* Add your custom styles here */
</style>
```

1. **PropertyListPage.razor**

```
@page "/search"
@inject IPropertyService PropertyService

<PageTitle>Property Search</PageTitle>

<div class="property-list-page">
    <div class="search-bar">
        <input type="text" placeholder="Search for propertie
s..." @bind-value="searchQuery" />
        <button class="btn btn-primary" @onclick="SearchPrope
rties">Search</button>
```

```
    </div>

    <div class="property-list">
        @if (properties == null)
        {
            <p><em>Loading...</em></p>
        }
        else
        {
            @foreach (var property in properties)
            {
                <div class="property-card">
                    <img src="@property.ImageUrl" alt="@prope
rty.Address" />
                    <h3>@property.Address</h3>
                    <p>@property.City, @property.State @prope
rty.ZipCode</p>
                    <p>Price: @property.Price.ToString("C")</
p>
                    <div class="actions">
                        <a href="/property/@property.Id" clas
s="btn btn-primary">View Details</a>
                        <button class="btn btn-outline-primar
y" @onclick="() => SaveProperty(property)">Save Property</but
ton>
                    </div>
                </div>
            }
        }
    </div>
</div>

@code {
    private string searchQuery;
    private List<Property> properties;
```

```
    protected override async Task OnInitializedAsync()
    {
        properties = await PropertyService.GetPropertiesAsync
();
    }

    private async Task SearchProperties()
    {
        properties = await PropertyService.SearchPropertiesAs
ync(searchQuery);
    }

    private void SaveProperty(Property property)
    {
        // Add logic to save the property
    }
}

<style>
    /* Add your custom styles here */
</style>
```

1. **PropertyDetailsPage.razor**

```
@page "/property/{PropertyId:int}"
@inject IPropertyService PropertyService

<PageTitle>Property Details</PageTitle>

<div class="property-details-page">
    @if (property == null)
    {
        <p><em>Loading...</em></p>
    }
    else
```

```
    {
        <div class="property-header">
            <h1>@property.Address</h1>
            <p>@property.City, @property.State @property.ZipC
ode</p>
        </div>

        <div class="property-details">
            <div class="property-images">
                <img src="@property.ImageUrl" alt="@property.
Address" />
                <!-- Add support for multiple images or a car
ousel -->
            </div>

            <div class="property-info">
                <h2>Property Details</h2>
                <p>Price: @property.Price.ToString("C")</p>
                <p>Bedrooms: @property.Bedrooms</p>
                <p>Bathrooms: @property.Bathrooms</p>
                <p>Square Feet: @property.SquareFeet</p>
                <p>@property.Description</p>
            </div>

            <div class="property-actions">
                <button class="btn btn-primary">Contact Agent
</button>
                <button class="btn btn-outline-primary">Make
Offer</button>
            </div>
        </div>

        <div class="property-map">
            <!-- Add a map component to display the property
location -->
        </div>
```

```
    }
</div>

@code {
    [Parameter]
    public int PropertyId { get; set; }

    private Property property;

    protected override async Task OnInitializedAsync()
    {
        property = await PropertyService.GetPropertyByIdAsync
(PropertyId);
    }
}

<style>
    /* Add your custom styles here */
</style>
```

1. **DashboardPage.razor**

```
@page "/dashboard"
@inject IPropertyService PropertyService
@inject ITransactionService TransactionService

<PageTitle>Dashboard</PageTitle>

<div class="dashboard-page">
    <div class="user-summary">
        <h2>Welcome back, John Doe!</h2>
        <p>Here's a quick summary of your home-buying journe
y:</p>
    </div>
```

```
    <div class="dashboard-widgets">
        <div class="widget">
            <h3>Saved Properties</h3>
            <p>You have @savedProperties.Count saved properti
es.</p>
            <a href="/search" class="btn btn-primary">View Sa
ved Properties</a>
        </div>

        <div class="widget">
            <h3>Ongoing Transactions</h3>
            <p>You have @ongoingTransactions.Count active tra
nsactions.</p>
            <a href="/transactions" class="btn btn-primary">V
iew Transactions</a>
        </div>

        <div class="widget">
            <h3>Mortgage Calculator</h3>
            <p>Estimate your monthly mortgage payments.</p>
            <a href="/mortgage-calculator" class="btn btn-pri
mary">Use Calculator</a>
        </div>
    </div>

    <div class="recent-activity">
        <h2>Recent Activity</h2>
        <div class="activity-list">
            <!-- Display the user's recent activities here --
>
        </div>
    </div>
</div>

@code {
    private List<Property> savedProperties;
```

```
    private List<Transaction> ongoingTransactions;

    protected override async Task OnInitializedAsync()
    {
        savedProperties = await PropertyService.GetSavedPrope
rtiesAsync();
        ongoingTransactions = await TransactionService.GetOng
oingTransactionsAsync();
    }
}

<style>
    /* Add your custom styles here */
</style>
```

1. **UserProfilePage.razor**

```
@page "/profile"
@inject IUserService UserService

<PageTitle>User Profile</PageTitle>

<div class="user-profile-page">
    <div class="profile-header">
        <h1>Your Profile</h1>
    </div>

    <div class="profile-form">
        <div class="form-group">
            <label for="name">Name</label>
            <input type="text" id="name" @bind-value="@user.N
ame" />
        </div>
        <div class="form-group">
            <label for="email">Email</label>
```

```
            <input type="email" id="email" @bind-value="@use
r.Email" />
        </div>
        <div class="form-group">
            <label for="phone">Phone</label>
            <input type="tel" id="phone" @bind-value="@user.P
hone" />
        </div>
        <div class="form-group">
            <label for="notifications">Notification Preferenc
es</label>
            <select id="notifications" @bind="@user.Notificat
ionPreferences">
                <option value="0">All Notifications</option>
                <option value="1">Important Notifications Onl
y</option>
                <option value="2">No Notifications</option>
            </select>
        </div>
        <button class="btn btn-primary" @onclick="UpdateProfi
le">Save Changes</button>
    </div>
</div>

@code {
    private User user;

    protected override async Task OnInitializedAsync()
    {
        user = await UserService.GetUserProfileAsync();
    }

    private async Task UpdateProfile()
    {
        await UserService.UpdateUserProfileAsync(user);
    }
```

```
}

<style>
    /* Add your custom styles here */
</style>
```

1. **SupportPage.razor**

```
@page "/support"
@inject ISupportService SupportService

<PageTitle>Support</PageTitle>

<div class="support-page">
    <div class="support-header">
        <h1>How can we help you?</h1>
    </div>

    <div class="support-content">
        <div class="knowledge-base">
            <h2>Knowledge Base</h2>
            <input type="text" placeholder="Search the knowle
dge base..." @bind-value="searchQuery" @onkeyup="SearchKnowle
dgeBase" />
            <div class="articles">
                @if (articles == null)
                {
                    <p><em>Loading...</em></p>
                }
                else
                {
                    @foreach (var article in articles)
                    {
                        <div class="article-card">
                            <h3>@article.Title</h3>
```

```
                            <p>@article.Description</p>
                            <a href="/support/articles/@artic
le.Id" class="btn btn-primary">Read More</a>
                        </div>
                    }
                }
            </div>
        </div>

        <div class="submit-query">
            <h2>Submit a Query</h2>
            <EditForm Model="@supportRequest" OnValidSubmit
="SubmitSupportRequest">
                <div class="form-group">
                    <label for="subject">Subject</label>
                    <input type="text" id="subject" @bind-val
ue="@supportRequest.Subject" />
                </div>
                <div class="form-group">
                    <label for="message">Message</label>
                    <textarea id="message" @bind-value="@supp
ortRequest.Message"></textarea>
                </div>
                <button type="submit" class="btn btn-primar
y">Submit</button>
            </EditForm>
        </div>
    </div>
</div>

@code {
    private string searchQuery;
    private List<KnowledgeBaseArticle> articles;
    private SupportRequest supportRequest = new SupportReques
t();
```

```
    protected override async Task OnInitializedAsync()
    {
        articles = await SupportService.GetKnowledgeBaseArtic
lesAsync();
    }

    private async Task SearchKnowledgeBase()
    {
        articles = await SupportService.SearchKnowledgeBaseAs
ync(searchQuery);
    }

    private async Task SubmitSupportRequest()
    {
        await SupportService.SubmitSupportRequestAsync(suppor
tRequest);
        // Reset the support request form
        supportRequest = new SupportRequest();
    }
}

<style>
    /* Add your custom styles here */
</style>
```

1. **OfferSubmissionPage.razor**

```
@page "/offer/{PropertyId:int}"
@inject IPropertyService PropertyService
@inject IOfferService OfferService

<PageTitle>Submit Offer</PageTitle>

<div class="offer-submission-page">
    <div class="offer-header">
```

```
        <h1>Submit Offer for @property.Address</h1>
    </div>

    <div class="offer-form">
        <EditForm Model="@offerDetails" OnValidSubmit="Submit
Offer">
            <div class="form-group">
                <label for="offerAmount">Offer Amount</label>
                <input type="number" id="offerAmount" @bind-v
alue="@offerDetails.OfferAmount" />
            </div>
            <div class="form-group">
                <label for="downPayment">Down Payment</label>
                <input type="number" id="downPayment" @bind-v
alue="@offerDetails.DownPayment" />
            </div>
            <div class="form-group">
                <label for="financingType">Financing Type</la
bel>
                <select id="financingType" @bind="@offerDetai
ls.FinancingType">
                    <option value="Conventional">Conventional
</option>
                    <option value="FHA">FHA</option>
                    <option value="VA">VA</option>
                    <option value="Cash">Cash</option>
                </select>
            </div>
            <div class="form-group">
                <label for="closingDate">Desired Closing Date
</label>
                <input type="date" id="closingDate" @bind-val
ue="@offerDetails.DesiredClosingDate" />
            </div>
            <div class="form-group">
                <label for="additionalTerms">Additional Terms
```

```
</label>
                <textarea id="additionalTerms" @bind-value="@
offerDetails.AdditionalTerms"></textarea>
            </div>
            <button type="submit" class="btn btn-primary">Sub
mit Offer</button>
        </EditForm>
    </div>

    <div class="offer-progress">
        <h2>Offer Submission Progress</h2>
        <div class="progress-bar">
            <!-- Implement a progress bar to show the offer s
ubmission status -->
        </div>
    </div>
</div>

@code {
    [Parameter]
    public int PropertyId { get; set; }

    private Property property;
    private OfferDetails offerDetails = new OfferDetails();

    protected override async Task OnInitializedAsync()
    {
        property = await PropertyService.GetPropertyByIdAsync
(PropertyId);
    }

    private async Task SubmitOffer()
    {
        await OfferService.SubmitOfferAsync(PropertyId, offer
Details);
        // Reset the offer details form
```

```
        offerDetails = new OfferDetails();
    }
}

<style>
    /* Add your custom styles here */
</style>
```

1. **TransactionPage.razor**

```
@page "/transactions"
@inject ITransactionService TransactionService

<PageTitle>Transactions</PageTitle>

<div class="transaction-page">
    <div class="transaction-header">
        <h1>Your Transactions</h1>
    </div>

    <div class="transaction-list">
        @if (transactions == null)
        {
            <p><em>Loading...</em></p>
        }
        else
        {
            @foreach (var transaction in transactions)
            {
                <div class="transaction-card">
                    <div class="transaction-details">
                        <h3>@transaction.Property.Address</h3
>
                        <p>@transaction.Status</p>
                        <p>Offer Submitted: @transaction.Offe
```

```
rDate.ToString("MM/dd/yyyy")</p>
                        <p>Closing Date: @transaction.Closing
Date.ToString("MM/dd/yyyy
```

1. **MainLayout.razor**

```
csharp
Copy
// MainLayout.razor
@inherits LayoutComponentBase

<div class="sidebar">
    <NavMenu />
</div>

<div class="main">
    <div class="top-row px-4">
        <a href="https://docs.microsoft.com/aspnet/" target
="_blank">About</a>
    </div>

    <div class="content px-4">
        @Body
    </div>
</div>
```

1. **NavMenu.razor**

```
csharp
Copy
// NavMenu.razor
<nav class="nav-menu">
    <ul>
        <li><a href="/">Home</a></li>
        <li><a href="/properties">Properties</a></li>
```

```
    </ul>
</nav>

<style>
.nav-menu {
    width: 200px;
    background-color: #f8f9fa;
    padding: 10px;
}

.nav-menu ul {
    list-style-type: none;
    padding: 0;
}

.nav-menu li {
    margin: 5px 0;
}

.nav-menu a {
    text-decoration: none;
    color: #333;
}
</style>
```

1. **Property.cs**

```
csharp
Copy
// Property.cs
namespace NexHomeAgentFrontend.Models
{
    public class Property
    {
        public int Id { get; set; }
```

```
        public string Address { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string ZipCode { get; set; }
        public decimal Price { get; set; }
        public string Description { get; set; }
    }
}
```

1. **ChatMessage.cs**

```
csharp
Copy
// ChatMessage.cs
namespace NexHomeAgentFrontend.Models
{
    public class ChatMessage
    {
        public string Text { get; set; }
        public bool IsFromUser { get; set; }
    }
}
```

1. **IPropertyService.cs**

```
csharp
Copy
// IPropertyService.cs
using System.Collections.Generic;
using System.Threading.Tasks;
using NexHomeAgentFrontend.Models;

public interface IPropertyService
{
    Task<List<Property>> GetPropertiesAsync();
```

```
    Task<Property> GetPropertyByIdAsync(int id);
    Task AddPropertyAsync(Property property);
    Task UpdatePropertyAsync(Property property);
    Task DeletePropertyAsync(int id);
}
```

1. **PropertyService.cs**

```
csharp
Copy
// PropertyService.cs
using System.Collections.Generic;
using System.Net.Http;
using System.Net.Http.Json;
using System.Threading.Tasks;
using NexHomeAgentFrontend.Models;

public class PropertyService : IPropertyService
{
    private readonly HttpClient _httpClient;

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

    public async Task<List<Property>> GetPropertiesAsync()
    {
        return await _httpClient.GetFromJsonAsync<List<Proper
ty>>("api/properties");
    }

    public async Task<Property> GetPropertyByIdAsync(int id)
    {
        return await _httpClient.GetFromJsonAsync<Property>
```

```
($"api/properties/{id}");
    }

    public async Task AddPropertyAsync(Property property)
    {
        await _httpClient.PostAsJsonAsync("api/properties", p
roperty);
    }

    public async Task UpdatePropertyAsync(Property property)
    {
        await _httpClient.PutAsJsonAsync($"api/properties/{pr
operty.Id}", property);
    }

    public async Task DeletePropertyAsync(int id)
    {
        await _httpClient.DeleteAsync($"api/properties/{i
d}");
    }
}
```

1. **IChatbotService.cs**

```
csharp
Copy
// IChatbotService.cs
using System.Threading.Tasks;

public interface IChatbotService
{
    Task<string> RespondToMessageAsync(string message);
}
```

1. **ChatbotService.cs**

```
csharp
Copy
// ChatbotService.cs
using System.Threading.Tasks;

public class ChatbotService : IChatbotService
{
    public async Task<string> RespondToMessageAsync(string me
ssage)
    {
// Implement the logic to interact with the AI chatbot servic
e
        return "Hello, I'm the AI chatbot. How can I assist y
ou today?";
    }
}
```

1. **app.css**

```
css
Copy
// app.css
body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-se
rif;
}

.table {
    width: 100%;
    border-collapse: collapse;
}

.table th, .table td {
    border: 1px solid #ddd;
    padding: 8px;
```

```
}

.table th {
    background-color: #f2f2f2;
    text-align: left;
}

.property-details {
    margin: 20px 0;
}

.chatbot-container {
    border: 1px solid #ddd;
    border-radius: 4px;
    padding: 16px;
}

.chatbot-header {
    border-bottom: 1px solid #ddd;
    padding-bottom: 8px;
}

.chatbot-messages {
    max-height: 300px;
    overflow-y: auto;
    padding: 8px 0;
}

.user-message {
    background-color: #f2f2f2;
    padding: 8px;
    margin-bottom: 4px;
}

.chatbot-message {
    padding: 8px;
```

```
    margin-bottom: 4px;
}

.chatbot-input {
    display: flex;
    margin-top: 8px;
}

.chatbot-input input {
    flex-grow: 1;
    margin-right: 8px;
}
```

## Components and Services in the Frontend

MVP Blazer

1. **Startup.cs**

```
// Startup.cs
public void ConfigureServices(IServiceCollection services)
{
    services.AddScoped<IPropertyService, PropertyService>();
    services.AddScoped<IChatbotService, ChatbotService>();
    // Register other services as needed
}
```

1. **PropertyListComponent.razor**

```
// PropertyListComponent.razor
@inject IPropertyService PropertyService

@state class PropertyListState
{
    public List<Property> Properties { get; set; }
```

```
    public string ErrorMessage { get; set; }
    public bool IsLoading { get; set; }
}

<PageTitle>Property List</PageTitle>

<h3>Property List</h3>

@if (state.IsLoading)
{
    <p><em>Loading...</em></p>
}
else if (!string.IsNullOrEmpty(state.ErrorMessage))
{
    <div class="alert alert-danger">@state.ErrorMessage</div>
}
else
{
    <BlazoriseDataGrid TItem="Property" Data="@state.Properti
es" Sortable="true" Filterable="true" PageSize="10" Responsiv
e="true">
        // Grid columns
    </BlazoriseDataGrid>
}

@code {
    private PropertyListState state = new();

    protected override async Task OnInitializedAsync()
    {
        try
        {
            state.IsLoading = true;
            state.Properties = await PropertyService.GetPrope
rtiesAsync();
        }
```

```
        catch (Exception ex)
        {
            state.ErrorMessage = "An error occurred while fet
ching the properties.";
            Logger.LogError(ex, state.ErrorMessage);
        }
        finally
        {
            state.IsLoading = false;
        }
    }
}
```

1. **App.razor**

```
// App.razor
<Router AppAssembly="@typeof(Program).Assembly">
    <Found Context="routeData">
        <RouteView RouteData="@routeData" DefaultLayout="@typ
eof(MainLayout)" />
    </Found>
    <NotFound>
        <LayoutView Layout="@typeof(MainLayout)">
            <p>Sorry, there's nothing at this address.</p>
        </LayoutView>
    </NotFound>
</Router>
```

1. **ChatbotComponent.razor**

```
// ChatbotComponent.razor
@inject IChatbotService ChatbotService

<div class="chatbot-container">
    <div class="chatbot-header">
```

```
        <h4>AI Chatbot</h4>
    </div>
    <div class="chatbot-messages">
        @foreach (var message in messages)
        {
            <div class="@(message.IsFromUser ? "user-message"
: "chatbot-message")">
                <span>@message.Text</span>
            </div>
        }
    </div>
    <div class="chatbot-input">
        <InputText @bind-Value="@userInput" placeholder="Type
your message..." />
        <button @onclick="SendMessage">Send</button>
    </div>
</div>

@code {
    private List<ChatMessage> messages = new();
    private string userInput;

    private async Task SendMessage()
    {
        var userMessage = new ChatMessage { Text = userInput,
IsFromUser = true };
        messages.Add(userMessage);

        var chatbotResponse = await ChatbotService.RespondToM
essageAsync(userInput);
        var chatbotMessage = new ChatMessage { Text = chatbot
Response, IsFromUser = false };
        messages.Add(chatbotMessage);

        userInput = string.Empty;
```

```
    }
}
```

1. **Program.cs**

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

    builder.Services.AddScoped(sp => new HttpClient { BaseAdd
ress = new Uri(builder.HostEnvironment.BaseAddress) });
    builder.Services.AddScoped<IPropertyService, PropertyServ
ice>();
    builder.Services.AddScoped<IChatbotService, ChatbotServic
e>();

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

These files contain the complete Blazor frontend code for the NexHomeAgent AI application, following Microsoft's best practices and standards. You can save each file with the corresponding file name (e.g., `Startup.cs`, `PropertyListComponent.razor`, `App.razor`, `ChatbotComponent.razor`, `Program.cs`) for future reference.